INTERVIEW QUESTIONS

Good collection of questions faced in Architect Level Technical Interviews

INTRODUCTION

Microservices is the latest trend in Technology world. It is the new architecture on which very few books have been written.

If you are aiming to get a job in companies with Microservices architecture like- Netflix, Amazon etc. then this book can help you prepare for the technical interview.

This books also covers Architect level information in Q&A format for easy grasp of the concept.

This book helps you in understanding the deep concepts behind Microservices in a Q&A format.

We keep revising this book from time to time to keep it up to date with the latest changes in Microservices world.

Copyright

Table of Contents

7. WHAT ARE THE CHARACTERISTICS OF A GOOD MICROSERVICE?

8. WHAT IS BOUNDED CONTEXT?

9. WHAT ARE THE POINTS TO REMEMBER DURING INTEGRATION OF MICROSERVICES?

10. IS IT A GOOD IDEA FOR MICROSERVICES TO SHARE A COMMON DATABASE?

11. WHAT IS THE PREFERRED TYPE OF COMMUNICATION BETWEEN MICROSERVICES? SYNCHRONOUS OR ASYNCHRONOUS?

12. WHAT IS THE DIFFERENCE BETWEEN ORCHESTRATION AND CHOREOGRAPHY IN MICROSERVICES ARCHITECTURE?

13. WHAT ARE THE ISSUES IN USING REST OVER HTTP FOR MICROSERVICES?

14. CAN WE CREATE MICROSERVICES AS STATE MACHINES?

15. WHAT IS REACTIVE EXTENSIONS?

16. WHAT IS DRY?

17. WHAT IS SEMANTIC VERSIONING?

18. IS IT A GOOD IDEA TO BUILD A MICROSERVICE OR BUY A COMMERCIAL OFF THE SHELF SOFTWARE?

19. WHY DO WE BREAK THE MONOLITH SOFTWARE INTO MICROSERVICES?

20. WHAT IS CONTINUOUS INTEGRATION?

21. WHAT IS CONTINUOUS DELIVERY?

22. WHAT IS UBIQUITOUS LANGUAGE?

23. WHAT IS THE BENEFIT OF SINGLE SERVICE PER HOST MODEL IN MICROSERVICES?

24. WHAT ARE DIFFERENT TYPES OF TESTS FOR MICROSERVICES?

25. WHAT IS MIKE COHN'S TEST PYRAMID?

26. WHAT IS THE DIFFERENCE BETWEEN MOCK OR STUB FOR MICROSERVICE TESTS?

37. WHAT IS THE DIFFERENCE BETWEEN AUTHENTICATION AND AUTHORIZATION?

38. HOW DOES HTTPS AUTHENTICATION WORKS IN MICROSERVICES?

39. WHAT ARE CLIENT CERTIFICATES?

40. WHY SOME BIG COMPANIES USE API KEYS FOR PROVIDING ACCESS TO PUBLIC APIS?

41. WHAT IS CONFUSED DEPUTY PROBLEM IN SECURITY CONTEXT?

42. HOW CAN WE SECURE DATA AT REST IN AN ORGANIZATION?

43. WHAT ARE THE DIFFERENT POINTS TO CONSIDER FOR SECURITY IN MICROSERVICES ARCHITECTURE?

44. WHAT IS CONWAY'S LAW?

45. WHAT ARE THE IMPORTANT CROSS-FUNCTIONAL REQUIREMENTS TO CONSIDER DURING THE DESIGN OF A

1. 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.

2. What are the benefits of Microservices architecture?

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

a. 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.
b. 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.

c. 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.
d. Reuse: Microservices help you in reusing the lessons learnt from one service to another.
e. 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.

3. 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.

4. 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.

5. 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.

6. 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.

7. What are the characteristics of a Good Microservice?

Good Microservices have these characteristics:

a. **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.

b. **Highly cohesive**: Microservices are highly cohesive so that each one of them can provide one set of behavior independently.
c. **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.
d. **Business Capability**: Microservices individually add business capability that is part of big picture in organization.

