Spring Framework 5 + Spring Boot 3.x —

Interview Preparation Sections

Section No.	Topic Area	Description	Focus
	Spring Core & IoC	`ApplicationContext`, bean creation, lifecycle, scopes	Foundation
	Dependency Injection (DI)	Constructor/setter injection, circular refs, qualifiers	Core Logic
	Spring Boot Auto-Configuration	`@SpringBootApplication`, `@Conditional*`, custom auto-config	Boot Magic
	Spring Beans & Lifecycle Hooks	`@PostConstruct`, `InitializingBean`, scopes, context refresh	Managed Behavior
	Annotations & Configuration	`@Configuration`, `@ComponentScan`, `@Bean`, `@Profile`	Java Config
	Spring AOP	`@Aspect`, cross-cutting concerns, transaction advice	Modular Logic
	Spring Boot Profiles & External Config	YAML/properties, env var injection, config precedence	Environment Adaptation
	Spring MVC (DispatcherServlet)	Controllers, handler mappings, views, validation	Web Layer
	REST API with Spring Boot	`@RestController`, exception handling, DTOs, HATEOAS	API Design
	Data Access with Spring Data JPA	Repositories, JPQL, projections, pagination	DB Access
	Transaction Management	`@Transactional`, propagation, rollback rules	Data Integrity
	Spring Security Basics	AuthenticationManager, filters, form vs JWT auth	Protection Layer
	Spring Security with JWT/OAuth2	Stateless auth, token validation, Spring Auth Server	Real-World SSO
	Testing Spring Apps	`@SpringBootTest`, `@WebMvcTest`, Test slices	CI-Ready Apps
	Spring Boot Actuator & Monitoring	Health endpoints, custom metrics, Micrometer	Production Insight
	Scheduling & Async	`@Scheduled`, cron, `@Async`, thread pools	Non-blocking Tasks
	Spring Events & ApplicationContextPublisher	Event publishing & listeners, use cases	Decoupled Signals
	Spring WebFlux (Reactive)	Mono, Flux, WebClient, reactive endpoints	High Throughput

Section No.	Topic Area	Description	Focus
	Spring Boot with Kafka/RabbitMQ	Messaging integration, producers/consumers	Event-driven Arch
	Spring Cloud Basics	Config server, Eureka, Feign, gateway filters	Microservices Infra
	Docker & Kubernetes with Spring Boot	Dockerfiles, liveness/readiness, config maps	K8s Deployment
	GraalVM & Native Images (Spring Boot 3)	AOT compilation, startup time optimization	Native Ops

Spring Framework – Section 1: Core & IoC

Question 1

What is the primary purpose of the Spring IoC container?

a) Execute SQL queries
b) Manage object creation and dependencies
c) Create REST endpoints
d) Handle web requests

Answer: b)
Real-World Insight:
In a banking microservice, the IoC container instantiates service classes like `CustomerService`, `AuditService`, and wires them automatically so developers don't manage them manually—supporting loosely coupled design.

Question 2

Which interface is **recommended** for loading Spring configurations in most modern applications?

a) `BeanFactory`
b) `ApplicationContext`
c) `Environment`
d) `ServletContext`

Answer: b)
Real-World Insight:
`ApplicationContext` is feature-rich—it loads property files, internationalization resources, event propagation, and bean post-processing. `BeanFactory` is used for lightweight containers, e.g., IoT or CLI apps.

Question 3

When would you use `@Lazy` annotation?

a) For eagerly loaded beans
b) For deferring bean initialization
c) For disabling autowiring
d) For setting bean scope

Answer: b)
Real-World Insight:
In a reporting module that runs only on weekends, loading the `HeavyReportGenerator` lazily saves memory and start-up time for daily operations.

Question 4

What is true about `@ComponentScan`?

a) It scans only `java.lang`
b) It scans only XML-defined beans
c) It detects annotated beans during startup
d) It disables auto-configuration

Answer: c)
Real-World Insight:
Using `@ComponentScan("com.bank.account")` in a Spring Boot app auto-registers your `@Service`, `@Repository`, and `@Component` beans without writing XML.

Question 5

Which bean scope ensures a new bean instance per HTTP request?

a) `singleton`
b) `request`
c) `session`
d) `prototype`

Answer: b)
Real-World Insight:
In a web banking app, user-specific request data like "TransferAmountValidator" can be request-scoped to avoid shared state issues between users.

Question 6

How does Spring identify which constructor to use in constructor-based injection?

a) Uses reflection on all constructors
b) Uses the constructor with most parameters

c) Uses constructor annotated with `@Autowired`
d) Uses the default constructor only

Answer: c)
Real-World Insight:
This ensures explicitness and avoids ambiguity when a class has multiple constructors (e.g., `UserService(String name)` and `UserService(String name, int id)`).

Question 7

What happens if two beans of the same type are found and no `@Qualifier` is used?

a) Spring randomly picks one
b) Spring throws `NoSuchBeanDefinitionException`
c) Spring throws `NoUniqueBeanDefinitionException`
d) Spring logs a warning

Answer: c)
Real-World Insight:
When you have two payment providers, `PayPalService` and `StripeService`, and no qualifier is specified, Spring cannot resolve ambiguity and fails to start.

Question 8

What does `@Primary` do?

a) Creates multiple beans
b) Hides a bean from DI
c) Designates the default bean to use
d) Makes the bean abstract

Answer: c)
Real-World Insight:
If both `H2DataSource` and `PostgresDataSource` are defined, mark one `@Primary` for default database in dev/staging environments.

Question 9

Which of the following is **not** a valid bean scope in Spring?

a) `singleton`
b) `session`
c) `thread`
d) `prototype`

Answer: c)
Real-World Insight:

Spring supports web-aware scopes like `session`, but `thread` is not built-in (requires customization via `Scope` interface).

Question 10

Which of the following is best suited for **global singleton utilities**?

a) `prototype`
b) `request`
c) `singleton`
d) `session`

Answer: c)
Real-World Insight:
Global audit loggers like `AuditTrailService` or system-wide configuration beans are typically singletons.

Question 11

Which of the following annotations enables annotation-based configuration in Spring?

a) `@EnableConfig`
b) `@SpringBean`
c) `@Configuration`
d) `@EnableAnnotationConfig`

Answer: c)
Real-World Insight:
Marking a class with `@Configuration` is essential to define `@Bean` methods. For example, configuring a custom `PasswordEncoder` bean in a `SecurityConfig`.

Question 12

Which method is called just after the bean is initialized?

a) `destroy()`
b) `init()`
c) `afterPropertiesSet()`
d) `onStart()`

Answer: c)
Real-World Insight:
Implement `InitializingBean` and override `afterPropertiesSet()` to perform validation or logging when bean properties are set, such as checking if a URL or credentials are initialized properly.

Question 13

Which lifecycle method can be annotated to run custom logic after initialization?

a) `@AutoInit`
b) `@PostInit`
c) `@InitBinder`
d) `@PostConstruct`

Answer: d)
Real-World Insight:
You might want to validate DB connections or load in-memory data at startup with `@PostConstruct`. It's more concise than implementing interfaces.

Question 14

What is the **default** scope of a Spring bean?

a) `prototype`
b) `singleton`
c) `request`
d) `session`

Answer: b)
Real-World Insight:
A singleton service like `UserService` is initialized once and reused across the application, reducing memory usage.

Question 15

How do you create a bean without annotations?

a) Add it in `application.properties`
b) Define it using `@EnableBean`
c) Declare it in an `@Configuration` class using `@Bean`
d) Use `BeanFactory.create()`

Answer: c)
Real-World Insight:
For third-party classes like `ObjectMapper`, you can't annotate them directly. Use `@Bean` inside `@Configuration` to wire them in.

Question 16

Why would you use `@ComponentScan(basePackages = {"com.app.service"})`?

a) To scan XML config
b) To disable scanning
c) To locate annotated components in the given packages
d) To override all existing beans

Answer: c)
Real-World Insight:
This lets you isolate scanning to relevant domains, e.g., separating internal and external modules in a large-scale e-commerce platform.

Question 17

What is the purpose of `BeanPostProcessor`?

a) Handle exceptions
b) Intercept bean creation process before/after initialization
c) Modify HTTP responses
d) Replace Spring Security filters

Answer: b)
Real-World Insight:
You might log or wrap beans (like `DataSource`) using proxies without changing the original bean class.

Question 18

When does Spring throw a `BeanCurrentlyInCreationException`?

a) When `@Lazy` bean is misconfigured
b) When a prototype bean has no constructor
c) When a circular dependency exists between singleton beans
d) When bean is not defined

Answer: c)
Real-World Insight:
This commonly occurs when services A → B → A form a loop. You can break it using setter injection or `@Lazy`.

Question 19

How does `@Qualifier` help in Spring DI?

a) Makes bean definition optional
b) Specifies which bean to inject when multiple of same type exist
c) Caches bean for reuse
d) Hides beans from being scanned

Answer: b)

Real-World Insight:

When you have both `mongoCustomerRepo` and `sqlCustomerRepo`, you use `@Qualifier("mongoCustomerRepo")` to pick the right one in your service.

