

Comprehensive list of 300+ Spring Boot interview questions tailored for a **Microservices Backend Engineer** role and , covering all important modules and grouped by topic.

Topics

1. Core Spring & Spring Boot (Beginner to Advanced)
2. Spring Boot Annotations & Configurations
3. Spring Data JPA & Transactions
4. Spring Security (Basic to OAuth2/JWT)
5. Spring Boot Testing (Unit, Integration, Testcontainers)
6. Microservices with Spring Boot (Design, Patterns, Interservice Communication)
7. Spring Cloud Ecosystem (Config, Discovery, Gateway, Resilience4j)
8. Performance, Observability & Production Readiness
9. Advanced Topics (Reactive, Event-Driven, CI/CD Integration)

Let's start one by one

1. Core Spring & Spring Boot (Beginner to Advanced)

1. **What is Spring Framework?** A lightweight Java framework for building enterprise-level applications with dependency injection, AOP, and more.
2. **What is Spring Boot and how is it different from Spring?** Spring Boot simplifies Spring setup with auto-configuration and embedded servers.
3. **What are the advantages of using Spring Boot?** Rapid development, reduced boilerplate, production-ready metrics, and easy integration.
4. **What is dependency injection in Spring?** A design pattern where dependencies are injected by the framework rather than created manually.
5. **What is the difference between @Component, @Service, @Repository?** All are stereotypes; they differ in semantics and layer-specific functionalities like exception translation.
6. **What is ApplicationContext?** The central interface to access Spring beans and configuration.

7. **What is the use of @SpringBootApplication?** It combines @Configuration, @EnableAutoConfiguration, and @ComponentScan.
8. **What is @EnableAutoConfiguration?** It tells Spring Boot to guess and configure beans based on classpath settings.
9. **How does Spring Boot decide what to auto-configure?** It uses `spring.factories` files and conditional annotations like `@ConditionalOnClass`.
10. **What is the role of `application.properties` or `application.yml`?** It holds configuration values like port, DB credentials, etc.
11. **What is a Bean in Spring?** A Java object managed by the Spring IoC container.
12. **How do you define a bean?** Using `@Bean` in a `@Configuration` class or component scanning annotations like `@Component`.
13. **What is the lifecycle of a Spring Bean?** Instantiate → Populate → Initialize → Use → Destroy.
14. **What is @Qualifier used for?** It helps in resolving bean ambiguity when multiple beans of the same type exist.
15. **Difference between Singleton and Prototype scope in Spring?** Singleton creates one instance per Spring container; Prototype creates a new instance each time.
16. **What is the use of @Value annotation?** Injects values from properties or SpEL into Spring beans.
17. **What is Spring AOP?** Aspect-Oriented Programming allows separation of cross-cutting concerns like logging and security.
18. **What are the key modules in Spring?** Core, Context, AOP, JDBC, ORM, Web, Security, etc.
19. **What is circular dependency and how does Spring handle it?** Two or more beans depend on each other. Spring can handle it via setter injection or lazy initialization.
20. **How do you run a Spring Boot application?** Using `SpringApplication.run()` or from the command line using `mvn spring-boot:run` or `./gradlew bootrun`

2. Spring Boot Annotations & Configurations

1. **What is @Configuration?** Marks a class as a source of bean definitions.
2. **What is @Bean?** Declares a bean explicitly within a `@Configuration` class.
3. **What is @ComponentScan?** Tells Spring where to look for components to auto-register.
4. **What is @Component?** A generic stereotype for defining a Spring-managed bean.
5. **What is the difference between @Component and @Bean?** `@Component` is used on classes; `@Bean` is used in config classes for method-level bean creation.
6. **What is @RestController?** Combines `@Controller` and `@ResponseBody`, used for REST APIs.
7. **What is @RequestMapping?** Maps HTTP requests to handler methods.
8. **Difference between @GetMapping and @RequestMapping(method=GET)?** `@GetMapping` is a shortcut for `@RequestMapping(method = RequestMethod.GET)`.
9. **What is @Autowired?** Automatically injects dependencies by type.
10. **What is constructor-based vs field-based injection?** Constructor is preferred for immutability and easier testing; field is less boilerplate but harder to test.
11. **What is @ConditionalOnProperty?** Enables a bean only when a specific property is present or has a given value.
12. **What is @Profile?** Enables beans only for specified active profiles (e.g., dev, prod).
13. **How do you set active profiles in Spring Boot?** Via `application.properties`, command-line args, or environment variables.
14. **What is @PropertySource?** Loads external properties files into the Spring Environment.
15. **What is @ConfigurationProperties?** Binds a POJO to properties from `application.yml/properties`.
16. **What is @EnableConfigurationProperties?** Enables `@ConfigurationProperties` support in Spring Boot.
17. **What is @EnableScheduling?** Enables support for scheduled tasks using `@Scheduled`.

