FullStack.Cafe - Kill Your Tech Interview

Q1: Why Would You Opt For Microservices Architecture? ☆☆

Topics: Microservices

Answer:

There are plenty of pros that are offered by Microservices architecture. Here are a few of them:

- Microservices can adapt easily to other frameworks or technologies.
- Failure of a single process does not affect the entire system.
- Provides support to big enterprises as well as small teams.
- Can be deployed independently and in relatively less time.

Q2: List down the advantages of Microservices Architecture ☆☆

Topics: Microservices

Answer:

- **Independent Development**. All microservices can be easily developed based on their individual functionality
- **Independent Deployment**. Based on their services, they can be individually deployed in any application
- **Fault Isolation**. Even if one service of the application does not work, the system still continues to function
- **Mixed Technology Stack**. Different languages and technologies can be used to build different services of the same application
- **Granular Scaling**. Individual components can scale as per need, there is no need to scale all components together

Q3: Define Microservice Architecture ☆☆

Topics: Microservices Software Architecture

Answer:

Microservices, aka ***Microservice Architecture***, is an architectural style that structures an application as a collection of small autonomous services, modeled around a **business domain.**

Q4: What Are The Fundamentals Of Microservices Design? ☆☆☆

Topics: Microservices

Answer:

- Define a scope
- Combine loose coupling with high cohesion
- Create a unique service which will act as an identifying source, much like a unique key in a database table
- Creating the correct API and take special care during integration.

- Restrict access to data and limit it to the required level
- Maintain a smooth flow between requests and response
- Automate most processes to reduce time complexity
- Keep the number of tables to a minimum level to reduce space complexity
- Monitor the architecture constantly and fix any flaw when detected.
- Data stores should be separated for each microservices.
- For each microservices, there should be an isolated build.
- Deploy microservices into containers.
- Servers should be treated as stateless.

Q5: What are the features of Microservices? ☆☆☆

Topics: Microservices

Answer:

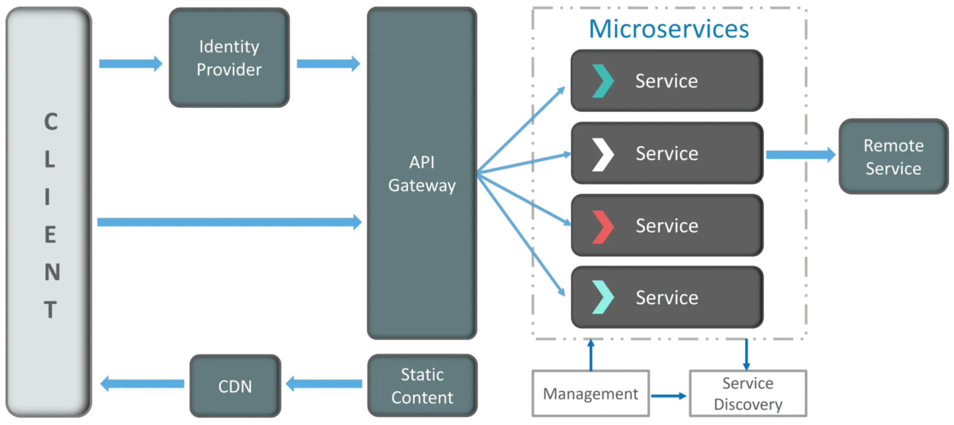
- **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
- **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
- **Business Capabilities** – Microservices are very simple and focus on a single capability
- **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
- **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
- **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
- **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
- **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

Q6: How does Microservice Architecture work? ☆☆☆

Topics: Microservices

Answer:

- **Clients** – Different users from various devices send requests.
- **Identity Providers** – Authenticates user or clients identities and issues security tokens.
- **API Gateway** – Handles client requests.
- **Static Content** – Houses all the content of the system.
- **Management** – Balances services on nodes and identifies failures.
- **Service Discovery** – A guide to find the route of communication between microservices.
- **Content Delivery Networks** – Distributed network of proxy servers and their data centers.
- **Remote Service** – Enables the remote access information that resides on a network of IT devices.



Q7: What is the difference between Monolithic, SOA and Microservices Architecture? ☆☆☆

Topics: Microservices Software Architecture SOA

Answer:

- **Monolithic Architecture** is similar to a big container wherein all the software components of an application are assembled together and tightly packaged.
- A **Service-Oriented Architecture** is a collection of services which communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.
- **Microservice Architecture** is an architectural style that structures an application as a collection of small autonomous services, modeled around a business domain.