Question 20

Which annotation combination will enable Spring to auto-detect beans and register them?

a) `@Bean` + `@Configuration`
b) `@Component` + `@ComponentScan`
c) `@RestController` + `@EnableWebMvc`
d) `@EnableAutoConfig` + `@Repository`

Answer: b)

Real-World Insight:

`@ComponentScan` tells Spring where to look for `@Component`, `@Service`, `@Repository`, and other stereotype annotations.

Question 21

If a bean is marked `@Scope("prototype")`, what happens on each injection?

a) Singleton object reused
b) New object is created each time
c) Request object is injected
d) Only works in XML config

Answer: b)

Real-World Insight:

Useful for stateful objects like `PdfExporter` or `ReportBuilder` which should not be shared.

Question 22

Which of these will run **before** `@PostConstruct`?

a) Constructor
b) `afterPropertiesSet()`
c) `@PreDestroy`
d) `@Autowired` field injection

Answer: a)

Real-World Insight:

The constructor is always first, followed by dependency injection, then `@PostConstruct`.

Question 23

Which approach is better for unit testable DI?

a) Field injection (`@Autowired`)
b) Constructor injection
c) Manual bean lookup
d) Static bean initialization

Answer: b)
Real-World Insight:
Constructor injection is preferred for immutability and testability. You can easily mock dependencies via constructor args in JUnit/Mockito.

Question 24

What is the use of `Environment` object in Spring?

a) It gives access to OS variables
b) It allows bean lifecycle modification
c) It provides access to property values & profiles
d) It logs Spring exceptions

Answer: c)
Real-World Insight:
You can use it to fetch dynamic values from property files or environment variables like AWS region or DB URLs.

Question 25

Which of the following allows multiple beans of the same class but different names?

a) `@Bean(name="a", name="b")`
b) `@ComponentScan`
c) Define them with different method names in `@Configuration` class
d) Spring doesn't support this

Answer: c)
Real-World Insight:
You can define `@Bean public DataSource mysqlDataSource()` and `@Bean public DataSource oracleDataSource()` in the same config for multiple DBs.

Question 11

Which of the following annotations enables annotation-based configuration in Spring?

a) `@EnableConfig`
b) `@SpringBean`

c) `@Configuration`
d) `@EnableAnnotationConfig`

Answer: c)
Real-World Insight:
Marking a class with `@Configuration` is essential to define `@Bean` methods. For example, configuring a custom `PasswordEncoder` bean in a `SecurityConfig`.

Question 12

Which method is called just after the bean is initialized?

a) `destroy()`
b) `init()`
c) `afterPropertiesSet()`
d) `onStart()`

Answer: c)
Real-World Insight:
Implement `InitializingBean` and override `afterPropertiesSet()` to perform validation or logging when bean properties are set, such as checking if a URL or credentials are initialized properly.

Question 13

Which lifecycle method can be annotated to run custom logic after initialization?

a) `@AutoInit`
b) `@PostInit`
c) `@InitBinder`
d) `@PostConstruct`

Answer: d)
Real-World Insight:
You might want to validate DB connections or load in-memory data at startup with `@PostConstruct`. It's more concise than implementing interfaces.

Question 14

What is the **default** scope of a Spring bean?

a) `prototype`
b) `singleton`
c) `request`
d) `session`

Answer: b)

Real-World Insight:

A singleton service like `UserService` is initialized once and reused across the application, reducing memory usage.

Question 15

How do you create a bean without annotations?

a) Add it in `application.properties`
b) Define it using `@EnableBean`
c) Declare it in an `@Configuration` class using `@Bean`
d) Use `BeanFactory.create()`

Answer: c)

Real-World Insight:

For third-party classes like `ObjectMapper`, you can't annotate them directly. Use `@Bean` inside `@Configuration` to wire them in.

Question 16

Why would you use `@ComponentScan(basePackages = {"com.app.service"})`?

a) To scan XML config
b) To disable scanning
c) To locate annotated components in the given packages
d) To override all existing beans

Answer: c)

Real-World Insight:

This lets you isolate scanning to relevant domains, e.g., separating internal and external modules in a large-scale e-commerce platform.

Question 17

What is the purpose of `BeanPostProcessor`?

a) Handle exceptions
b) Intercept bean creation process before/after initialization
c) Modify HTTP responses
d) Replace Spring Security filters

Answer: b)

Real-World Insight:

You might log or wrap beans (like `DataSource`) using proxies without changing the original bean class.

Question 18

When does Spring throw a `BeanCurrentlyInCreationException`?

a) When `@Lazy` bean is misconfigured
b) When a prototype bean has no constructor
c) When a circular dependency exists between singleton beans
d) When bean is not defined

Answer: c)
Real-World Insight:
This commonly occurs when services A → B → A form a loop. You can break it using setter injection or `@Lazy`.

Question 19

How does `@Qualifier` help in Spring DI?

a) Makes bean definition optional
b) Specifies which bean to inject when multiple of same type exist
c) Caches bean for reuse
d) Hides beans from being scanned

Answer: b)
Real-World Insight:
When you have both `mongoCustomerRepo` and `sqlCustomerRepo`, you use `@Qualifier("mongoCustomerRepo")` to pick the right one in your service.

Question 20

Which annotation combination will enable Spring to auto-detect beans and register them?

a) `@Bean` + `@Configuration`
b) `@Component` + `@ComponentScan`
c) `@RestController` + `@EnableWebMvc`
d) `@EnableAutoConfig` + `@Repository`

Answer: b)
Real-World Insight:
`@ComponentScan` tells Spring where to look for `@Component`, `@Service`, `@Repository`, and other stereotype annotations.

Question 21

If a bean is marked `@Scope("prototype")`, what happens on each injection?

a) Singleton object reused
b) New object is created each time
c) Request object is injected
d) Only works in XML config

Answer: b)
Real-World Insight:
Useful for stateful objects like `PdfExporter` or `ReportBuilder` which should not be shared.

Question 22

Which of these will run **before** `@PostConstruct`?

a) Constructor
b) `afterPropertiesSet()`
c) `@PreDestroy`
d) `@Autowired` field injection

Answer: a)
Real-World Insight:
The constructor is always first, followed by dependency injection, then `@PostConstruct`.

Question 23

Which approach is better for unit testable DI?

a) Field injection (`@Autowired`)
b) Constructor injection
c) Manual bean lookup
d) Static bean initialization

Answer: b)
Real-World Insight:
Constructor injection is preferred for immutability and testability. You can easily mock dependencies via constructor args in JUnit/Mockito.

Question 24

What is the use of `Environment` object in Spring?

a) It gives access to OS variables
b) It allows bean lifecycle modification
c) It provides access to property values & profiles
d) It logs Spring exceptions

Answer: c)
Real-World Insight:

You can use it to fetch dynamic values from property files or environment variables like AWS region or DB URLs.

Question 25

Which of the following allows multiple beans of the same class but different names?

a) `@Bean(name="a", name="b")`
b) `@ComponentScan`
c) Define them with different method names in `@Configuration` class
d) Spring doesn't support this

Answer: c)
Real-World Insight:
You can define `@Bean public DataSource mysqlDataSource()` and `@Bean public DataSource oracleDataSource()` in the same config for multiple DBs.

Section 2: ***Dependency Injection (DI)***

Topics: Constructor vs Setter injection, circular dependencies, field injection, and qualifiers — with real-world-based explanations.

Question 1

Which form of injection is considered **best for immutability and testing**?

a) Field injection
b) Setter injection
c) Constructor injection
d) Static injection

Answer: c)
Real-World Insight:
Constructor injection makes it easier to use mocking frameworks like Mockito in unit tests and ensures that dependencies are not optional, as in the case of services like `TransactionProcessor`.

Question 2

Which annotation is commonly used for **qualifying multiple beans** of the same type?

a) `@Primary`
b) `@Qualifier`

c) `@Autowired`
d) `@BeanName`

Answer: b)
Real-World Insight:
When you have multiple data sources (e.g., MySQL and PostgreSQL), `@Qualifier("mysqlDataSource")` tells Spring which one to inject.

Question 3

What happens when Spring detects a **circular dependency** between two beans using **constructor injection**?

a) It automatically resolves it
b) It fails at startup
c) It skips one constructor
d) It chooses field injection

Answer: b)
Real-World Insight:
For example, if `ServiceA` needs `ServiceB` in its constructor and vice versa, Spring throws `BeanCurrentlyInCreationException`. You can fix it with setter injection or `@Lazy`.

Question 4

How does Spring resolve circular dependencies when using **setter injection**?

a) Via proxy objects
b) By reordering class files
c) Using `@Primary`
d) It doesn't support it

Answer: a)
Real-World Insight:
Setter injection is more flexible for resolving circular references because Spring can create the objects first and then call the setters to inject dependencies later.

Question 5

Which injection is generally discouraged due to **tight coupling and poor testability**?

a) Constructor injection
b) Setter injection
c) Field injection
d) Bean method injection

Answer: c)

Real-World Insight:

Field injection like `@Autowired private EmailService emailService;` makes testing harder and violates DI principles — Mockito or other test tools can't easily pass mock objects into private fields.