18. **What is @Scheduled?** Schedules method execution at fixed intervals or cron expressions.
19. **What is @EnableAsync?** Enables asynchronous execution using `@Async`.
20. **What is @Async?** Executes a method in a separate thread asynchronously.

3. Spring Data JPA & Transactions

1. **What is Spring Data JPA?** A module that simplifies data access using JPA with minimal boilerplate.
2. **What is JpaRepository?** An interface that provides CRUD and pagination support out of the box.
3. **Difference between CrudRepository and JpaRepository?** `JpaRepository` extends `CrudRepository` and adds features like pagination and sorting.
4. **How do you define a custom query in Spring Data JPA?** Using `@Query` with JPQL or native SQL in a repository method.
5. **What is the use of @Entity?** Marks a class as a JPA entity mapped to a table.
6. **What is @Id?** Marks the primary key of an entity.
7. **What is @GeneratedValue?** Specifies how the primary key is generated (e.g., AUTO, IDENTITY).
8. **What is the N+1 select problem in JPA?** Occurs when lazy-loaded relationships trigger multiple unnecessary queries.
9. **How to solve N+1 problem?** Use `@EntityGraph`, fetch joins, or batch fetching.
10. **What is the default fetch type for @OneToMany and @ManyToOne?**
`@OneToMany`: LAZY, `@ManyToOne`: EAGER.
11. **What is @Transactional?** Manages transaction boundaries declaratively.
12. **Where should @Transactional be placed?** On service layer methods that involve data manipulation.
13. **What is the difference between @Transactional(readOnly = true) and without it?** Read-only optimizes performance for read operations.
14. **Can @Transactional be used on private methods?** No, it works only on public methods due to proxy-based AOP.

15. **How do you handle transactions in nested service calls?** Use `PROPAGATION` settings in `@Transactional`.
16. **What is propagation in Spring transactions?** Defines how transactions relate across method boundaries.
17. **What is isolation in transactions?** Defines how transaction integrity is maintained with concurrent access.
18. **How do you handle optimistic locking in JPA?** Use `@Version` annotation to detect concurrent modifications.
19. **How do you implement pagination in Spring Data JPA?** Use `Pageable` and return type as `Page<T>`.
20. **How do you audit entities in JPA?** Use `@CreatedDate`, `@LastModifiedDate` with `@EntityListeners`.

4. Spring Security (Basic to OAuth2/JWT)

1. **What is Spring Security?** A framework for securing Java applications with authentication, authorization, and protection against common attacks.
2. **What are the key components of Spring Security?**
`AuthenticationManager`, `SecurityFilterChain`, `UserDetailsService`, and `SecurityContext`.
3. **How do you configure HTTP Basic Authentication in Spring Boot?** Using `httpBasic()` in the `SecurityFilterChain` bean.
4. **What is the difference between authentication and authorization?**
Authentication verifies identity; authorization determines access rights.
5. **What is the default security behavior of Spring Boot?** All endpoints are secured with HTTP Basic and in-memory user/password.
6. **How do you create a custom login form in Spring Security?** Override the default login page using `formLogin().loginPage("/login")`.
7. **What is CSRF and how does Spring handle it?** Cross-Site Request Forgery; Spring Security includes CSRF tokens by default.
8. **How do you disable CSRF protection in Spring Security?**
`http.csrf().disable()` – only in stateless APIs.
9. **How do you secure REST APIs in Spring Boot?** Use stateless JWT authentication or OAuth2 with `SecurityFilterChain`.

10. **What is JWT?** JSON Web Token — a stateless, signed token for user identity and claims.
11. **What are the parts of a JWT token?** Header, Payload (Claims), and Signature.
12. **How do you implement JWT-based authentication in Spring Boot?**
Generate a token at login, and use a filter to validate token on every request.
13. **What is the role of OncePerRequestFilter in JWT authentication?** It's a custom filter that validates and sets authentication per request.
14. **How do you store user details for authentication?** Implement `UserDetails` and `UserDetailsService`.
15. **What is PasswordEncoder in Spring Security?** Interface used to encode and match passwords (e.g., BCrypt).
16. **What is @EnableWebSecurity?** Enables Spring Security configuration.
17. **How to implement method-level security in Spring?** Enable with `@EnableMethodSecurity`, then use `@PreAuthorize`, `@Secured`, etc.
18. **What is OAuth2 and how is it different from JWT?** OAuth2 is a protocol for delegated authorization; JWT is a token format.
19. **How do you use Spring Security OAuth2 for login?** Use `spring-boot-starter-oauth2-client` and configure providers like Google.
20. **What is the difference between stateful and stateless security?** Stateful keeps session; stateless uses tokens like JWT with no server-side session.