Q8: What are the challenges you face while working Microservice Architectures? ☆☆☆

Topics: Microservices

Answer:

Developing a number of smaller microservices sounds easy, but the challenges often faced while developing them are as follows.

- **Automate the Components**: Difficult to automate because there are a number of smaller components. So for each component, we have to follow the stages of Build, Deploy and, Monitor.
- **Perceptibility**: Maintaining a large number of components together becomes difficult to deploy, maintain, monitor and identify problems. It requires great perceptibility around all the components.
- **Configuration Management**: Maintaining the configurations for the components across the various environments becomes tough sometimes.
- **Debugging**: Difficult to find out each and every service for an error. It is essential to maintain centralized logging and dashboards to debug problems.

Q9: How can we perform Cross-Functional testing? ☆☆☆

Topics: Microservices

Answer:

Cross-functional testing is verification of non-functional requirements. These requirements are such characteristics of a system that cannot be implemented like a normal feature. Eg. Number of concurrent users supported by system, usability of site etc.

Q10: What are main differences between Microservices and Monolithic Architecture? ☆☆☆

Topics: Microservices

Answer:

Microservices

- Service Startup is fast
- Microservices are loosely coupled architecture.
- Changes done in a single data model does not affect other Microservices.
- Microservices focuses on products, not projects

Monolithic Architecture

- Service startup takes time
- Monolithic architecture is mostly tightly coupled.
- Any changes in the data model affect the entire database
- Monolithic put emphasize over the whole project

Q11: What are the standard patterns of orchestrating microservices? ☆☆☆

Topics: Microservices

Answer:

As we start to model more and more complex logic, we have to deal with the problem of managing business processes that stretch across the boundary of individual services.

- With **orchestration**, we rely on a central brain to guide and drive the process, much like the conductor in an orchestra. The orchestration style corresponds more to the SOA idea of orchestration/task services. For example we could wrap the business flow in its own service. Where the proxy orchestrates the interaction between the microservices like shown in the below picture.

- With **choreography**, we inform each part of the system of its job, and let it work out the details, like dancers all find- ing their way and reacting to others around them in a ballet. The choreography style corresponds to the dumb pipes and smart endpoints mentioned by Martin Fowler's. That approach is also called the **domain approach** and is using domain events, where each service publish events regarding what have happened and other services can subscribe to those events.

Q12: What are smart endpoints and dumb pipes? ☆☆☆

Topics: Microservices

Answer:

- **Smart endpoints** just meaning actual business rules and any other validations happens behind those endpoints which are not visible to anyone to the consumers of those endpoints think of it as a place where actual Magic happens.

- **Dumb pipelines** means any communication means where no further actions e.g validations are taken place, it simply carries the data across that particular channel and it may also be replaceable if need be. The infrastructure chosen is typically dumb (dumb as in acts as a message router only). It just means that routing is the only function the pipes should be doing.

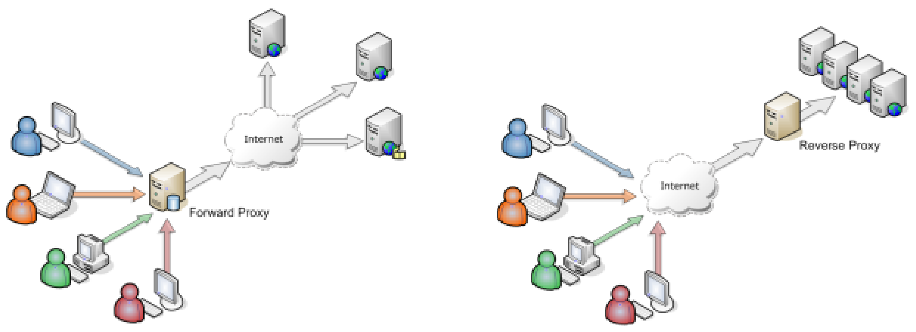
Q13: What is the difference between a proxy server and a reverse proxy server? ☆☆☆

Topics: Microservices

Answer:

A simple definition would be:

- **Forward Proxy**: Acting on behalf of a requestor (or service consumer)
- **Reverse Proxy**: Acting on behalf of service/content producer.