Question 6

What does `@Primary` do in DI?

a) Prevents multiple bean creation
b) Denotes preferred bean if multiple candidates are found
c) Marks a bean as read-only
d) Disables bean proxying

Answer: b)

Real-World Insight:

You can mark `@Primary` on your `LocalEmailSender` implementation so it gets injected unless `@Qualifier` is explicitly used elsewhere.

Question 7

If you declare `@Autowired(required = false)` on a field, what behavior are you expecting?

a) Fail if bean not found
b) Throw runtime exception
c) Ignore if bean not found
d) Inject all beans of that type

Answer: c)

Real-World Insight:

Useful in plugin architectures where optional beans like `CustomValidator` may or may not be present depending on the deployment.

Question 8

Which is **true** for setter-based injection?

a) Dependencies must be final
b) Allows optional dependencies
c) Runs before bean instantiation
d) Doesn’t support `@Autowired`

Answer: b)

Real-World Insight:

Setter injection lets you configure dependencies like audit loggers or retry policies that are optional based on environment.

Question 9

Which annotation resolves **ambiguity** when multiple beans of the same type exist?

a) `@Default`
b) `@Qualifier`
c) `@Autowired`
d) `@BeanName`

Answer: b)
Real-World Insight:
If you inject `PaymentService` and both `StripeService` and `RazorPayService` are beans, use `@Qualifier("stripeService")`.

Question 10

If both `@Primary` and `@Qualifier` are present, which one takes precedence?

a) `@Primary`
b) `@Qualifier`
c) Whichever appears first
d) Both are ignored

Answer: b)
Real-World Insight:
If `@Primary` is set on a bean, but a `@Qualifier` is explicitly specified at the injection site, Spring will respect the qualifier and override the primary.

Question 11

Which of the following is true about `@Autowired` on a constructor in Spring Boot 3?

a) It is optional if there's only one constructor
b) It must be declared explicitly
c) It is deprecated
d) It doesn’t work with Lombok

Answer: a)
Real-World Insight:
Spring Boot 3 infers constructor injection automatically if there’s a single constructor — great for clean code in service-layer beans.

Question 12

What does `@Inject` belong to?

a) Spring Core
b) Java EE / JSR-330
c) Jakarta Annotations
d) Kotlin DI

Answer: b)
Real-World Insight:
`@Inject` works like `@Autowired` but is part of the JSR-330 standard. You may see this in portable Java projects or frameworks like Dagger.

Question 13

Which DI method is best for **optional dependencies**?

a) Constructor injection
b) Setter injection
c) Field injection
d) Static injection

Answer: b)
Real-World Insight:
You might use setter injection for `EmailNotifier` which is used only if email notifications are enabled via config.

Question 14

You want to inject a list of all beans of a certain type. What should you do?

a) Use `@InjectMany`
b) Use `@Autowired List<MyType>`
c) Use `@Qualifier("all")`
d) Create a factory method

Answer: b)
Real-World Insight:
When developing `RuleEngine` where multiple `RuleEvaluator` implementations exist, injecting `List<RuleEvaluator>` lets Spring inject all beans of that interface.

Question 15

In Spring Boot, when using constructor injection with **Lombok**, which annotation should be used?

a) `@AllArgsConstructor`
b) `@Data`

c) `@RequiredArgsConstructor`
d) `@Builder`

Answer: c)
Real-World Insight:
`@RequiredArgsConstructor` automatically generates a constructor for all final fields, allowing Spring Boot to perform constructor-based DI cleanly.

Question 16

What does Spring inject when a bean implements an interface with multiple implementations **but no `@Primary` or `@Qualifier`** is provided?

a) The first one in alphabetical order
b) A proxy bean
c) Throws `NoUniqueBeanDefinitionException`
d) All available beans

Answer: c)
Real-World Insight:
If you define both `InMemoryUserRepo` and `PostgresUserRepo` and inject just `UserRepository`, Spring fails unless you use `@Qualifier` or `@Primary`.

Question 17

What will happen if `@Autowired` is applied on a **private setter**?

a) Spring ignores it
b) Spring injects it via reflection
c) Application fails to start
d) Setter must be public

Answer: b)
Real-World Insight:
Spring uses reflection internally and can inject even into private/protected setters—used often in secured services or legacy components.

Question 18

What happens if you declare `@Autowired` on two constructors?

a) Spring injects both
b) Spring throws a startup error
c) Spring injects the no-arg constructor
d) Spring fails with ambiguous constructor exception

Answer: d)

Real-World Insight:

If `CustomerService(String name)` and `CustomerService(String name, int age)` both are annotated, Spring cannot choose — this is a design flaw unless resolved using `@Autowired(required=false)` or `@Primary`.

Question 19

What is the main advantage of using `@ConfigurationProperties` over `@Value`?

a) Can inject values only from XML
b) Supports property validation and grouping
c) Requires fewer annotations
d) Used only in Spring WebFlux

Answer: b)

Real-World Insight:

A `@ConfigurationProperties(prefix="mail")` bean groups SMTP settings like host, port, credentials in a single POJO — great for testability and structure.

Question 20

What will happen if no bean is available and `@Autowired` is used without `required = false`?

a) Bean is created at runtime
b) Null is injected
c) Spring throws `NoSuchBeanDefinitionException`
d) Spring skips injection

Answer: c)

Real-World Insight:

For non-optional services like `PaymentProcessor`, a missing bean should break startup to avoid runtime failures.

Question 21

What's the best practice to **conditionally inject a bean** based on active profile?

a) Use `@Lazy`
b) Use `@Profile("dev")` on the bean
c) Use a separate class
d) Use `@DependsOn`

Answer: b)

Real-World Insight:

You may want to use `MockSMSService` only in `dev`, and real provider in `prod` — `@Profile` enables that segregation cleanly.

Question 22

What does Spring do when injecting a dependency through a constructor annotated with `@Autowired(required=false)`?

a) Always inject
b) Throw error if bean missing
c) Skip constructor if dependency not found
d) Treat as optional and inject `null`

Answer: d)
Real-World Insight:
This lets you avoid manually writing overloaded constructors just for optional injection (e.g., optional `AuditLogger`).

Question 23

Which annotation combination ensures a bean is used only in specific environments?

a) `@Autowired + @Qualifier`
b) `@Primary + @Profile`
c) `@Profile + @Component`
d) `@DependsOn + @Component`

Answer: c)
Real-World Insight:
E.g., you might want `@Component @Profile("cloud")` for AWS-specific beans and exclude them in local/dev.

Question 24

Why is **constructor injection** considered more robust than **field injection**?

a) Supports static fields
b) Allows circular injection
c) Supports final fields and immutability
d) Requires less configuration

Answer: c)
Real-World Insight:
You can’t inject final fields via setters or field injection. Constructor injection is the only way to keep them immutable.

Question 25

What happens if a bean's dependency is a prototype-scoped bean injected into a singleton-scoped bean?

a) Prototype bean is recreated on each call
b) Singleton holds the same instance forever
c) Spring throws error
d) Injection is skipped

Answer: b)
Real-World Insight:
To ensure dynamic behavior (e.g., random token generator), inject via `ObjectProvider<TokenGenerator>` to get a new instance every time.

⚙ Section 3: Spring Boot Auto-Configuration

Key Topics: `@SpringBootApplication`, `@EnableAutoConfiguration`, `@Conditional*`, `@ConfigurationProperties`, and custom auto-config — with real-world insights.

Question 1

What does `@SpringBootApplication` combine?

a) `@Component`, `@Service`, `@Bean`
b) `@ComponentScan`, `@Configuration`, `@EnableAutoConfiguration`
c) `@Controller`, `@RestController`, `@Autowired`
d) `@EnableScheduling`, `@EnableAsync`, `@EnableAutoConfiguration`

Answer: b)
Real-World Insight:
This is typically placed on your main class `MyApp.java` to bootstrap everything — component scanning, bean registration, and config loading — with a single annotation.

Question 2

What is the role of `@EnableAutoConfiguration`?

a) Enables component scanning
b) Triggers bean initialization explicitly
c) Automatically configures beans based on classpath & properties
d) Starts Tomcat server manually

Answer: c)
Real-World Insight:

If Spring Boot detects `spring-boot-starter-data-jpa` and a datasource property, it auto-configures `EntityManager`, `TransactionManager`, and `DataSource` without explicit bean setup.

Question 3

Which annotation allows enabling a bean **only when a property exists with a specific value**?

a) `@ConditionalOnMissingBean`
b) `@EnableAutoConfig`
c) `@ConditionalOnProperty`
d) `@ComponentScan`

Answer: c)
Real-World Insight:
You can enable a `DevEmailSender` bean only if `app.mail.enabled=true` using:

```
java
CopyEdit
@ConditionalOnProperty(name="app.mail.enabled", havingValue="true")
```

Question 4

What does `@ConditionalOnMissingBean` do?

a) Registers a fallback bean if the specified bean does not exist
b) Prevents bean registration
c) Fails the build
d) Checks for null fields

Answer: a)
Real-World Insight:
It’s used often in libraries to provide defaults — e.g., `DefaultAuditLogger` will only load if the app hasn't defined its own logger.