5. Spring Boot Testing (Unit, Integration, Testcontainers)

1. **What is @SpringBootTest used for?** Loads the full application context for integration tests.
2. **What is the difference between @WebMvcTest and @SpringBootTest?**
`@WebMvcTest` loads only web layer; `@SpringBootTest` loads the full context.
3. **How do you test a controller in isolation?** Use `@WebMvcTest` with `MockMvc`.
4. **What is MockMvc?** A testing utility for simulating HTTP requests in web layer tests.
5. **What is @DataJpaTest?** Loads only JPA components with in-memory DB for testing repositories.

6. **How do you mock beans in Spring tests?** Use `@MockBean` to replace a bean in the test context.
7. **What is `@TestConfiguration`?** Used to define custom beans or overrides only for tests.
8. **How do you test service layer methods?** Use `@ExtendWith(MockitoExtension.class)` and `@InjectMocks`.
9. **What is the role of `TestEntityManager` in `@DataJpaTest`?** Simplifies interaction with JPA for test setups.
10. **How do you test REST endpoints end-to-end?** Use `@SpringBootTest(webEnvironment = RANDOM_PORT) + TestRestTemplate`.
11. **What is `@Transactional` used for in tests?** Rolls back DB changes after each test automatically.
12. **How do you test Kafka or external systems without Docker?** Use embedded brokers or in-memory test implementations.
13. **What are `Testcontainers`?** Java library to run real Docker containers for testing DBs, Kafka, etc.
14. **How do you mock external API calls in Spring Boot tests?** Use `MockRestServiceServer` or `WireMock`.
15. **How do you verify caching behavior in tests?** Use metrics, mock calls, or cache manager introspection.
16. **How do you test retry logic or circuit breakers?** Simulate failures and verify fallback or retry behavior using mocks or test configs.
17. **What is `@EnabledIf` and how is it useful in tests?** Used to conditionally enable test methods based on properties or env.
18. **How to write parameterized tests in Spring Boot?** Use JUnit 5's `@ParameterizedTest` and `@CsvSource` or `@MethodSource`.
19. **How to assert exceptions in Spring tests?** Use `assertThrows` in JUnit or `ExpectedException` rule in JUnit 4.
20. **How to measure test coverage in Spring Boot projects?** Integrate with JaCoCo, SonarQube, or IntelliJ coverage tools.

Perfect! Now diving into one of the most critical sections for microservices backend engineers:

6. Microservices with Spring Boot (Design, Patterns, Interservice Communication)

1. **What are microservices?** Independent, loosely coupled services that handle specific business capabilities.
2. **What are the benefits of microservices architecture?** Scalability, independent deployments, tech flexibility, fault isolation.
3. **How do microservices communicate with each other?** Mostly via REST, gRPC, messaging (Kafka, RabbitMQ).
4. **What is service discovery and why is it needed?** It enables dynamic lookup of service instances; useful in dynamic environments.
5. **What tools can you use for service discovery?** Netflix Eureka, Consul, Zookeeper.
6. **What is Spring Cloud?** A set of tools for building microservices using Spring Boot (e.g., config, discovery, gateway, resilience).
7. **What is Spring Cloud Config?** Centralized configuration server for managing configs across environments.
8. **How do you externalize configuration in Spring Boot microservices?** Use `application.yml`, Spring Cloud Config, environment variables.
9. **How do you implement interservice REST communication in Spring Boot?** Use `RestTemplate` (legacy), `WebClient` (reactive), or Feign client.
10. **What is Spring Cloud OpenFeign?** Declarative REST client for simplified HTTP communication between services.
11. **What is circuit breaker pattern?** Prevents cascading failures by stopping calls to a failing service temporarily.
12. **How do you implement a circuit breaker in Spring Boot?** Using Resilience4j or Spring Cloud Circuit Breaker.
13. **What is retry pattern in microservices?** Retry failed operations automatically with configurable delays.
14. **How to apply retries in Spring Boot?** Use Spring Retry or Resilience4j retry module.
15. **What is rate limiting and how is it done?** Restricts number of requests per time window; use API gateways, bucket/token algorithms.