8. 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.

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

Some of the important points to remember during integration of Microservices are:

I. **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.

II. **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.

III. **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.

IV. **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.

10. 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.

11. 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.

12. 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.

13. 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.

14. Can we create Microservices as State

the design of a Microservice?

Some of the important cross-functional requirements for design consideration are:

I. Availability
II. Latency
III. Durability of data

46. What is a Circuit Breaker pattern in the context of Microservice?

Circuit Breaker is a software design pattern similar to the one used in electrical devices.

In the context of Microservices, it can be implemented in various scenarios.

If the number of requests to a Microservice increases beyond a

3 highlighters

threshold then circuit breaker becomes active and blocks extra calls.
3 highlighters

If the number of failed requests to a Microservice reach a threshold then circuit breaker becomes active and the requests are throttled for a period of time. After that circuit breaker sends a few requests to check if Microservice has recovered or not.

In Microservices, sometimes circuit breaker stores ad queues up the failed requests for retry at a later time. In certain cases, it is better to fail fast and give immediate response to clients for a synchronous request.

Also circuit breaker can be used to implement Bulkhead pattern in Microservices.

47. What is Bulkhead design pattern?

Bulkhead is a design pattern to create a scalable and stable system. In this pattern, if one part of a system fails, then other parts keep

functioning normally rather than bringing the whole system down.

While designing a system, we create separate components in a watertight compartment like manner. If one component fails, we just close access to that component and try to fix it. And rest of the system keeps working business as usual (BAU).

Netflix's Hysterix library provides two ways to implement Bulkhead design pattern.

I. **Thread Isolation**: In this case, if calls to a component fail then these are directed to thread pool with fixed number of threads.
II. **Semaphore Isolation**: In this case, all callers acquire a permit before making a request. If caller doesn't get permit then it means a component is down.

48. What is Idempotency of a Microservice operation?

Idempotency refers to getting same result even after doing same operation multiple times in a Microservice. Eg. Let say, we have to send a reminder email to a customer for upcoming payment on due date.

Our Microservice provides an idempotent Reminder operation, so that even if we call this operation multiple times on the due date, it sends only one email per day. This will give better experience to customer instead of sending multiple reminders for same payment on same day.

49. How can you scale a Database?

To scale a Database, we have to identify our goals. Do we want to increase the availability of Database or we want to provide better performance for read or write operations on Database. Once the goals are clear, we can use different strategies for different goals.

I. Read Scalability: We can use caching to increase the

performance of reads. Most of the databases have inbuilt features like- index etc to provide better read performance.

II. Write Scalability: One technique to improve write scalability is Sharding. IN this we create multiple database nodes, each catering to one set of data. Eg. If we have two nodes, one node can store products with IDs ending from 1-5 and the other node can store products with IDs from 6-10.

III. Availability: To increase availability, we can create read replicas for the database. Or we can use Cassandra like ring architecture with multiple nodes.

50. What is Command Query Responsibility Segregation (CQRS) design pattern?

Command Query Responsibility Segregation (CQRS) is a design pattern for storing and querying information in which storing of

information can be done in a system separate from the system used for querying the information. It can help us in scaling a system and providing better performance for both reads and writes.

In this, Commands are the requests to modify data in the system. Once commands are validated, data is updated.

Query can work on updated data set and provide results to clients.

Command and Query logic can be implemented in different services that can live on different servers.

In case there is high load on Query service, then you just scale the Query service. Also you can support multiple formats for the response from a Query service to satisfy requirements of multiple clients.

51. How will you implement Caching in Microservice?

Caching is a performance improvement technique for query results from a service. It helps us in minimizing the calls to database for getting results. In the Internet, HTTP protocol also uses caching to serve web pages faster with increasing load.

In Microservices, we can implement Caching in three ways.