Question 5

Which annotation is used to define **external configuration** for a POJO in Spring Boot?

a) `@PropertySource`
b) `@Value`
c) `@ConfigurationProperties`
d) `@AppPropertyConfig`

Answer: c)
Real-World Insight:
You can map `application.yaml` properties to a `MailProperties` class like:

```
java
CopyEdit
@ConfigurationProperties(prefix = "app.mail")
```

Question 6

What is the correct file where auto-configuration classes are listed in Spring Boot?

a) `spring.factories`
b) `spring.properties`
c) `application.yml`
d) `beans.xml`

Answer: a)
Real-World Insight:
Libraries register their auto-configs using `META-INF/spring.factories`. E.g., `spring-boot-autoconfigure` lists `DataSourceAutoConfiguration`.

Question 7

Which class is automatically used by Spring Boot to start a web application?

a) `AppStarter`
b) `WebServerFactoryCustomizer`
c) `SpringApplication.run()`
d) `AppLauncher.run()`

Answer: c)
Real-World Insight:
The entry point for Spring Boot apps is:

```
java
CopyEdit
public static void main(String[] args) {
    SpringApplication.run(MyApp.class, args);
}
```

Question 8

What does `@ConditionalOnClass` do?

a) Registers a bean when the class is present in the classpath
b) Checks if a field is null
c) Skips configuration
d) Autowires the class directly

Answer: a)
Real-World Insight:

Used in custom starters — you may auto-configure MongoDB beans only if `com.mongodb.MongoClient` is on the classpath.

Question 9

Why use `@ConfigurationProperties` over `@Value`?

a) `@Value` cannot load environment variables
b) `@ConfigurationProperties` supports complex hierarchies and validation
c) `@Value` is slower
d) `@ConfigurationProperties` allows injecting @Autowired beans

Answer: b)
Real-World Insight:
Use `@ConfigurationProperties` when working with nested configurations like SMTP, credentials, retry policies — all grouped in a single POJO.

Question 10

How can you **exclude** a particular auto-configuration?

a) `@EnableAutoConfiguration(exclude=...)`
b) `@SpringBootApplication(scanBasePackages=...)`
c) Set `exclude-autoconfig=true` in `application.properties`
d) Use `@ComponentScan(exclude=...)`

Answer: a)
Real-World Insight:
Exclude `DataSourceAutoConfiguration.class` if you're not using a database:

```
java
CopyEdit
@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })
```

Question 11

What is a **common use** of `@ConditionalOnBean`?

a) Load a bean if a particular bean is missing
b) Load a bean only if another specific bean exists
c) Load all beans of a type
d) Exclude autoconfig completely

Answer: b)
Real-World Insight:
Suppose you register a `KafkaRetryListener` only if a `KafkaTemplate` bean is present — use `@ConditionalOnBean(KafkaTemplate.class)`.

Question 12

When does Spring evaluate `@Conditional*` annotations?

a) During runtime, at every method call
b) At build time
c) During bean creation, before initialization
d) After application starts

Answer: c)
Real-World Insight:
Spring processes these conditions during context initialization. It's helpful for skipping expensive beans (e.g., `ElasticsearchClient`) unless needed.

Question 13

Which is **true** about writing custom auto-configurations?

a) Only works with XML configuration
b) Must be annotated with `@Configuration` and declared in `spring.factories`
c) Can only be used in starter projects
d) Doesn’t support conditionals

Answer: b)
Real-World Insight:
A custom starter like `feature-toggle-spring-boot-starter` may define `FeatureToggleConfig` with `@ConditionalOnProperty` and list it in `META-INF/spring.factories`.

Question 14

Which Spring Boot annotation helps create modular startup logic based on **Java class presence**?

a) `@ConditionalOnProperty`
b) `@ConditionalOnClass`
c) `@AutoInject`
d) `@ClasspathCheck`

Answer: b)
Real-World Insight:
If you're building a messaging integration, you can conditionally create `JmsTemplate` beans if `javax.jms.ConnectionFactory` is present.

Question 15

Which of the following annotations can help conditionally load beans in **specific Spring profiles**?

a) `@ConditionalOnProfile`
b) `@Profile("dev")`
c) `@BeanProfile`
d) `@EnvCondition`

Answer: b)
Real-World Insight:
Using `@Profile("prod")` ensures a `ProdSecurityConfig` loads only in production and not in local/dev.

Question 16

How would you register a fallback service only if the primary service is not present?

a) `@EnableAutoConfiguration`
b) `@ConditionalOnMissingBean`
c) `@FallbackBean`
d) `@PrimaryFallback`

Answer: b)
Real-World Insight:
If a user doesn't define a `MetricsPublisher`, your library can register a default `ConsoleMetricsPublisher`.

Question 17

What does `@ConditionalOnExpression` allow?

a) Write profile-based YAML blocks
b) Create beans based on SpEL expressions
c) Prevent cyclic dependency
d) Schedule tasks

Answer: b)
Real-World Insight:
You can load a bean conditionally using dynamic conditions:

```
java
CopyEdit
@ConditionalOnExpression("'${env.name}' == 'test'")
```

Question 18

How is Spring Boot's auto-configuration prioritized?

a) Alphabetical order
b) Depends on class name

c) Based on `@Order` annotations
d) Using `AutoConfigureOrder` or `@AutoConfigureBefore/After`

Answer: d)
Real-World Insight:
In custom starters, you may want `LoggingAutoConfig` to run after `BasicAuditConfig`.
Use:

```
java
CopyEdit
@AutoConfigureAfter(BasicAuditConfig.class)
```

Question 19

Which file is used by Spring Boot to register auto-configuration classes for discovery?

a) `beans.xml`
b) `application.yml`
c) `spring.factories`
d) `boot.autoconfig`

Answer: c)
Real-World Insight:
Inside the JAR under `META-INF/spring.factories`, declare:

```
properties
CopyEdit
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.example.autoconfig.MyAutoConfig
```

Question 20

What is the purpose of the `@SpringBootConfiguration` annotation?

a) Marks the app as web-bootable
b) Overrides `application.properties`
c) Meta-annotation used in place of `@Configuration` in main class
d) Starts Spring Cloud Config

Answer: c)
Real-World Insight:
`@SpringBootApplication` uses it internally. You rarely use it directly unless writing test configurations or starter libraries.

Question 21

Why does `@EnableAutoConfiguration` scan the classpath?

a) To load XML config files
b) To auto-register beans from dependencies
c) To clean temp files
d) To load Swagger UI

Answer: b)
Real-World Insight:
If your classpath includes `spring-boot-starter-security`, Spring automatically configures security filters unless overridden.

Question 22

How do you prevent auto-configuration of a specific class in a test?

a) Set `enabled=false` in properties
b) Use `@SpringBootTest(disableAutoConfig=true)`
c) Use `@ImportAutoConfiguration(exclude=...)`
d) Use `@MockAutoConfiguration`

Answer: c)
Real-World Insight:
For tests, if you don't want `SecurityAutoConfiguration`, exclude it:

```
java
CopyEdit
@ImportAutoConfiguration(exclude = SecurityAutoConfiguration.class)
```

Question 23

Which conditional lets you write **multi-class checks**?

a) `@ConditionalAllClasses`
b) `@ConditionalOnClass`
c) `@ConditionalAllPresent`
d) `@ConditionalImports`

Answer: b)
Real-World Insight:
You can write:

```
java
CopyEdit
@ConditionalOnClass({ DataSource.class, EntityManager.class })
```

to ensure both are present before initializing a JPA config.

Question 24

Which annotation is used **alongside** `@ConfigurationProperties` to register the config bean?

a) `@EnableAutoConfiguration`
b) `@RegisterPropertyBean`
c) `@EnableConfigurationProperties`
d) `@ScanConfigProperties`

Answer: c)
Real-World Insight:
If you define `AppConfigProperties`, add:

```
java
CopyEdit
@EnableConfigurationProperties(AppConfigProperties.class)
```

to register it for injection.

Question 25

What's a good real-world reason to create custom auto-configuration?

a) Save memory
b) Abstract environment-specific logic for reuse
c) Avoid Spring Boot dependency
d) Avoid YAML files

Answer: b)
Real-World Insight:
E.g., you can provide reusable `KafkaAuditLoggerAutoConfig` in internal libraries and use `@ConditionalOnProperty` to activate it per project.

Spring Beans & Lifecycle Hooks

Topics: `@PostConstruct`, `InitializingBean`, Bean Scopes, Lifecycle Callbacks, Context Events — with real-world, interview-grade reasoning.

Question 1

When is `@PostConstruct` executed in the Spring lifecycle?

a) Before dependency injection
b) Right after bean is instantiated and dependencies are injected
c) Before constructor
d) After context is closed

Answer: b)

Real-World Insight:

It’s ideal for initializing connections or logging startup metadata:

```
java
CopyEdit
@PostConstruct
public void init() {
    logger.info("Audit service initialized");
}
```

Question 2

What interface should a bean implement to run logic **after properties are set**?

a) `DisposableBean`
b) `ContextRefreshedListener`
c) `InitializingBean`
d) `Runnable`

Answer: c)

Real-World Insight:

You might use `afterPropertiesSet()` in `KafkaProducerInitializer` to register topics once all connection properties are injected.