16. **What is service registry and discovery using Eureka?** Eureka server acts as registry; services register and discover other services dynamically.
17. **How do you implement centralized logging in microservices?** Use ELK stack (Elasticsearch, Logstash, Kibana) or tools like Fluentd + Loki.
18. **What is the role of Spring Cloud Gateway?** Provides routing, load balancing, and security at the edge of the microservice system.
19. **Difference between Spring Cloud Gateway and Zuul?** Gateway is more modern, reactive, and better integrated with Spring 5.
20. **How do you version REST APIs in microservices?** Use URI path (**/v1/**), headers, or content negotiation.
21. **How do you trace requests across microservices?** Use tools like Sleuth (adds trace IDs), Zipkin or OpenTelemetry.
22. **How do you secure microservices communication?** Use mutual TLS, OAuth2, or signed JWT tokens.
23. **What is API Gateway and its role in microservices?** Entry point for clients; handles routing, authentication, rate limiting.
24. **How do you handle partial failures in microservices?** Use fallback mechanisms, circuit breakers, retries, and compensation logic.
25. **What are sagas in microservices?** Distributed transaction pattern; coordinates local transactions with compensating actions.
26. **How do you implement sagas in Spring Boot?** Use event-driven architecture (Kafka + state machine) or orchestration using a coordinator.
27. **What is an orchestrator vs. choreographer in saga pattern?** Orchestrator controls flow; choreographer relies on events and each service reacts.
28. **How do you handle data consistency in microservices?** Use eventual consistency and sagas instead of distributed transactions.
29. **How do you implement async communication between microservices?** Use message brokers like Kafka, RabbitMQ, or Pulsar.
30. **What is idempotency and why is it important in microservices?** Ensures the same operation can be safely repeated without side effects.

Awesome! Now let's dive into messaging — essential for resilient, async microservices.

7. Kafka & Messaging Patterns in Spring Boot Microservices

1. **What is Apache Kafka?** A distributed event streaming platform used for high-throughput messaging between systems.
2. **What are topics and partitions in Kafka?** Topics are categories for messages; partitions provide scalability and parallelism.
3. **How do producers and consumers work in Kafka?** Producers send messages to topics; consumers read messages from topics (by partition).
4. **What is Kafka broker?** A Kafka server that handles incoming messages and manages storage.
5. **What is a Kafka consumer group?** A group of consumers sharing the load of reading messages from a topic.
6. **How do you ensure message ordering in Kafka?** Kafka preserves order within a partition.
7. **How do you integrate Kafka with Spring Boot?** Use `spring-kafka` starter and configure `@KafkaListener`, producer, and consumer factories.
8. **What is `@KafkaListener`?** Annotation to mark a method as a Kafka message consumer.
9. **How do you send messages to Kafka in Spring Boot?** Use `KafkaTemplate.send(topic, message)`.
10. **How do you ensure reliable delivery in Kafka consumers?** Handle `ackMode`, implement retries, and commit offsets manually if needed.
11. **What is offset in Kafka?** A unique ID for each message within a partition, used to track consumption.
12. **How do you handle offset commits in Kafka?** Automatically (default) or manually using `Acknowledgment.acknowledge()`.
13. **What is dead letter topic (DLT)?** A topic where failed messages go for separate handling.
14. **How do you configure DLT in Spring Kafka?** Use `DefaultErrorHandler` with `DeadLetterPublishingRecoverer`.
15. **What is idempotent producer in Kafka?** A producer that ensures no duplicate messages during retries.
16. **How do you handle retries in Spring Kafka consumer?** Use `RetryTemplate` (deprecated) or `DefaultErrorHandler` with backoff.

17. **What is a Kafka schema registry?** A service to manage Avro/JSON/Protobuf schemas for message validation.
18. **How do you serialize/deserialize messages in Kafka?** Use `StringSerializer`, `JsonSerializer`, or Avro with Schema Registry.
19. **How do you ensure message durability in Kafka?** Use `acks=all`, replication, and proper `min.insync.replicas`.
20. **What is the difference between at-most-once, at-least-once, and exactly-once delivery?** — At-most-once: risk of loss — At-least-once: risk of duplicates — Exactly-once: guarantees one delivery (Kafka supports it via idempotence + transactions)
21. **How do you scale Kafka consumers in Spring Boot?** Use concurrency settings or multiple instances with same group ID.
22. **What is the use of Kafka headers?** Carry metadata (like correlation IDs, type info) with messages.
23. **How do you simulate Kafka in tests without Docker?** Use `EmbeddedKafkaBroker` provided by `spring-kafka-test`.
24. **How do you log or trace Kafka messages?** Add interceptors or enrich with Sleuth/Zipkin-compatible trace IDs.
25. **How do you implement request-reply over Kafka?** Use correlation ID headers, dedicated reply topic, and temporary consumers.

