1. What are microservices?

Microservices is an architectural style that structures an application as a collection of small, autonomous services, modeled around a business domain.
Example: In an e-commerce system:
- User service handles users
- Product service handles product catalog
- Order service handles order management

2. What are the main characteristics of microservices?

- Independently deployable
- Decentralized data management
- Domain-driven design
- Lightweight communication (HTTP/REST, Messaging)
- Failure isolation

3. How is a microservice different from a monolith?

Monolith:
- Single codebase, deploy entire app at once, difficult to scale and isolate failures.
Microservice:
- Multiple small codebases, independent deployments, fault isolation, scalable individually.

4. What is Domain-Driven Design (DDD) and how is it related to microservices?

DDD is a design method where software reflects real-world business domains. Microservices use DDD to define services within bounded contexts. Example: “Order” means different things in Order vs Billing Service.

5. What is a bounded context?

A bounded context is a clearly defined boundary in which a domain model is valid. Different services may have the same terms but different meanings.

6. Why is loose coupling important in microservices?

Loose coupling allows services to change independently, reducing risk of cascading failures and enabling flexible deployment.

7. What is cohesion in microservices?

Cohesion measures how focused a service's responsibilities are. High cohesion means the service does one job well, making it easier to maintain.

8. What are the advantages of microservices?

- Scalability
- Independent deployment
- Fault isolation
- Technology diversity
- Faster time-to-market

9. What are the challenges of microservices?

- Complex inter-service communication
- Data consistency
- Monitoring and logging
- Deployment complexity
- Debugging distributed issues

10. What is Conway’s Law and how does it apply to microservices?

Conway's Law: System designs mirror the communication structures of the organization. Microservices align with team-based structures, each team owning a service.

11. What is the role of a service registry in microservices?

It keeps track of service instances and their locations (host/port). Example tools: Eureka, Consul.

12. What is service discovery?

A mechanism to automatically detect service instances via a registry. Can be client-side (Ribbon) or server-side (Kubernetes).

13. What is the difference between synchronous and asynchronous communication?

- Synchronous: Direct request-response (REST)
- Asynchronous: Messaging-based (Kafka, RabbitMQ), decouples services.

14. What is eventual consistency?

Data consistency is not immediate but guaranteed over time. Used to maintain performance in distributed systems.

15. How do microservices share data?

They typically don’t. Each has its own database and share data via APIs or event-based communication.

16. What are APIs in the context of microservices?

APIs allow services to communicate. REST, gRPC, and GraphQL are common formats.

17. What is an API Gateway?

A single entry point that routes client requests to services, manages auth, rate-limiting, and logging. Examples: Zuul, Spring Cloud Gateway.

18. What is a circuit breaker?

A resilience pattern to stop calls to failing services and prevent cascading failures. Tools: Hystrix, Resilience4j.

19. What is a fallback mechanism?

An alternative response (cached or default) returned when a service is unavailable or fails.

20. What is service orchestration vs choreography?

- Orchestration: Central controller directs flow.
- Choreography: Services react to events independently.

21. What is API versioning?

A strategy to manage changes in APIs while keeping backward compatibility. Example: /v1/orders vs /v2/orders.

22. What is the role of Docker in microservices?

Docker packages services in containers for consistent deployment across environments.

23. What is the role of Kubernetes in microservices?

Kubernetes manages container deployment, scaling, and health monitoring in production environments.

24. What is the 12-Factor App and how does it relate to microservices?

It’s a set of best practices for building scalable apps. Microservices naturally follow many principles like statelessness, config via environment, etc.

25. How do you handle transactions across microservices?

Use distributed patterns like the SAGA pattern or event-driven approaches to maintain consistency.