Question 3

What happens if both `@PostConstruct` and `afterPropertiesSet()` are defined?

a) Only one runs
b) Spring throws error
c) Both run — `@PostConstruct` first
d) Only `@PostConstruct` runs in Boot 3.x

Answer: c)

Real-World Insight:

Spring guarantees that `@PostConstruct` runs **before** `afterPropertiesSet()` — useful when integrating legacy interfaces + annotation-based logic.

Question 4

Which annotation or mechanism runs **before bean destruction**?

a) `@PreDestroy`
b) `afterPropertiesSet()`
c) `@BeanClose`
d) `@BeanShutdown`

Answer: a)

Real-World Insight:

Use this to release resources like:

```
java
CopyEdit
@PreDestroy
public void shutdown() {
    executorService.shutdownNow();
}
```

Question 5

Which **scope** creates a new bean instance **every time** it is requested?

a) `@Singleton`
b) `@RequestScoped`
c) `@SessionScoped`
d) `@Scope("prototype")`

Answer: d)

Real-World Insight:

Useful for creating new `ReportGenerator` or `PdfBuilder` instances on each request.

Question 6

Which Spring event is fired **after the application context has been fully initialized**?

a) `ContextClosedEvent`
b) `ContextStartedEvent`
c) `ContextRefreshedEvent`
d) `AppBootstrapEvent`

Answer: c)

Real-World Insight:

If you need to preload caches, warm-up services, or trigger analytics tracking, listen to:

```
java
CopyEdit
@EventListener(ContextRefreshedEvent.class)
```

Question 7

In Spring Boot, what is the **default bean scope**?

a) `@Request`
b) `@Singleton`

c) `@Prototype`
d) `@Application`

Answer: b)
Real-World Insight:
Every bean is a singleton unless otherwise configured, which simplifies wiring services like `EmailSender`, `UserService`.

Question 8

How do you inject a prototype-scoped bean into a singleton?

a) Regular `@Autowired` works
b) Use `ObjectFactory<T>` or `Provider<T>`
c) Use `@InjectPrototype`
d) Spring doesn't allow it

Answer: b)
Real-World Insight:
Injecting directly causes the prototype to be created only once. Use:

```
java
CopyEdit
@Autowired
private ObjectProvider<AuditToken> auditTokenProvider;
```

Question 9

Which lifecycle callback is **more flexible** and preferred in newer apps?

a) `InitializingBean`
b) `@PostConstruct`
c) `Constructor`
d) `afterInit()`

Answer: b)
Real-World Insight:
It works even if you don't control the bean class (e.g., via XML config or external library) and supports clean separation.

Question 10

How do you listen for Spring Context shutdown events?

a) Implement `DisposableBean`
b) Annotate a method with `@OnShutdown`

c) Use `@EventListener(ContextClosedEvent.class)`
d) Spring does not support this

Answer: c)
Real-World Insight:
You can gracefully close DB pools, flush metrics, or log:

```
java
CopyEdit
@EventListener
public void onShutdown(ContextClosedEvent e) { ... }
```

Question 11

What's the correct order of lifecycle methods in Spring?

a) Constructor → `@PostConstruct` → `afterPropertiesSet()` → custom init-method
b) Constructor → `afterPropertiesSet()` → `@PostConstruct`
c) `@PostConstruct` → Constructor → `afterPropertiesSet()`
d) Constructor → `afterPropertiesSet()` → destroy()

Answer: a)
Real-World Insight:
Spring calls constructors first, then injects dependencies, runs `@PostConstruct`, and finally `afterPropertiesSet()` — often used to register post-injection logic.

Question 12

What happens to a `@PostConstruct` method in a `@Component` **when using JUnit tests** with `@MockBean` on dependencies?

a) It's skipped
b) It still runs
c) It causes an exception
d) Only if bean is singleton

Answer: b)
Real-World Insight:
In `@SpringBootTest`, mocked dependencies are injected before `@PostConstruct` — useful to stub external APIs while still verifying initialization.

Question 13

What interface allows you to define custom shutdown logic?

a) `ApplicationContextListener`
b) `DisposableBean`
c) `LifecycleAwareBean`
d) `ShutdownHook`

Answer: b)

Real-World Insight:

Use `destroy()` method in `DisposableBean` for legacy cleanup — e.g., stop message queues, threads, release file handles.

Question 14

Which of the following is a **customizable bean lifecycle hook** defined in XML or JavaConfig?

a) `@Bean(lifecycle="init")`
b) `@PostCreate(init)`
c) `@Bean(initMethod="init", destroyMethod="shutdown")`
d) `@Configure(lifecycle="onLoad")`

Answer: c)

Real-World Insight:

Used for third-party beans where you can't annotate the class:

```
java
CopyEdit
@Bean(initMethod="init", destroyMethod="close")
public DataSource myDS() { ... }
```

Question 15

Why is using `@PreDestroy` preferred over `DisposableBean`?

a) Can be used on fields
b) Allows multiple destroy methods
c) Decouples lifecycle logic from interfaces
d) Auto-wires cleanup logic

Answer: c)

Real-World Insight:

`@PreDestroy` works with POJOs and gives clean separation. Interface-based shutdown (`DisposableBean`) tightly couples your logic.

Question 16

Which is **not** a valid Spring bean scope?

a) `singleton`
b) `request`
c) `refresh`
d) `session`

Answer: c)

Real-World Insight:

Valid scopes include `singleton`, `prototype`, `request`, `session`, and `application`. `refresh` is not recognized.

Question 17

How do you initialize **application-level caches or background services** after the app has started?

a) `@PostConstruct`
b) `@EventListener(ContextRefreshedEvent.class)`
c) `afterPropertiesSet()`
d) `@Schedule(init=true)`

Answer: b)
Real-World Insight:
Use event-based triggers for asynchronous startup tasks like loading user roles or fetching reference data.

Question 18

When injecting a `prototype` bean into a `singleton`, what will happen **by default**?

a) A new bean is injected each time
b) Spring throws exception
c) A single instance is injected at creation
d) Spring reuses the prototype on each injection

Answer: c)
Real-World Insight:
To get new instances each time, inject via `ObjectFactory<T>` or `Provider<T>`.

Question 19

What's the **biggest drawback** of using `InitializingBean` and `DisposableBean`?

a) Lack of validation
b) Slower startup
c) Tight coupling to Spring lifecycle
d) Doesn't support external configs

Answer: c)
Real-World Insight:
These interfaces force your code to depend on Spring — not ideal for library or modular reuse.

Question 20

Which listener lets you **perform an action after all beans are created** but before they are used?

a) `BeanFactoryPostProcessor`
b) `ApplicationRunner`
c) `CommandLineRunner`
d) `SmartInitializingSingleton`

Answer: d)
Real-World Insight:
It runs after singleton beans are instantiated but before the application starts accepting requests — useful for validation, audit checks, etc.

Question 21

What scope would you use to ensure a **new object per HTTP request**?

a) `@Scope("request")`
b) `@Scope("singleton")`
c) `@RequestBean`
d) `@Scope("session")`

Answer: a)
Real-World Insight:
For REST APIs where `RequestLogger`, `RequestContext` are needed, use `request` scope.

Question 22

Which interface would you use to **react to container startup or shutdown events**?

a) `ApplicationEventPublisher`
b) `ApplicationContextAware`
c) `ApplicationListener<>`
d) `LifecycleEventAware`

Answer: c)
Real-World Insight:
Create beans like:

```
java
CopyEdit
public class MyListener implements ApplicationListener<ContextStartedEvent> {
    public void onApplicationEvent(...) { ... }
}
```

Question 23

What does the `@Lookup` annotation do?

a) Lookup fields by type
b) Forces prototype-scoped method to return new bean every time

c) Allows access to context beans
d) Injects singleton only

Answer: b)
Real-World Insight:
Useful for getting a fresh bean even inside a singleton:

```
java
CopyEdit
@Lookup
public ReportGenerator getReportGenerator() { ... }
```

Question 24

Which hook is best used to **verify external dependencies (e.g., DB or Kafka)** are ready?

a) `afterPropertiesSet()`
b) `@PostConstruct`
c) `@EventListener(ContextRefreshedEvent.class)`
d) `@BeanReady`

Answer: c)
Real-World Insight:
You can ping APIs or databases only after the entire context is loaded to avoid startup errors.

Question 25

What's the main difference between `@PostConstruct` and constructor logic?

a) `@PostConstruct` executes after constructor + dependency injection
b) Constructor executes last
c) Both are equivalent
d) `@PostConstruct` doesn't need Spring context

Answer: a)
Real-World Insight:
Constructor can't use `@Autowired` fields reliably. Use `@PostConstruct` if your logic depends on injected services.

Spring Annotations & Configuration

Focus: `@Configuration`, `@ComponentScan`, `@Bean`, `@Profile`, JavaConfig vs XML
Use-case: Java-based bean registration, conditional configuration, and modularization.

Q1. What does `@Configuration` denote in a Spring application?

a) Creates a REST controller
b) Marks a class as a component
c) Indicates a class contains bean definitions
d) Auto-configures JPA settings

Explanation:
`@Configuration` is used to define beans in JavaConfig. The methods annotated with `@Bean` inside such classes register objects in the Spring container.

Real-world: You'd typically place `@Configuration` in a core config file like `AppConfig.java`, managing services, datasources, or caches without using XML.