8. Spring Boot Advanced Concepts (Async, Caching, Validation, Scheduling, DevOps)

Async & Scheduling

1. **How do you make a method asynchronous in Spring Boot?** Use `@Async` on a method and enable `@EnableAsync`.
2. **What is the return type of an `@Async` method?** `Future`, `CompletableFuture`, or `void`.
3. **How do you schedule tasks in Spring Boot?** Use `@Scheduled` on a method and enable `@EnableScheduling`.
4. **What are different `@Scheduled` options?** `fixedRate`, `fixedDelay`, `cron`.
5. **What is the difference between `fixedRate` and `fixedDelay`?** — `fixedRate`: starts every X ms regardless of previous run - `fixedDelay`: waits for previous execution to finish, then waits delay

Caching

1. **How do you enable caching in Spring Boot?** Add `@EnableCaching` and use `@Cacheable`, `@CachePut`, `@CacheEvict`.
2. **What is `@Cacheable`?** Caches method return value based on key (usually method args).
3. **What is `@CachePut`?** Updates the cache without skipping method execution.]
4. **What is `@CacheEvict`?** Removes entries from the cache.
5. **How do you configure cache manager in Spring Boot?** Use built-in `ConcurrentMapCacheManager` or configure Redis, Ehcache, Caffeine.
6. **What is a custom key generator in caching?** A bean implementing `KeyGenerator` to customize cache keys.
7. **How to disable cache during tests?** Override cache configuration or use profiles to turn off `@EnableCaching`.

Validation

1. **How do you validate user input in Spring Boot?** Use JSR-303 annotations like `@NotNull`, `@Size` with `@Valid`.
2. **How to apply validation to nested objects?** Use `@Valid` inside the parent DTO.
3. **What is the difference between `@Valid` and `@Validated`?** — `@Valid` is JSR-303 (no groups) - `@Validated` supports validation groups from Spring.
4. **How do you handle validation errors globally?** Implement `@ControllerAdvice` with `@ExceptionHandler`.
5. **How to return custom error messages from validation failures?** Use `BindingResult` or customize `MethodArgumentNotValidException`.

DevOps & Configuration

1. **What is Spring Boot Actuator?** Provides production-ready endpoints (health, metrics, env, etc.).
2. **How do you expose actuator endpoints securely?** Use `management.endpoints.web.exposure.include` + security configs.
3. **How to override application properties per environment?** Use profiles like `application-dev.yml`, `application-prod.yml`.

4. **What is the order of property resolution in Spring Boot?** Command-line > Env vars > `application.yml` > Defaults.
5. **How do you use profile-specific beans?** Annotate beans with `@Profile("dev")`, etc.
6. **How do you package a Spring Boot application?** With `spring-boot-maven-plugin` or `spring-boot-gradle-plugin`.
7. **How do you deploy Spring Boot apps?** As a fat JAR, via cloud (AWS/GCP), containerized with Docker.
8. **How do you monitor Spring Boot in production?** Use Actuator + Prometheus + Grafana + distributed tracing (Zipkin, Jaeger).
9. **What is the use of Micrometer in Spring Boot?** Provides metrics collection and export to systems like Prometheus, Datadog.

Absolutely! Here's a focused section on the **Spring Cloud Ecosystem** — a must-know for microservices backend engineers. Let's go deep on **Spring Cloud Config, Eureka (Discovery), Gateway, and Resilience4j**.

9. Spring Cloud Ecosystem (Config, Discovery, Gateway, Resilience4j)

Spring Cloud Config

1. **What is Spring Cloud Config?** A centralized configuration server to manage properties across microservices.
2. **How do you set up a Spring Cloud Config Server?** Use `@EnableConfigServer` and point it to a Git repo or local file system.
3. **How do client services connect to Config Server?** Using `spring.cloud.config.uri` in the client's `bootstrap.yml`.
4. **What's the difference between `bootstrap.yml` and `application.yml`?** `bootstrap.yml` loads first and is used for config server settings.
5. **How do you refresh config properties at runtime?** Use `@RefreshScope` and hit the `/actuator/refresh` endpoint.
6. **What happens if the config server is down?** Clients can fail to start unless `fail-fast=false` is set or fallback configs are used.
7. **How do you secure Spring Cloud Config Server?** Use basic auth, OAuth2, or API Gateway protection.

