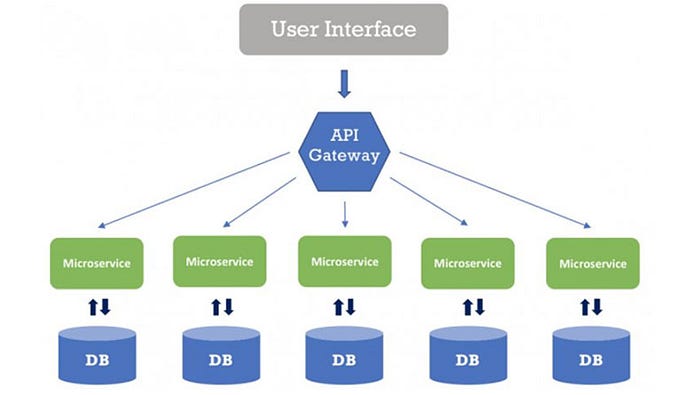
# 50 Microservices Design and Architecture Interview Questions for Experienced Java Programmers

**Common Microservices Interview Questions.**

**What is a Microservices architecture and how does it differ from a monolithic architecture?**

Microservices are small services which provide a specific functionality. For example, the Authentication Service can provide authentication functionality and UserService can provide all the functionality related to Users.

Here is a simple example of how Microservices look like:



**What are the benefits and drawbacks of a Microservices architecture? Pros and Cons?**

Microservices architecture is an approach to building software systems that involves breaking down a monolithic application into a set of small, modular services that can be developed, deployed, and scaled independently. Here are some of the benefits and drawbacks of this approach:

Pros of Microservices:

* **Flexibility**: Microservices architecture allows for flexibility in terms of technology choices, as each service can be implemented using different languages, frameworks, and databases. For example, you can implement one Microservices in Java and other in C++ or Python.
* **Scalability**: Microservices can be scaled independently, which allows for better resource utilization and faster scaling of the overall system. With Cloud computing, Kubernetes can scale Microservices very easily depending upon load.
* **Resilience**: Microservices architecture allows for more fault-tolerant systems, as a failure in one service can be isolated and handled without affecting the entire system.
* **Agility**: Microservices architecture allows for faster development and deployment cycles, as changes can be made to a single service without impacting the entire system.
* **Reusability**: Microservices can be reused across multiple applications, which can result in cost savings and increased efficiency.

**Drawbacks:**

* *Complexity*: Microservices architecture can increase the complexity of the system, as there are more moving parts and more interactions between services.
* *Testing and Debugging*: Testing and debugging a Microservices architecture can be more complex, as it requires testing each service individually, as well as testing their interactions.
* *Monitoring and Management*: Microservices architecture requires more monitoring and management, as there are more services to keep track of and manage.
* *Inter-service communication*: Microservices architecture increases the number of network calls between services, which can lead to increased latency, and if not handled properly, to cascading failures.
* *Security*: Microservices architecture can make it more challenging to implement security measures, as each service may need to be secured individually.

In conclusion, Microservices architecture offers many benefits in terms of flexibility, scalability, and resiliency, but it also increases the complexity of the system and requires more monitoring and management. It’s important to weigh the benefits and drawbacks and choose the right approach that fits the specific requirements and constraints of your system.

**How to design and implement Microservices?**

You can use any framework to develop Microservices in different programming languages but in Java you can use Spring Boot and Spring Cloud to implement Microservices.

**What are the key characteristics of a well-designed Microservice?**  
Well-designed Microservices have clear, single responsibilities, are loosely coupled, have high cohesion, communicate via APIs, have bounded context, and are independently deployable, testable and scalable.

**How to handle testing and continuous integration in a Microservices architecture?**

In Microservices architecture, testing and continuous integration should be done at service level, with automated tests and continuous delivery pipeline for each service. This allows for independent deployment and scaling of services.