Q2. How does `@Component` differ from `@Configuration`?

a) Not picked up during scanning
b) Configuration supports `@Bean` methods
c) Loads external XML
d) Must extend an interface

Explanation:
`@Component` is a generic stereotype for autodetected beans. `@Configuration` is special — it not only defines a bean but enhances it (via CGLIB proxying).

Real-world: You'd use `@Component` for classes like `EmailService`, and `@Configuration` to group beans like `RestTemplate` or external API configs.

Q3. What is the use of `@ComponentScan`?

a) Loads web views
b) Scans for `@Entity` classes
c) Scans for classes annotated with `@Component`, `@Service`, `@Repository`, etc.
d) Configures filters

Explanation:
This annotation tells Spring where to look for annotated beans in the specified package(s).

Real-world: In microservice structure, you may restrict scanning to `com.mybank.services` only to reduce startup time and bean collisions.

Q4. How is `@Bean` different from `@Component`?

a) Only used in Spring Boot
b) `@Bean` registers manually-created beans, `@Component` is autodetected
c) Both are identical
d) Only works on interfaces

Explanation:
`@Bean` methods are used when you need programmatic logic during bean creation, often for 3rd-party libs.

Real-world: Use `@Bean` for `ObjectMapper`, `RestTemplate`, or `ThreadPoolExecutor` where constructor args need custom setup.

Q5. Is this configuration valid?

```
java
CopyEdit
@Configuration
@ComponentScan(basePackages = "com.example")
```

a) Valid
b) Only works with XML
c) Requires `@EnableComponentScan`
d) Runtime error

Explanation:
A Java-based configuration class like the above is sufficient to enable Spring scanning.

Real-world: This is how you bootstrap services in a typical Spring Boot app using manual configurations.

Q6. How to conditionally load a bean in production only?

a) `@ComponentScan("prod")`
b) `@Profile("prod")`
c) `@EnvOnly("prod")`
d) `@SpringBootProfile("prod")`

Explanation:
`@Profile` ensures a bean only loads for the active profile.

Real-world: You may use separate datasources: H2 for `dev`, MySQL for `prod`. This avoids production config from leaking into lower environments.

Q7. Where can you set the active profile?

a) `pom.xml`
b) `application.yml`
c) `beans.xml`
d) `logback.xml`

Explanation:
Spring Boot looks for `spring.profiles.active` in properties/yaml files.

Real-world:

```
yaml
CopyEdit
spring:
  profiles:
    active: staging
```

— often customized per environment (CI, QA, prod).

Q8. What happens if two beans of same type exist without qualifiers?

a) Picks randomly
b) Application crashes
c) Injects both
d) Throws `NoUniqueBeanDefinitionException`

Explanation:
Spring cannot resolve ambiguity unless you use `@Primary` or `@Qualifier`.

Real-world: Useful when you have two `DataSource` beans — one for reads, one for writes.

Q9. What does `@Import` do?

a) Loads CSS
b) Imports another Spring config class
c) Adds external JARs
d) Finds `@Service` classes

Explanation:
Allows modularizing configurations into smaller `@Configuration` classes.

Real-world:

```
java
CopyEdit
@Import({RedisConfig.class, KafkaConfig.class})
```

— this keeps microservice setup clean and extendable.

Q10. Which annotation lets you define a method that returns a Spring-managed bean?

a) `@EnableBean`
b) `@Autowired`
c) `@Bean`
d) `@SpringComponent`

Explanation:
Use this inside `@Configuration` to register manually-created beans.

Real-world: Needed for configuring custom CORS filters or Kafka listeners.

Q11. How do you manually register a `DataSource` in Java config?

```
java
CopyEdit
@Bean
public DataSource dataSource() {
    ...
}
```

a) Works only with `@Component`
b) Requires `@Inject`
c) Valid inside `@Configuration`
d) Not valid in Spring Boot

Explanation:
Bean methods are valid only inside classes annotated with `@Configuration`.

Real-world: Custom DB setups — for example, Oracle DataSource or HikariCP pool — are often initialized like this for fine-tuned control.

Q12. What is the use of `@Primary`?

a) Applies only to `@Component`
b) Resolves ambiguity during autowiring
c) Replaces `@Profile`
d) Works only in XML config

Explanation:
Spring chooses the `@Primary` bean when multiple beans of the same type exist.

Real-world: Mark your default `RestTemplate` or `DataSource` as primary, but still keep others for special use cases.

Q13. Which of the following is a Spring stereotype annotation?

a) `@Role`
b) `@Service`
c) `@NamedBean`
d) `@XmlConfig`

Explanation:
`@Service`, `@Repository`, and `@Controller` are specialized stereotypes of `@Component`.

Real-world: `@Service` is ideal for business logic classes (e.g., `LoanProcessingService`).

Q14. How does Spring Boot load auto-configuration classes?

a) Using `@EnableAutoConfiguration` at runtime
b) Scans `META-INF/spring.factories` for `EnableAutoConfiguration` entries
c) Via `application.yaml`
d) With `@SpringAutoInject`

Explanation:
Spring Boot uses SPI (Service Provider Interface) to load config classes defined in the classpath.

Real-world: When you include `spring-boot-starter-data-jpa`, it auto-loads `HibernateJpaAutoConfiguration`.

Q15. What does `@ComponentScan(basePackages={...})` do?

a) Prevents scanning
b) Scans for beans in specified packages
c) Registers entities
d) Starts scheduled jobs

Explanation:
Used in JavaConfig to define where Spring looks for annotated classes.

Real-world: In large apps, restrict scanning to improve boot time and isolate unrelated modules.

Q16. What happens when using `@SpringBootApplication`?

a) You must add `@ComponentScan` separately
b) It implicitly includes `@ComponentScan`
c) It disables scanning
d) Works only with `main()` method

Explanation:
`@SpringBootApplication` = `@Configuration + @EnableAutoConfiguration + @ComponentScan`.

Real-world: Start Spring Boot microservices with just this single annotation.

Q17. What happens when a class is annotated with `@Profile("prod")`?

a) Ignores class

b) Loads all beans in that class when profile is active

c) Disables injection

d) Only works on REST controllers

Explanation:
The entire configuration class will activate conditionally.

Real-world: Profile-based configuration for caching (e.g., `@Profile("dev")` uses `ConcurrentMapCache`, while `prod` uses `RedisCache`).

Q18. Which tool validates Spring configurations during dev?

a) Spring Security

b) Spring Check

c) Spring Boot DevTools

d) Spring CLI

Explanation:
DevTools reloads apps on change and gives insights on configuration errors.

Real-world: In dev environments, it helps catch missing beans or incorrect property bindings early.

Q19. How do you avoid `NoSuchBeanDefinitionException`?

a) Avoid `@ComponentScan`

b) Use profiles sparingly

c) Use `@ConditionalOnMissingBean`

d) Never use `@Bean`

Explanation:
This tells Spring to register a bean only if no other of same type exists.

Real-world: When providing fallback beans (e.g., custom `RestTemplateBuilder`).

Q20. Which of the following is not Java-based configuration?

a) `@Configuration`

b) `@ComponentScan`

c) `beans.xml`

d) `@Bean`

Explanation:
`beans.xml` is part of XML configuration, not JavaConfig.

Real-world: JavaConfig has mostly replaced `beans.xml` in modern Spring Boot projects.

Q21. How does Spring resolve component conflicts?

a) Picks randomly
b) Uses the latest version
c) Resolves using `@Primary`, `@Qualifier`, or explicit bean name
d) Skips the bean

Explanation:
Spring lets you handle multiple candidates using qualifiers or naming.

Real-world: You often inject multiple mail providers (`SendGrid`, `SES`) and switch via qualifier.

Q22. What does `@Conditional` do?

a) Injects YAML
b) Loads beans based on conditions
c) Enables logging
d) Enables `@Controller`

Explanation:
Often used in Spring Boot auto-config to conditionally load beans.

Real-world: Use `@ConditionalOnClass(Jedis.class)` to only load Redis beans if Redis is present in the classpath.

Q23. What if a class has `@Bean` methods but no `@Configuration`?

a) Works fine
b) Spring throws an error
c) Beans won't be registered properly
d) Still picked via component scan

Explanation:
`@Configuration` is required to let Spring treat the class as a config provider.

Real-world: Omitting this leads to subtle startup bugs — beans silently don't initialize.

Q24. What helps in grouping config from YAML into structured beans?

a) `@PropertySource`
b) `@ConfigurationProperties`
c) `@Value`
d) `@PropertyGroup`

Explanation:
`@ConfigurationProperties` binds external config into typed POJOs.

Real-world: Use it for structured config like `server.port`, `database.user`, `jwt.expiry`.

Q25. How to share common YAML config across profiles?

a) Copy-paste in all profiles
b) Use `@Profile("default")`
c) Use `spring.config.import=application-common.yml`
d) Not supported

Explanation:
Spring Boot supports importing common YAML via `spring.config.import`.

Real-world: Helps manage large config across `dev`, `qa`, `prod` by keeping shared logic DRY.

Spring Boot Profiles & External Config

Focus: `YAML` vs `properties`, `@Value`, `@ConfigurationProperties`, profile activation, config override — all enhanced with explanations and real-world use cases.