8. **Can Spring Cloud Config auto-refresh values without endpoint hit?** Yes, via Spring Cloud Bus and message broker (like RabbitMQ/Kafka).

Spring Cloud Discovery (Eureka)

1. **What is Eureka Server?** A registry where services register themselves and discover other services.
2. **How do you create a Eureka server?** Add `@EnableEurekaServer` and appropriate dependencies.
3. **How do you make a Spring Boot service discoverable?** Add `@EnableEurekaClient` and `spring.application.name` in config.
4. **What is the difference between Eureka Client and Service Discovery?** Eureka Client registers itself and uses Eureka to discover others.
5. **What is self-preservation mode in Eureka?** A mode to prevent mass eviction when heartbeat failures spike.
6. **How does Eureka handle service health checks?** Uses built-in or custom health endpoints (from Actuator).
7. **What is instance metadata in Eureka?** Custom key-value data like version, region, etc., associated with an instance.


Spring Cloud Gateway

1. **What is Spring Cloud Gateway?** A non-blocking API Gateway built on Spring WebFlux, supports routing, filtering, rate limiting.
2. **How does routing work in Gateway?** Defined in `application.yml` or Java DSL using route predicates.
3. **What are filters in Spring Cloud Gateway?** Pre/post filters to modify requests/responses (like auth, headers, etc.).
4. **What is a global filter vs route filter?** Global applies to all routes; route filters apply to specific ones.
5. **How do you apply rate limiting in Gateway?** Use `RequestRateLimiter` filter with Redis backend.
6. **How do you secure APIs in Gateway?** With `spring-security` + JWT filter or OAuth2 proxy integration.
7. **Can you load balance routes in Gateway?** Yes, using service discovery (`lb://service-name` syntax).

8. **What is path rewriting in Gateway?** Modify incoming paths before forwarding, using `RewritePath` filter.

Resilience4j (Circuit Breaker, Retry, RateLimiter)

1. **What is Resilience4j?** A fault tolerance library for Java with support for circuit breakers, retries, rate limiting.
2. **How to use Resilience4j in Spring Boot?** Add the dependency and use annotations like `@CircuitBreaker`, `@Retry`, etc.
3. **What is the circuit breaker pattern?** Prevents a system from performing an operation that's likely to fail.
4. **What happens when a circuit breaker is open?** Calls fail immediately, fallback logic can be triggered.
5. **What is the fallback method in Resilience4j?** A method defined to be called when the main one fails.
6. **How do you configure Resilience4j?** Using `application.yml` (e.g., `resilience4j.circuitbreaker.instances.myService`).
7. **What is the retry pattern in Resilience4j?** Automatically retries failed operations a defined number of times.
8. **What is a rate limiter in Resilience4j?** Limits the number of calls in a defined period to protect services.
9. **What is a bulkhead pattern?** Isolates failures and limits concurrent calls to protect the system.
10. **How does time limiter work in Resilience4j?** Times out calls that take too long.
11. **Can Resilience4j be integrated with metrics and tracing?** Yes, integrates well with Micrometer .

Awesome choice! This section is  for **senior backend/microservices engineers**, especially those preparing for **staff or system design interviews**.

10. Performance, Observability & Production Readiness

Performance Tuning

1. **How do you analyze performance bottlenecks in Spring Boot apps?** Use tools like JFR, JProfiler, or VisualVM + enable Actuator metrics.

2. **What is lazy initialization in Spring Boot?** Delays bean creation until it's actually needed using `spring.main.lazy-initialization=true`.
3. **How to reduce startup time of Spring Boot apps?** Enable lazy init, reduce autoconfigurations, use `spring-context-indexer`.
4. **What is the role of `@Indexed` in performance?** Improves component scanning using generated indexes (especially for large apps).
5. **How do you avoid N+1 queries in JPA?** Use `@EntityGraph`, fetch joins (`JOIN FETCH`), or batch fetching.
6. **How do you optimize Hibernate performance?** Enable 2nd level cache, batch inserts/updates, set fetch size, use `@Query` instead of criteria APIs for hot paths.
7. **How do you handle memory leaks in Spring Boot apps?** Use heap dumps (`jmap`), analyze with Eclipse MAT, avoid static references holding Spring beans.
8. **What is garbage collection tuning in Spring Boot?** Set JVM flags (`-XX:+UseG1GC`, `-Xms`, `-Xmx`, etc.) based on load and latency sensitivity.