Example:

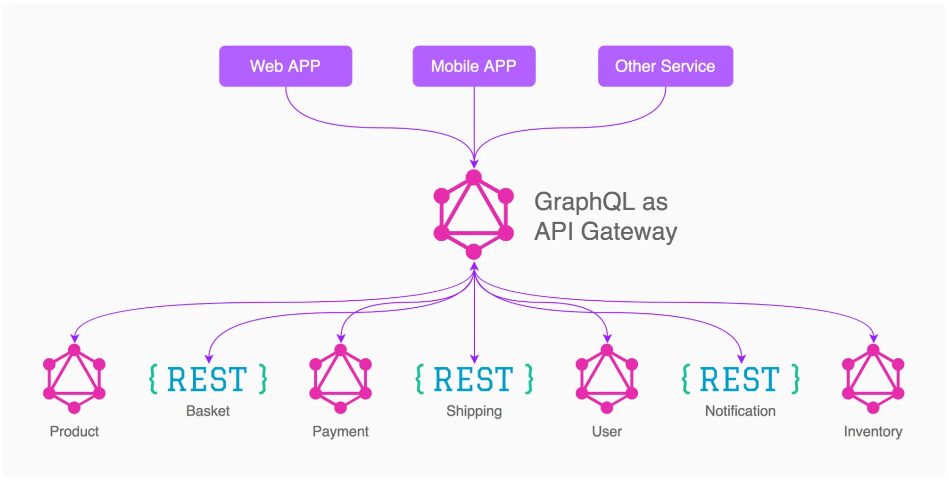
Setting up a proxy in your browser so that Netflix doesn't know what country you're in is a *forward proxy*; an upstream service that directs an incoming request (perhaps you want to send one request to two servers) is a *reverse proxy*.

Q14: Whether do you find GraphQL the right fit for designing *microservice* architecture? ☆☆☆

Topics: Microservices GraphQL

Answer:

GraphQL and microservices are a perfect fit, because GraphQL hides the fact that you have a microservice architecture from the clients. From a backend perspective, you want to split everything into microservices, but from a frontend perspective, you would like all your data to come from a single API. Using GraphQL is the best way I know of that lets you do both. It lets you split up your backend into microservices, while still providing a single API to all your application, and allowing joins across data from different services.



Q15: What are the pros and cons of Microservice Architecture? ☆☆☆☆

Topics: Microservices

Answer:

Pros:

- Freedom to use different technologies
- Each microservices focuses on single capability
- Supports individual deployable units
- Allow frequent software releases
- Ensures security of each service
- Mulitple services are parallelly developed and deployed

Cons:

- Increases troubleshooting challenges
- Increases delay due to remote calls
- Increased efforts for configuration and other operations
- Difficult to maintain transaction safety
- Tough to track data across various boundaries
- Difficult to code between services

Q16: What do you understand by Distributed Transaction? ☆☆☆☆

Topics: Microservices

Answer:

Distributed Transaction is any situation where a single event results in the mutation of two or more separate sources of data which cannot be committed atomically. In the world of microservices, it becomes even more complex as each service is a unit of work and most of the time multiple services have to work together to make a business successful.

Q17: What do you understand by Contract Testing? ☆☆☆☆

Topics: Microservices

Answer:

According to Martin Flower, **contract test** is a test at the boundary of an external service which verifies that it meets the contract expected by a consuming service.

Also, contract testing does not test the behavior of the service in depth. Rather, it tests that the inputs & outputs of service calls contain required attributes and the response latency, throughput is within allowed limits.

Q18: Can we create State Machines out of Microservices? ☆☆☆☆

Topics: Microservices

Answer:

As we know that each Microservice owning its own database is an independently deployable program unit, this, in turn, lets us create a State Machine out of it. So, we can specify different states and events for a particular microservice.

For Example, we can define an Order microservice. An Order can have different states. The transitions of Order states can be independent events in the Order microservice.

Q19: Explain what is the API Gateway pattern ☆☆☆☆

Topics: Microservices API Design

Answer:

An **API Gateway** is a server that is the single entry point into the system. It is similar to the Facade pattern from object-oriented design. The API Gateway encapsulates the internal system architecture and provides an API that is tailored to each client. It might have other responsibilities such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.