**We use Docker and Kubernetes for deploying our Microservices in Cloud.**

Docker is used to create a Docker image of whole service and then Kubernetes to deploy it on AWS or Azure. Service is managed by K8 so it takes care of starting stopped instances and increasing them if load is increased

**How to handle service resiliency in case of failures?**  
K8 does that for you and starts a new Microservice or restart the same one.

**How to ensure that Microservices are loosely coupled and highly cohesive? (Hint — keep it small)**

**How** **does a Java Microservice and .NET Microservice can talk with each other? (Hint Json)**

**How to handle cross-cutting concerns, such as security, in a Microservices architecture?**

**Why is debugging so tough on Microservice Architecture?**

**How to handle data consistency in a Microservices architecture?**

**How do you ensure that Microservices are scalable and resilient?**

**How to handle service discovery and registration in a Microservices architecture?**

**How to handle service communication and data sharing in a Microservices architecture?**

**How to handle service versioning and backward compatibility in a Microservices architecture?**

**How to monitor and troubleshoot Microservices?**

**How to handle deployments and rollbacks in a Microservices architecture?**

**How to handle service governance and lifecycle management in a Microservices architecture?**

**How to handle security and access control in a Microservices architecture?**

**How do you do data integration and data migration in a Microservices architecture?**

**How to handle service composition and orchestration in a Microservices architecture?**

**How do you deploy your Java Microservices?**

**What Java Frameworks can you use to create Microservices?**  
 **hint — Quarkus, Spring Boot, and MicroNaut**

**How many Microservices do you have in your project? How do you find if a user says that one of his orders is missing in database?**

**hint — one database per microservice is a pattern**

**What is API Gateway pattern and how is it used in Microservices architecture? Please explain what problem does it solve and whether its mandatory for Microservices or not?**

API Gateway is one of the essential Microservices patterns which is used to provide a single-entry point for external consumers to access the services. It acts as a reverse proxy and routing layer, which is responsible for request routing, composition, and protocol translation, among other things.

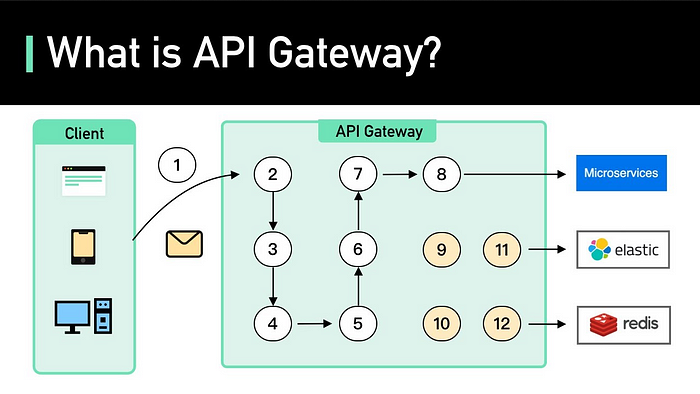
API Gateway pattern solves several problems in Microservices architecture:

* It decouples the external consumers from the internal implementation of the services. This allows the services to evolve and scale independently, without affecting the external consumers.
* It provides a single-entry point for external consumers, which simplifies the client-side service discovery and reduces the number of network calls.
* It can handle cross-cutting concerns such as security, rate limiting, and caching at the edge of the architecture, rather than scattering them across the services.
* It can aggregate multiple services into a single response, reducing the number of network calls and improving the performance of the client-side.
* It can handle protocol and content type translations, allowing the services to be implemented using different protocols and data formats.

**It is not mandatory for Microservices architecture,** but it is commonly used to help manage the complexity and improve the performance of Microservices. It can also be used to provide consistent security, rate limiting, and caching policies across the Microservices.

It’s worth noting that depending on the size of your Microservices environment and the number of requests, it may make sense to have multiple API Gateways in order to distribute the load and improve scalability.

Here is a nice diagram which explains API Gateway design pattern in Microservices:



**Can you Explain Circuit Breaker pattern and how is it used to handle service failures in Microservices architecture? What problem does it solve.**

It improves Service availability. Circuit breaker pattern is a technique used to prevent cascading failures by temporarily preventing further calls to a service that is failing frequently. It helps to improve the resiliency of the system.

