Focus Areas

- **Why you chose that approach**
- **Trade-offs**
- **Failure scenarios**
- **How Spring Boot / Spring Cloud specifically helps**
- **Real production experience**

Architecture & Design Scenarios

1. You are asked to break a legacy monolith into Spring Boot microservices. How would you identify service boundaries and avoid tight coupling?
2. A service is becoming too large and slow to deploy. How would you decide whether to split it further?
3. Two microservices need the same business logic. Do you duplicate it or extract a common service/library? Why?
4. Your microservices share a common database schema today. How would you refactor to database-per-service with zero downtime?
5. How would you design microservices for both REST and event-driven communication?
6. A new requirement demands rapid feature changes with minimal regression. How would you design for change?
7. How would you implement a Strangler Fig pattern using Spring Boot?
8. How do you manage backward compatibility while evolving APIs?
9. How do you decide between synchronous REST vs asynchronous messaging?
10. How would you design microservices to support multi-tenancy?

Performance & Scalability Scenarios

11. One Spring Boot service experiences high CPU but low traffic. How do you investigate?
12. A downstream service slows down and starts impacting multiple upstream services. How do you isolate the failure?
13. Your APIs are slow only during peak hours. What metrics would you analyze first?
14. How would you implement caching without breaking data consistency?
15. How do you scale stateful services in Spring Boot?
16. What changes would you make when a service must handle 10× traffic overnight?
17. How do you optimize JVM memory and GC for containerized Spring Boot apps?

18. How do you tune database connection pools for high-throughput services?
19. How would you detect and fix thread starvation issues?
20. How do you design APIs to be idempotent under retries?

Resilience & Failure Handling Scenarios

21. A dependent service is intermittently failing. How do you protect your service?
22. When would you use circuit breaker vs retry?
23. How would you design fallback logic without hiding real failures?
24. How do you handle partial failures in a distributed transaction?
25. Your retry mechanism is causing traffic storms. How do you fix it?
26. How would you implement bulkhead isolation in Spring Boot?
27. How do you test circuit breaker behavior in lower environments?
28. What happens if the configuration server goes down?
29. How do you ensure graceful degradation?
30. How do you implement timeout strategies across microservices?

Data Consistency & Transactions

31. A business transaction spans three microservices. How would you maintain consistency?
32. When would you choose Saga orchestration over choreography?
33. How would you implement Saga using Spring Boot?
34. How do you handle duplicate events in event-driven systems?
35. How do you design compensating transactions?
36. How do you handle schema evolution with zero downtime?
37. How do you manage read-after-write consistency?
38. How do you avoid distributed locks?
39. How do you ensure exactly-once processing?
40. How would you debug data inconsistency issues in production?

Security & Compliance Scenarios

41. How would you secure internal service-to-service communication?

42. An API Gateway is compromised. How do you limit blast radius?
43. How do you propagate user identity across microservices?
44. How do you handle token expiration during long-running requests?
45. How would you implement role-based access at service level?
46. How do you secure secrets across environments?
47. How do you handle GDPR/PII data in microservices?
48. How do you audit and trace user actions across services?
49. How do you protect APIs from DDoS attacks?
50. How do you implement Zero-Trust architecture in Spring Boot?

Deployment, DevOps & Cloud Scenarios

51. A deployment caused partial outage. How would you rollback safely?
52. How do you implement blue-green deployment for Spring Boot services?
53. How do you perform canary releases with Kubernetes?
54. How do you handle configuration changes without restarting services?
55. How do you troubleshoot crashes after scaling pods?
56. How do you manage version compatibility during rolling upgrades?
57. What happens if one microservice version is incompatible with others?
58. How do you design health checks for Kubernetes?
59. How do you detect memory leaks in production?
60. How do you design observability (logs, metrics, traces) from day one?