I. **Client Side**: In this case, information is cached at client side and service has to inform client when information needs to be replaced in cache. There is very less network traffic in this case.
II. **Proxy Side**: In this case, there is a proxy server between client and server that maintains the cache. It can be used to cache results for multiple services with least overhead.
III. **Server Side**: In this case, server provides caching with tools like Memcache or Redis. It is very opaque to clients. With multiple clients, this is the best strategy for caching.

52. What is CAP theorem?

Eric Brewer published CAP theorem in a paper that states that it is impossible for a distributed computing system to simultaneously provide all three guarantees:

I. **Consistency** (all nodes see the same data at the same time)
II. **Availability** (every request receives a response about whether it succeeded or failed)
III. **Partition tolerance** (the system continues to operate despite arbitrary partitioning due to network failures)

Most of the systems have CP or AP. For a system to run on a network, it needs P i.e. Partition Tolerance. So CA is often ruled out.

Between CP and AP, AP systems are easier to scale. They do not have to worry about Consistency in distributed environment.

53. How will you implement Service Discovery in Microservices architecture?

In Microservices architecture, there are multiple Microservices that provide APIs for different purposes. At times, it is difficult for a Developer to find whether an API already exists or not. If there is already an API provided by another Microservice, then there is no need to reinvent it. There comes the need for Service Discovery. Following are some ways to implement Service Discovery:

I. **DNS**: We can simply using Domain Naming System in such a way that it is intuitive to find a service. You can either use subdomains or a domain-naming template that provides convention for naming services.
II. **Zookeeper**: It is open source software for maintaining configuration information, naming, providing distributed synchronization and providing group services. It has a hierarchical namespace for storing information about services and their configuration.
III. **Consul**: It is another Rest based tool for dynamic service registry. You can use it to register a new service, add health-checks for a service etc.

IV. **Eureka**: Netflix has provided an open source system Eureka that provides load-balancing capabilities in addition to service discovery. It is preferred in a Java environment.

54. What is a good tool for documenting the Microservices?

Swagger is a good tool to document the APIs provided by Microservices. It is an open source tool that is very easy to use. It provides interactive documentation.

You can use Swagger annotation to add documentation in REST APIs and then use Swagger to generate a web interface. By using that Web Interface you can see to list of APIs, their inputs and even invoke these APIs to get results from these APIs.

Swagger UI is a very helpful tool for a newcomer to understand different APIs provided by a Microservice.

55. In which scenarios, implementing Microservices architecture is not a good idea?

Microservices architecture is a great architecture if it is done correctly. To do it correct, you need to think in terms of bounded contexts and create services for each context. If you are new to a domain then you may not have expert level knowledge in that domain to create a bounded context. This can lead to poor choice of bounded context and corresponding Microservices. So you should not try Microservices architecture in a completely new domain.

At times it is a good idea to start with Monolith software and then gradually decompose it into Microservices.

If your organization is new and does not have support for growing number of Microservices, then it is better to go with Monolith software. With time and increased maturity of the organization, you can pick up the Microservices architecture.

56. What are the major principles of Microservices?

Microservices is a newly growing domain, so there is no fixed list of principles of Microservices. Some of the major principles are:

I. **Business Domain Model**: A Microservice implements the functional requirements in a specific business domain that is characterized by a bounded context. It does one specific and does it very well.
II. **Automation**: Often there are multiple Microservices in a system, so it is essential to use automation in deployment, testing and building of software. The more we automate, the more time we save in introducing more Microservices.
III. **Information hiding**: Microservices hide their internal implementation details. This ensures that no assumptions are made within the system about the technology stack used by a service. This information hiding also helps in changing the implementation details at any time.
IV. **Choreography**: Most of the Microservice systems adopt choreography pattern in which there is no central command. Each Microservice behaves like a state machine and it knows how to behave in different events.
V. **Deployment**: Microservices are small parts of software that can be independently deployed. They do not require much downtime like Monolith software.
VI. **Isolation**: Any failure in Microservices architecture can be easily isolated to the specific service that causes it. It can also help in isolating the impact of failure on related services. This can ensure that one failure does not cause complete shutdown.

Thank You