A major benefit of using an API Gateway is that it encapsulates the internal structure of the application. Rather than having to invoke specific services, clients simply talk to the gateway.

Q20: How should the various services share a common DB Schema and code? ☆☆☆☆

Topics: Microservices

Answer:

The "purest" approach, i.e. the one that gives you the least amount of coupling, is to *not share any code*. In practice, as usual, it's a tradeoff:

- If the shared functionality is substantial, I'd go for a seperate service.
- If it's just constants, a shared library might be the best solution. You need to be very careful about backwards compatibility, though. Use a packaging system or some source code linkage such as git-tree for distribution.
- For configuration data, you could also implement a specific service, possibly using some existing technology such as LDAP.
- Finally, for simple code that is likely to evolve independently, just duplicating might be the best solution.

Regarding schema - if you want to play by the book, then each microservice has its own database. You don't touch mine, I don't touch yours. That's the better way around this.

FullStack.Cafe - Kill Your Tech Interview

Q1: What is Idempotence? ☆☆☆☆

Topics: Microservices

Answer:

Idempotence refers to a scenario where you perform a task repetitively but the end result remains constant or similar.

Q2: What is the role of an architect in Microservices architecture? ☆☆☆☆

Topics: Microservices

Answer:

An architect in microservices architecture plays the following roles:

- Decides broad strokes about the layout of the overall software system.
- Helps in deciding the zoning of the components. So, they make sure components are mutually cohesive, but not tightly coupled.
- Code with developers and learn the challenges faced in day-to-day life.
- Make recommendations for certain tools and technologies to the team developing microservices.
- Provide technical governance so that the teams in their technical development follow principles of Microservice.

Q3: Mention some benefits and drawbacks of an API Gateway ☆☆☆☆

Topics: Microservices

Answer:

There are some:

- A major benefit of using an API Gateway is that it encapsulates the internal structure of the application. Rather than having to invoke specific services, clients simply talk to the gateway.
- It is yet another highly available component that must be developed, deployed, and managed. There is also a risk that the API Gateway becomes a development bottleneck.
- Developers must update the API Gateway in order to expose each microservice’s endpoints.

Q4: What is Materialized View pattern and when will you use it? ☆☆☆☆

Topics: Microservices

Answer:

Materialized View pattern is the solution for aggregating data from multiple microservices and used *when* we need to implement queries that retrieve data from several microservices. In this approach, we generate, in advance (prepare denormalized data before the actual queries happen), a read-only table with the data that's owned by multiple microservices. The table has a format suited to the client app's needs or API Gateway.

A key point is that a materialized view and the data it contains is completely disposable because it can be entirely rebuilt from the source data stores.

This approach not only solves the problem of *how to query and join across microservices*, but it also improves performance considerably when compared with a complex join, because you already have the data that the application needs in the query table.

Q5: What Did The Law Stated By Melvin Conway Implied? ☆☆☆☆☆

Topics: Microservices

Answer:

Conway’s Law applies to modular software systems and states that:

"Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization’s communication structure".

Q6: Name the main differences between SOA and Microservices? ☆☆☆☆☆

Topics: Microservices SOA

Answer:

- SOA uses Enterprise Service Bus for communication whereas microservices use much simpler messaging systems.
- Each microservice stores data independently while in SOA components share the same storage.
- For microservices, it’s typical to use Cloud while for SOA Application Servers are much more common.
- SOA is still a monolith, in order to make changes, you need to change the entire architecture.
- SOA using only heavy-weight technologies and protocols (like SOAP, etc) whereas microservices is the leaner, meaner, more agile approach (REST/GraphQL).

Q7: What is the difference between Cohesion and Coupling? ☆☆☆☆☆

Topics: Microservices Software Architecture

Answer:

Cohesion refers to what the class (or module) can do. Low cohesion would mean that the class does a great variety of actions - it is broad, unfocused on what it should do. High cohesion means that the class is focused on what it should be doing, i.e. only methods relating to the intention of the class.

As for **coupling**, it refers to how related or dependent two classes/modules are toward each other. For low coupled classes, changing something major in one class should not affect the other. High coupling would make it difficult to change and maintain your code; since classes are closely knit together, making a change could require an entire system revamp.

Good software design has **high cohesion** and **low coupling**.