**What is Command Query Responsibility Segregation (CQRS) pattern and when is it appropriate to use in Microservices architecture?**

CQRS stands for Command Query Responsibility Segregation. It’s one of the popular Microservices design patterns that separates the read and write operations in a system and allows them to evolve independently. It allows for a more scalable and performant system, but also increases complexity.

**What is the retry pattern in Microservices? When and how to use it?**

**Retry pattern** is a technique used in Microservices architecture to handle service failures automatically. It involves automatically retrying a failed service call a certain number of times, with a delay between retries.

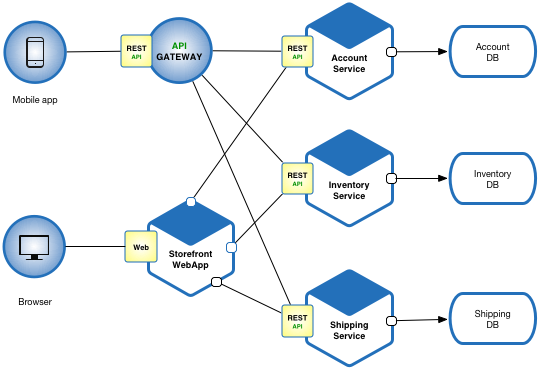
This pattern helps to improve the robustness and resiliency of the system by increasing the chances of a successful call, even in the presence of temporary failures. It’s also used to handle flaky service dependencies gracefully, by retrying the calls to them instead of breaking the whole process and returning an error.

Here are few **Microservices design patterns and principle-based questions** for practice:

1. **Can you Explain Event-Driven pattern and how is it used in Microservices architecture?**
2. **Can you Explain Service Registry pattern and how is it used in Microservices architecture?**
3. **Can you Explain Sidecar pattern and how is it used in Microservices architecture?**
4. **Can you Explain Service Mesh pattern and how is it used in Microservices architecture?**
5. **Can you Explain Backend for Frontend pattern and how is it used in Microservices architecture?**
6. **Can you Explain Bulkhead pattern and how is it used in Microservices architecture?**
7. **What is Saga pattern? What problem does it solve?**
8. **Can you Explain Outbox pattern and how is it used in Microservices architecture?**
9. **What is Self-Containment pattern and how is it used in Microservices architecture?**
10. **Can you Explain External Configuration pattern and how is it used in Microservices architecture?**
11. **What is Strangler pattern and how is it used in Microservices architecture?**

## 15 Advanced Microservices Interview Questions for Experienced Developers

These are more advanced questions on Microservices for experienced Java developers like 5 to 10 years' experience and it covers advanced topics like data replication, data partitioning, Orchestration and service choreography, Security etc.



1. **How to do data partitioning and data replication in MS?**
2. **Have you done any service partitioning and service scaling in a microservices architecture? If not, how can you do it?**
3. **Explain service orchestration and service choreography in microservices?**
4. **What challenges have you faced while developing Microservices in your project?**
5. **How do you handle service security and service encryption in microservices?**
6. **How will you implement service monitoring and service logging in a microservices architecture?**
7. **How do you handle service tracing and service debugging in a microservices architecture?**
8. **What is service testing and service quality assurance in a microservices architecture?**
9. **How do you handle service deployment and service rollback in a microservices architecture?**
10. **How do you handle service governance and service lifecycle management in a Microservices Architecture?**
11. **How do you handle service migration and service modernization in a microservices architecture?**
12. **How do you handle service integration and service API management in a microservices architecture?**
13. **How do you handle service performance and service optimization in a microservices architecture?**
14. **How will you make sure that your Microservices is not affecting other Microservices in the same host?**
15. **How do you organize your Microservices? Does all code remain the same repo or do you create multiple repos for different Microservices?**
16. **What is better? Different database for different Microservice or single database for all Microservices? and why?**