Microservices

1. What are Microservices?

Microservices are a software development approach where an application is broken down into small, independent services that communicate via APIs. Each service focuses on a specific business capability and can be developed, deployed, and scaled independently.

2. What are the key features of Microservices?

- Decentralized Data Management
- Independent Deployment & Scaling
- Lightweight Communication (REST, gRPC)
- Fault Tolerance & Resilience
- Automated Deployment & Monitoring

3. How do microservices differ from monolithic architecture?

In a monolithic architecture, the entire application is developed and deployed as a single unit, often resulting in tight coupling between components. Microservices architecture decomposes the application into smaller, loosely coupled services, each with its own database and independent deployment lifecycle.

4. What are the advantages of Microservices?

- **Scalability** – Each service can be scaled independently.
- **Flexibility in Technology** – Different services can use different tech stacks.
- **Faster Development & Deployment** – Teams can work on individual services without affecting the entire application.
- **Fault Isolation** – Failure in one service doesn't impact others.
- **Better Maintainability** – Easier to update and enhance specific functionalities.

5. How do you handle service discovery in Microservices?

Service discovery is managed using tools like **Eureka**, **Consul**, and **Zookeeper**, which help locate and manage microservices dynamically.

6. What are the challenges of Microservices?

- **Data Management** – Handling distributed data across multiple services.
- **Service Communication** – Managing API calls, latency, and failure handling.
- **Deployment Complexity** – Requires CI/CD pipelines for frequent deployments.
- **Security** – Managing authentication & authorization across multiple services.

7. What is Circuit Breaker Pattern in Microservices?

The **Circuit Breaker Pattern** helps prevent cascading failures by detecting service failures and stopping requests to the failing service until it recovers. **Netflix Hystrix** (deprecated) and **Resilience4j** (recommended) are popular implementations.

8. What is Event-Driven Architecture in Microservices?

In an **Event-Driven Architecture**, microservices communicate asynchronously using events (Kafka, RabbitMQ). This approach reduces coupling, improves performance, and enhances scalability.

9. How do you secure Microservices?

- ✔ **OAuth 2.0 / JWT** – Secure API communication.
- ✔ **API Gateway Security** – Rate limiting, authentication.
- ✔ **Service-to-Service Security** – Mutual TLS, token-based auth.
- ✔ **Role-Based Access Control (RBAC)** – Ensuring proper authorization.

10. What is the difference between Monolithic and Microservices Architecture?

Feature	Monolithic	Microservices
Deployment	Entire app as one unit	Independent deployment per service
Scalability	Limited	Highly scalable
Fault Isolation	Affects entire app	Isolated service failures
Technology Flexibility	Single tech stack	Different technologies per service

11. How does Spring Boot support Microservices development?

Spring Boot simplifies microservices development by providing:

- **Spring Cloud** for distributed systems support
- **Embedded servers** (Tomcat, Jetty) for standalone deployment
- **Auto-configuration** to reduce boilerplate code
- **Actuator** for monitoring microservices
- **Spring Cloud Config** for centralized configuration management

12. What is an API Gateway, and how does it work in Spring Boot Microservices?

An API Gateway is a single-entry point for all microservices, handling:

- ✔ Authentication & Authorization
- ✔ Request Routing
- ✔ Load Balancing
- ✔ Rate Limiting

Spring Cloud Gateway and Netflix Zuul are commonly used API Gateway solutions in Spring Boot.

13. How do you implement inter-service communication in Microservices using Spring Boot?

Inter-service communication can be achieved using:

- **REST API calls** (via RestTemplate or WebClient)
- **gRPC** (for high-performance communication)
- **Message Brokers** (Kafka, RabbitMQ) for asynchronous communication

14. How does Spring Boot handle Distributed Configuration in Microservices?

Spring Boot uses **Spring Cloud Config** to manage centralized configuration.

- The **Spring Cloud Config Server** stores and serves configurations.
- Each microservice fetches configuration dynamically from the server.
- It supports **Git, JDBC, Vault**, and other storage options.

15. What is Service Registration and Discovery in Spring Boot Microservices?

Service discovery helps microservices locate each other dynamically.
Spring Boot supports **Eureka Server** for service registration and discovery.

- Services register themselves with **Eureka Server**.
- Other services discover them dynamically using **Eureka Client**.

16. **How do you handle transactions across multiple microservices in Spring Boot?**
    Microservices don't share databases, so distributed transactions are challenging.
    Possible solutions:

- **SAGA Pattern** (Orchestration or Choreography)
- **Eventual Consistency** using message queues (Kafka, RabbitMQ)
- **Compensating Transactions** for rollback scenarios

17. **How do you monitor Microservices in Spring Boot?**
    Spring Boot provides **Spring Boot Actuator** for monitoring microservices.

- Exposes endpoints like /health, /metrics, /info
- Works with **Prometheus, Grafana, ELK Stack** for real-time monitoring

18. **How does Spring Boot handle Security in Microservices?**
    Spring Security and OAuth 2.0/JWT are used to secure microservices:

- ✔ **OAuth 2.0** – Token-based authentication for secure API access.
- ✔ **JWT (JSON Web Token)** – Stateless authentication with token validation.
- ✔ **API Gateway Security** – Authentication and rate limiting at entry points.

19. **What is Load Balancing in Microservices?**
    Load balancing distributes incoming traffic across multiple service instances.
    Spring Boot supports load balancing using:

- **Spring Cloud LoadBalancer** (recommended in newer versions)
- **Ribbon** (deprecated in Spring Cloud, Spring Cloud LoadBalancer as the recommended alternative)

20. **What is Spring Cloud Sleuth, and how does it help in Microservices?**
    Spring Cloud Sleuth is used for **Distributed Tracing**. It helps track requests across multiple microservices.

- Generates trace IDs for each request.
- Works with **Zipkin** or **Jaeger** for request tracking and debugging.