Q8: What is a Consumer-Driven Contract (CDC)? ☆☆☆☆☆

Topics: Microservices

Answer:

This is basically a pattern for developing Microservices so that they can be used by external systems. When we work on microservices, there is a particular provider who builds it and there are one or more consumers who use Microservice.

Generally, providers specify the interfaces in an XML document. But in Consumer Driven Contract, each consumer of service conveys the interface expected from the Provider.

Q9: What are Reactive Extensions in Microservices? ☆☆☆☆☆

Topics: Microservices

Answer:

Reactive Extensions also are known as Rx. It is a design approach in which we collect results by calling multiple services and then compile a combined response. These calls can be synchronous or asynchronous, blocking or non-blocking. Rx is a very popular tool in distributed systems which works opposite to legacy flows.

Q10: What is the most accepted transaction strategy for microservices? ☆☆☆☆☆

Topics: Microservices Software Architecture

Answer:

Microservices introduce eventual consistency issues because of their laudable insistence on decentralized data management. With a monolith, you can update a bunch of things together in a single transaction. Microservices require multiple resources to update, and distributed transactions are frowned upon (for good reason). So now, developers need to be aware of consistency issues, and figure out how to detect when things are out of sync before doing anything the code will regret.

Think how transactions occur and what kind make sense for your services then, you can implement a rollback mechanism that un-does the original operation, or a 2-phase commit system that reserves the original operation until told to commit for real.

Financial services do this kind of thing all the time - if I want to move money from my bank to your bank, there is no single transaction like you'd have in a DB. You don't know what systems either bank is running, so must effectively treat each like your microservices. In this case, my bank would move my money from my account to a holding account and then tell your bank they have some money, if that send fails, my bank will refund my account with the money they tried to send.

Q11: What does it mean that shifting to microservices creates a run-time problem? ☆☆☆☆☆

Topics: Microservices

Answer:

If you are unable to design a monolith that is cleanly divided into components, you will also be unable to design a microservice system.

Dividing your system into encapsulated, cohesive, and decoupled components is a good idea. It allows you to tackle different problems separately. But you can do that perfectly well in a monolithic deployment (see Fowler: Microservice Premium). After all, this is what OOP has been teaching for many decades! If you decide to turn your components into microservices, you do not gain any architectural advantage. You gain some flexibility regarding technology choice and possibly (but not necessarily!) some scalability. But you are guaranteed some headache stemming from (a) the distributed nature of the system, and (b) the communication between components. Choosing microservices means that you have other problems that are so pressing that you are willing to use microservices despite these problems.

Q12: Why would one use sagas over 2PC and vice versa? ☆☆☆☆☆

Topics: Microservices

Answer:

Here are two approaches which I know are used to implement distributed transactions:

- 2-phase commit (2PC)
- Sagas

2PC is a protocol for applications to transparently utilize global ACID transactions by the support of the platform. Being embedded in the platform, it is transparent to the business logic and the application code as far as I know.

Sagas, on the other hand, are series of local transactions, where each local transaction mutates and persist the entities along with some flag indicating the phase of the global transaction and commits the change. In the other words, state of the transaction is part of the domain model. Rollback is the matter of committing a series of "inverted" transactions. Events emitted by the services triggers these local transactions in either case.

- Typically, 2PC is for *immediate* transactions.
- Typically, Sagas are for *long running* transactions.

I personally consider Saga capable of doing what 2PC can do, but they have the overhead of implementing the redo mechanism. Opposite is not accurate. I think Sagas are universal, while 2PC involves platform/vendor lockdown and lacks platform independence.

Q13: Provide an example of "smart pipes" and "dumb endpoint" ☆☆☆☆☆

Topics: Microservices

Answer:

Components in a system use "pipes" (HTTP/S, queues, etc...) to communicate with each other. Usually these pipes flow through an ESB (Enterprise Service Bus) which does a number of things to the messages being passed between components.

It might do:

- Security checks
- Routing
- Business flow / validation

- Transformation

Once it's completed these tasks the message will be forwarded onto the "endpoint" component. This is an example of "smart pipes" as lots of logic and processing reside inside the ESB (part of the system of "pipes"). The endpoints can then be "dumb" as the ESB has done all the work.

"Smart endpoints and dumb pipes" advocates the opposite scenario. That the lanes of communication should be stripped of business processing and logic and should literally only distribute messages between components. It's then the components themselves that do processing / logic / validation etc... on those messages.