Observability (Logs, Metrics, Tracing)

1. **What is observability in microservices?** The ability to measure internal state from external outputs (logs, metrics, traces).
2. **What is the difference between monitoring and observability?** Monitoring = alerting on known issues; observability = debugging unknowns.
3. **What are the 3 pillars of observability?** Logs, Metrics, Traces. (*My personal favourite*)
4. **How do you enable logging in Spring Boot?** Logback is default; customize via `logback-spring.xml` or `application.yml`.
5. **How do you structure logs for production?** Use JSON logs, correlation IDs, include service name/version.
6. **How do you trace requests across services?** Use Spring Cloud Sleuth + Zipkin/Jaeger for distributed tracing.
7. **What does Sleuth add to logs?** Trace ID, Span ID, Parent Span ID for tracking request flows.
8. **What is the difference between trace ID and span ID?** Trace ID = end-to-end request; Span ID = individual operation.

9. **How to collect metrics in Spring Boot?** Via Actuator + Micrometer, exporting to Prometheus, Datadog, New Relic, etc.
10. **What are common out-of-box metrics from Actuator?** JVM memory, GC, thread counts, HTTP response times, database pool usage.
11. **How to create custom metrics?** Use Micrometer's [Counter](#), [Timer](#), [Gauge](#), [DistributionSummary](#).
12. **What is the role of tags/labels in metrics?** Add context to metrics (e.g., endpoint, status code, region).

Production Readiness

1. **What is Spring Boot Actuator and why is it important?** Adds endpoints to monitor and manage applications in production.
2. **How do you secure Actuator endpoints?** Restrict with roles, IPs, or API keys; expose only necessary endpoints.
3. **What are critical Actuator endpoints to expose in production?** [/health](#), [/metrics](#), [/prometheus](#), [/info](#), [/loggers](#), [/env](#).
4. **How do you dynamically change logging level in production?** Use [/actuator/loggers/{logger-name}](#) endpoint.
5. **What should go in the [/info](#) endpoint?** Build info, Git commit, app version, environment.
6. **How do you track application deployments?** Use [info.*](#) properties + log Git version and build number on startup.
7. **What is the difference between readiness and liveness probes? -**
Readiness: app is ready to receive traffic. - **Liveness:** app is alive, not hung/crashed.
8. **How to implement readiness and liveness probes in Spring Boot?** Via Actuator endpoints [/actuator/health/readiness](#) and [/liveness](#).
9. **How do you gracefully shutdown Spring Boot apps?** Enable [server.shutdown=graceful](#), release resources in [@PreDestroy](#).
10. **What is connection draining in context of graceful shutdown?** Stop accepting new requests, allow in-flight ones to complete.
11. **How do you manage secrets in production?** Use Vault, AWS Secrets Manager, or K8s secrets (not in properties files).

12. **What is the purpose of Spring Cloud Vault?** Fetches secrets securely from HashiCorp Vault using AppRole or Token auth.
13. **What is chaos engineering and how does it apply to Spring apps?** Injecting failures (e.g., network loss, latency) to test resilience.

11. Advanced Topics (Reactive, Event-Driven, CI/CD Integration)

⚡ Reactive Programming (Spring WebFlux, Project Reactor)

1. **What is reactive programming?** A programming paradigm for asynchronous, non-blocking, event-driven applications.
2. **What is Spring WebFlux?** A reactive web framework introduced in Spring 5 built on Project Reactor.
3. **Difference between Spring MVC and WebFlux?** MVC is synchronous & servlet-based; WebFlux is async & reactive using Netty.
4. **What is Mono and Flux?** - Mono = 0 or 1 item - Flux = 0 to N items (stream)
5. **How does backpressure work in Reactor?** It's a mechanism to control data flow and prevent overwhelming consumers.
6. **What is Schedulers.parallel() vs Schedulers.boundedElastic()?** - parallel() for CPU-bound -boundedElastic() for blocking I/O operations (e.g., DB/file)
7. **How do you test reactive code?** Use StepVerifier from Reactor Test.
8. **How do you call a blocking API in WebFlux?** Wrap it with Mono.fromCallable() and schedule on boundedElastic.
9. **How do you handle exceptions in WebFlux?** Using .onErrorResume(), .onErrorMap() or @ControllerAdvice + WebExceptionHandler.
10. **What is the role of @ResponseStatus and ResponseEntity in WebFlux?** Both are used for returning proper HTTP responses in reactive handlers.
11. **How do you stream real-time data to the client?** Use Flux with MediaType.TEXT_EVENT_STREAM (SSE).
12. **What is the Reactor Context and how is it used?** Thread-local-like context propagation in reactive flows.
13. **Is WebClient thread-safe?** Yes, it's designed to be reused across multiple threads.

14. **How does WebClient differ from RestTemplate?** WebClient is non-blocking and supports reactive streams.

Event-Driven Architecture (EDA, Kafka, Spring Events)

1. **What is event-driven architecture?** System components communicate by emitting and reacting to events.
2. **Difference between event-driven and message-driven?** Event-driven = notify others something happened. Message-driven = send data/command to do something.
3. **What is the role of Kafka in event-driven systems?** Kafka acts as a distributed event log for asynchronous communication.
4. **How do you produce/consume Kafka messages in Spring Boot?** Use `spring-kafka` with `@KafkaListener`, `KafkaTemplate`.
5. **What is idempotency and why is it important in event consumers?** Ensures duplicate events don't cause duplicate state changes.
6. **How to implement exactly-once semantics in Kafka?** Enable idempotent producer + transactional consumer + stateful offset mgmt.
7. **What is an outbox pattern?** Stores domain events in a DB table and publishes them reliably via a background process.
8. **How do you emit domain events in Spring?** Use `ApplicationEventPublisher` and listen via `@EventListener`.
9. **Why is `@TransactionalEventListener` useful?** Fires events *after* transaction commits—guaranteeing consistency.
10. **How do you handle schema evolution in Kafka messages?** Use Avro/Protobuf + Schema Registry.
11. **What is a dead letter topic in Kafka?** A topic to store failed or malformed messages for later inspection.
12. **What are Kafka partitions and why are they important?** Enable parallelism and ordering within partitions.
13. **How do you ensure ordering of events in Kafka?** Use the same partition key to route related events to the same partition.

CI/CD Integration

1. **What are common CI tools used with Spring Boot?** GitHub Actions, GitLab CI, Jenkins, CircleCI, Bitbucket Pipelines.

2. **How do you build a Spring Boot project in CI?** Use Gradle/Maven wrapper with `clean build` and test steps.
3. **What is the purpose of `bootJar` in Gradle?** Creates an executable fat jar with embedded Tomcat/Netty.
4. **How do you run integration tests in CI without Docker?** Use Testcontainers with `@DynamicPropertySource` or in-memory DBs.
5. **How do you create a test coverage report?** Use JaCoCo or Kotlin Kover in the build pipeline.
6. **What is a typical CI pipeline for Spring Boot?** Steps: Checkout → Build → Test → Code Quality → Package → Deploy.
7. **How do you secure secrets in CI pipelines?** Use CI tool's built-in secrets manager or external vault.
8. **What is blue-green deployment and how is it used?** Two production environments (blue & green); switch traffic with zero downtime.
9. **How to perform canary releases for Spring Boot apps?** Deploy to a small % of users/instances, monitor, then roll out further.
10. **How do you deploy Spring Boot apps to Kubernetes?** Build container → push to registry → apply Helm/Manifest files.
11. **How do you manage config in CI/CD across environments?** Use Spring Profiles, Config Server, or environment-specific pipelines.
12. **What is the role of `application-prod.yml` vs `application.yml`?** Overrides base config for production-specific settings.
13. **How do you verify deployment success post-deploy?** Health checks, synthetic monitoring, log analysis, smoke tests.
14. **What's a rollback strategy in CI/CD?** Re-deploy previous stable version or use feature flags to disable behavior.
15. **How do you tag and version your Spring Boot releases?** Use semantic versioning (`1.2.0`), Git tags, and auto-versioning in CI.

Bonus: Architecture & Design Mindset

1. **How do you architect reactive + event-driven microservices?** Use WebFlux for reactive APIs, Kafka for async events, and backpressure-aware consumers.

2. **When NOT to use WebFlux?** When you rely heavily on blocking libraries (JDBC, legacy APIs).
3. **When to use event-driven over REST?** When eventual consistency is acceptable and loose coupling is desired.
4. **What is CQRS and where does it fit in EDA?** Command Query Responsibility Segregation — writes via commands/events, reads via optimized queries.
5. **What is the role of feature flags in production?** Enable/disable functionality without redeploying code.
6. **What is the circuit breaker's place in CI/CD?** It helps apps degrade gracefully during rollout issues.
7. **How do you run chaos tests before going live?** Inject latency/failure using ChaosMonkey4SpringBoot or Gremlin.
8. **What's the difference between rollback and remediation?** Rollback = undo deploy; remediation = fix on-the-fly (hotfix, config).
9. **How do you monitor a canary release safely?** Use separate metrics/alerts, logs, and tracing for that instance group.
10. **How do you ensure release confidence in production?** Combine observability, automated testing, rollback strategies, and staged deployment.

— — -