Q1. Which file format allows hierarchical configuration in Spring Boot?

a) `.properties`
b) `.env`
c) `.xml`
d) `.yml`

Explanation:
YAML provides structured, tree-like config useful for nested properties.

Real-world: Ideal when grouping configs — like DB, caching, mailer setup — in a readable way:

```
yaml
CopyEdit
spring:
  datasource:
    url: jdbc:mysql://localhost/db
    username: root
```

Q2. What is the default active profile in Spring Boot?

a) `dev`
b) `default`

c) `prod`

d) none

Explanation:
If no profile is explicitly set, Spring Boot falls back to a profile named `default`.

Real-world: Great fallback strategy when deploying to environments without explicit overrides.

Q3. What is the correct way to activate a Spring profile?

a) `spring.active=dev`

b) `spring.profiles.active=dev`

c) `spring.boot.profile=dev`

d) `spring.env.dev=true`

Explanation:
`spring.profiles.active` is the standard key in `application.yml/properties`.

Real-world: Set in CI/CD pipelines or with CLI:

```
bash
CopyEdit
java -jar app.jar --spring.profiles.active=qa
```

Q4. Which config source has the highest priority?

a) `application.yml`

b) `application.properties`

c) Command-line args

d) `application-{profile}.properties`

Explanation:
Command-line properties override all others — including YAML/properties files.

Real-world: Pass temporary values during deployment, e.g. `--server.port=9090`.

Q5. What is the correct YAML syntax for nested config?

```
yaml
CopyEdit
spring:
  datasource:
    url: jdbc:mysql://localhost/db
    username: root
```

a) Valid

b) Must use colons

c) Not supported

d) Keys must be flat

Explanation:
YAML supports indentation-based nesting — unlike `.properties`.

Real-world: Improves readability when configuring Spring Security or OAuth providers.

Q6. What binds grouped YAML props into a POJO?

a) `@PropertySource`
b) `@ValueMap`
c) `@ConfigurationProperties`
d) `@InjectProps`

Explanation:
`@ConfigurationProperties(prefix="...")` maps YAML structure into a class.

Real-world: Cleanly inject multiple related props (e.g., `mail.server.host`, `port`, `auth.enabled`).

Q7. What registers a `@ConfigurationProperties` bean?

a) `@Component`
b) `@EnableConfigurationProperties`
c) `@AutoConfig`
d) `@EnableBinding`

Explanation:
This tells Spring Boot to bind properties into your POJO.

Real-world: Can be used to bind external config for SDKs, e.g., Twilio, Stripe keys.

Q8. What's the order of precedence (high to low)?

1. Command-line args
2. Env variables
3. `application.properties`
4. `@PropertySource`

Answer: 1 > 2 > 3 > 4

Explanation:
Spring Boot respects an internal `PropertySources` priority list.

Real-world: Helpful when diagnosing why a property isn't being applied.

Q9. How do you inject an environment variable?

a) `@Autowired("${VAR}")`
b) `@Value("${VAR}")`
c) `${env.VAR}`
d) `@InjectEnv`

Explanation:
`@Value` is a common way to read env variables injected at runtime.

Real-world: Pass secrets (like DB passwords) via Docker/Kubernetes environment injection.

Q10. Where do you define Actuator endpoint exposure?

a) `application-{profile}.yml`
b) `bootstrap.properties`
c) `system.properties`
d) `devtools.properties`

Explanation:
These go into the profile-specific config files for dev/ops segregation.

Real-world:

```
yaml
CopyEdit
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

Q11. What happens if both YAML and `.properties` exist?

a) `.properties` takes priority
b) YAML overrides
c) Spring crashes
d) It depends on OS

Explanation:
If both reside in the same location, `.properties` wins due to Spring's internal ordering.

Real-world: Use YAML for structure; switch to `.properties` only for local dev overrides.

Q12. Which annotation chooses beans based on profile?

a) `@Env("dev")`
b) `@Profile("dev")`

c) `@ActivateProfile`
d) `@EnvironmentMatch`

Explanation:
Spring will register beans only if the profile matches.

Real-world: Have different `DataSource` beans for `dev` (H2) and `prod` (PostgreSQL).

Q13. Which config is loaded before Spring context?

a) `bootstrap.yml`
b) `application.properties`
c) `devtools.properties`
d) `system.properties`

Explanation:
Used by Spring Cloud Config, Vault, etc. to bootstrap secrets before full context loads.

Real-world: Externalize configs stored in AWS Parameter Store or HashiCorp Vault.

Q14. Best use of `@ConfigurationProperties`?

a) In `@RestController`
b) As data entity
c) As structured config POJO
d) In `application.properties`

Explanation:
It binds config directly to a class structure, enabling validation.

Real-world:

```
java
CopyEdit
@ConfigurationProperties(prefix = "app.auth")
public class AuthConfig {
    private String clientId;
    private String secret;
}
```

Q15. Which YAML is invalid?

```
yaml
CopyEdit
my:
  key: value
    wrong: indent
```

b) Invalid due to indent

Explanation:
Indentation errors break YAML — always align nested keys.

Real-world: YAML parsing issues can crash services at startup. Use IDE validation plugins.

Q16. How to declare a default profile in Spring?

a) `@Profile("default")`
b) `@DefaultProfile`
c) `@Environment("default")`
d) No such thing

Explanation:
Explicit `@Profile("default")` marks beans for fallback use when no profile is set.

Q17. Which is optional in Spring Boot 2.2+ for `@ConfigurationProperties`?

a) `@Component`
b) `@EnableConfigurationProperties`
c) `@Bean`
d) `@Import`

Explanation:
Spring Boot 2.2+ auto-detects `@ConfigurationProperties` via classpath scanning.

Q18. How can database config be externalized?

d) Any of the above
a) `@Bean`-level hardcoding
b) `application.properties`
c) System env var

Explanation:
Spring supports all 3 via its property resolution engine.

Real-world: Combine Docker secrets with profiles and fallback in code as last resort.

Q19. How do you add default fallback in `@Value`?

```
java
CopyEdit
@Value("${app.name:DefaultApp}")
```

c) Valid fallback

Explanation:
The colon `:` acts as a default if the property isn't found.

Real-world: Avoids nulls during early dev stages or when config is incomplete.

Q20. How do you validate config values?

a) Use `@Value` with regex
b) Use `@Validated` with `@ConfigurationProperties`
c) Use YAML schema
d) Cannot validate

Explanation:
Spring can validate bound POJO properties using JSR-303 annotations.

Real-world: Enforce constraints like `@Min(1)` on thread pool sizes, `@Email` on support emails.

Q21. Correct way to inject nested YAML config?

```
yaml
CopyEdit
app:
  security:
    mode: strict

java
CopyEdit
@Value("${app.security.mode}")
```

a) Valid

Q22. Why prefer `@ConfigurationProperties` over `@Value`?

a) Better IDE support
b) Less code
c) Grouping + strong typing
d) Works in Boot 3.0 only

Explanation:
`@ConfigurationProperties` enables structured config + bean validation.

Q23. How do you manage secrets across environments?

a) Use `application-dev.yml`
b) Use Spring Cloud Vault/SecretsManager

c) Encrypt inline
d) Store in app.properties

Explanation:
Externalized secret managers + `bootstrap.yml` allow secure loading of credentials.

Q24. What if both CLI and env var provide same key?

b) CLI wins

Explanation:
Command-line args override everything in Spring's hierarchy.

Real-world: Run container with:

```
bash
CopyEdit
java -jar app.jar --db.url=...
```

Q25. Best practice for validating external config?

b) `@Validated + @ConfigurationProperties` + YAML

Explanation:
You ensure safety and documentation via typed, validated POJOs.

Real-world: Catch startup errors like missing `jwt.expiry` or invalid `threadPoolSize`.

Spring MVC Web Layer

1. What is the role of `DispatcherServlet` in Spring MVC?

A. It renders views
B. It handles dependency injection
C. It acts as the front controller
D. It maps URLs to servlets

Answer: C
Explanation: `DispatcherServlet` is the front controller in Spring MVC responsible for routing incoming HTTP requests to appropriate handler methods.
Insight: It enables a centralized request processing pipeline, essential for request filtering, security, and view rendering.

---

2. Which annotation is used to define a Spring MVC controller class?

A. `@Service`
B. `@Component`
C. `@Controller`
D. `@RestService`

Answer: C
Explanation: `@Controller` marks a class as a web controller that handles HTTP requests.
Insight: In a typical layered architecture, `@Controller` classes form the entry point for user interactions.

---

3. How is `@RestController` different from `@Controller`?

A. It injects repositories
B. It returns only views
C. It combines `@Controller` and `@ResponseBody`
D. It uses XML views

Answer: C
Explanation: `@RestController` implies `@Controller + @ResponseBody`, so all methods return JSON/XML instead of views.
Insight: `@RestController` is ideal for REST APIs where HTML rendering is not required.

---

4. Which annotation binds a path variable from the URI to a method parameter?

A. `@RequestParam`
B. `@PathVariable`
C. `@RequestBody`
D. `@Header`

Answer: B
Explanation: `@PathVariable` maps URI path segments to method parameters.
Insight: Useful for RESTful patterns like `/customers/{id}`.

---

5. Which file typically configures the `DispatcherServlet` in XML-based Spring MVC?

A. `web.xml`
B. `application.properties`
C. `dispatcher-servlet.xml`
D. `pom.xml`

Answer: A
Explanation: `web.xml` defines servlet mappings, including the `DispatcherServlet`.
Insight: Though annotations replaced XML configs in modern Spring Boot, legacy apps still use this.

---

6. What happens when `DispatcherServlet` receives a request?

A. It renders the JSP directly
B. It executes filters
C. It delegates to a handler via handler mappings
D. It connects to a database

Answer: C
Explanation: `DispatcherServlet` relies on `HandlerMapping` to route to appropriate `@Controller` methods.
Insight: This allows pluggable URL-routing strategies (e.g., `@RequestMapping`, `HandlerInterceptor`).

---

7. What annotation do you use to bind query parameters to method arguments?

A. `@PathParam`
B. `@RequestParam`
C. `@QueryParam`
D. `@Param`

Answer: B
Explanation: `@RequestParam` extracts parameters from the query string.
Insight: Works well with forms and GET requests like `/search?name=foo`.

---

8. What is the default view resolver in Spring MVC?

A. `ThymeleafViewResolver`
B. `FreeMarkerViewResolver`
C. `InternalResourceViewResolver`
D. `JSPViewResolver`

Answer: C
Explanation: `InternalResourceViewResolver` is the default and resolves views to JSP pages.
Insight: You can configure the prefix/suffix like `/WEB-INF/views/` + `.jsp`.

---

9. In Spring Boot, how are controllers auto-detected?

A. Via `component-scan` XML
B. Explicit registration
C. Package scanning with `@SpringBootApplication`
D. Through servlet annotations

Answer: C
Explanation: `@SpringBootApplication` includes component scanning.
Insight: Keep controllers under the root package for auto-detection.

---

10. Which annotation is used to map HTTP methods like GET/POST?

A. `@Route`
B. `@Method`
C. `@RequestMapping`
D. `@HttpMapping`

Answer: C
Explanation: `@RequestMapping` with `method = RequestMethod.GET` maps specific HTTP verbs.
Insight: Spring 4+ introduced `@GetMapping`, `@PostMapping`, etc., for clarity.

---

11. How can you return JSON from a controller method?

A. Use `@View`
B. Return `String`
C. Annotate with `@ResponseBody`
D. Use `ModelAndView`

Answer: C
Explanation: `@ResponseBody` writes the return value to the response body.
Insight: Use this for REST APIs and AJAX endpoints.

---

12. What object does Spring use to store model attributes for a view?

A. `HashMap`
B. `Model`
C. `Request`
D. `Session`

Answer: B
Explanation: `Model` holds attributes to be rendered in the view.
Insight: Often used in Thymeleaf or JSP to populate forms.

---

13. Which annotation is used to perform input validation on a model object?

A. `@Assert`
B. `@Validated`
C. `@Valid`
D. `@Check`

Answer: C
Explanation: `@Valid` triggers JSR-303 bean validation annotations like `@NotNull`.
Insight: Combine with `BindingResult` for fine-grained error handling.

---

14. What happens if validation fails with `@Valid`?

A. HTTP 500 is returned
B. The method is skipped
C. An exception is thrown unless `BindingResult` is used
D. The app exits

Answer: C
Explanation: Without `BindingResult`, Spring throws `MethodArgumentNotValidException`.
Insight: Useful for gracefully handling user input issues.

---

15. What does `ModelAndView` object represent?

A. JSON view only
B. Session metadata
C. A model with the name of a view
D. A controller

Answer: C
Explanation: `ModelAndView` encapsulates both the view name and model attributes.
Insight: Still used in legacy apps, but `Model` + `String` return is more common in modern Spring.

---

16. What is the role of `HandlerInterceptor` in Spring MVC?

A. Handles controller input
B. Applies pre/post-processing logic around handler methods
C. Validates database input
D. Maps URL patterns

Answer: B
Explanation: Interceptors allow custom logic before/after a controller is invoked.
Insight: Common for logging, authentication, and metrics.

---

17. How are validation messages customized in Spring MVC?

A. Hardcoded in Java
B. Declared in `messages.properties`
C. Via system logs
D. In JSP comments

Answer: B
Explanation: Spring uses message sources to resolve validation messages via keys.
Insight: Supports internationalization and dynamic user feedback.

---

18. Which class is at the heart of request handling in Spring MVC?

A. `WebApplicationContext`
B. `DispatcherServlet`
C. `ServletRequest`
D. `ApplicationContext`

Answer: B
Explanation: All web requests go through `DispatcherServlet`.
Insight: You can register interceptors and filters to hook into its lifecycle.

---

19. What is returned by default when a controller method returns a `String`?

A. JSON
B. View name
C. Raw HTML
D. HTTP status

Answer: B
Explanation: Spring interprets `String` return as a view name (e.g., `home.jsp`).
Insight: Return `@ResponseBody` to bypass view resolution.

---

20. Which configuration is required for validation to work?

A. `@EnableValidation`
B. `spring.validation.enabled=true`
C. `@Valid` + bean validation provider (Hibernate Validator)
D. None, it works out of the box

Answer: C
Explanation: Spring integrates with JSR-303 via Hibernate Validator.
Insight: Spring Boot auto-configures this, but you can also define your own validator.

---

21. Which type of controller supports asynchronous request processing?

A. `@AsyncController`
B. `Callable<T>` return type
C. `@Scheduled`
D. `@SyncController`

Answer: B
Explanation: Returning `Callable<T>` enables async request handling in Spring MVC.
Insight: Useful when calling external APIs or non-blocking I/O.

---

22. How do you handle form submission errors in Spring MVC?

A. Use try-catch
B. Use `BindingResult`
C. Return null
D. Use `HttpSession`

Answer: B
Explanation: `BindingResult` captures validation errors when `@Valid` fails.
Insight: You can show friendly messages and re-render the form.

---

23. Which configuration enables custom error pages in Spring MVC?

A. `application.yaml` only
B. `@ErrorHandler`
C. `SimpleMappingExceptionResolver`
D. `HandlerInterceptor`

Answer: C
Explanation: `SimpleMappingExceptionResolver` maps exceptions to specific view names.
Insight: Deprecated in Spring Boot in favor of `ErrorController`.

---

24. When would you use `@ModelAttribute`?

A. To fetch cookies
B. To map request headers
C. To bind a method param or expose data to model
D. To log request

Answer: C
Explanation: `@ModelAttribute` binds form fields or adds attributes to the model.
Insight: Often used in pre-populating form options like dropdowns.

---

25. What does the `ViewResolver` do in Spring MVC?

A. Authenticates users
B. Maps controllers
C. Resolves logical view names to actual templates
D. Compiles Java code

Answer: C

Explanation: ViewResolvers map logical view names to real templates like JSPs or Thymeleaf files.

Insight: You can chain multiple resolvers or define custom logic (e.g., JSON fallback).

25 MCQs — REST API with Spring Boot

1. What does `@RestController` do in Spring Boot?

A. Registers a view
B. Returns HTML templates
C. Returns objects as JSON or XML
D. Handles database transactions

Answer: C

Explanation: `@RestController` is a combination of `@Controller` and `@ResponseBody`, sending JSON/XML responses directly.

Insight: Ideal for microservices and modern frontend-backend architectures.

---

2. What is the default return format for `@RestController`?

A. XML
B. HTML
C. JSON
D. CSV

Answer: C

Explanation: Spring Boot auto-configures Jackson for JSON serialization.

Insight: To switch to XML, you need to include a JAXB or Jackson XML dependency.

---

3. What is the purpose of DTOs in REST APIs?

A. Represent database entities
B. Carry data between processes
C. Manage security
D. Log audit events

Answer: B

Explanation: DTO (Data Transfer Object) is used to structure API input/output data separately from entity models.

Insight: Prevents exposing sensitive fields and reduces tight coupling.

---

4. What annotation is used to map HTTP GET requests?

A. `@GetRequest`
B. `@GetMapping`
C. `@RequestGet`
D. `@HttpGet`

Answer: B
Explanation: `@GetMapping` is a composed annotation for `@RequestMapping(method=GET)`.
Insight: Simplifies REST endpoint declaration.

---

5. What is the role of `@ExceptionHandler` in Spring?

A. Retry failed requests
B. Map exceptions to HTTP responses
C. Encrypt request data
D. Validate incoming DTOs

Answer: B
Explanation: Used within controllers to catch and handle exceptions at a method level.
Insight: Combine with `@ControllerAdvice` for global handling.

---

6. What HTTP status code typically represents a successful `POST` that creates a resource?

A. 200 OK
B. 204 No Content
C. 201 Created
D. 202 Accepted

Answer: C
Explanation: 201 indicates resource creation. Optionally, include a `Location` header.
Insight: REST clients rely on status codes for flow control and retries.

---

7. What does HATEOAS stand for?

A. Hypermedia as the Engine of Application State
B. HTML and Tags Engine for API Syntax
C. Host-API Translation Extension Over Application Services
D. Hibernate Authorization Template for Enterprise API Solutions

Answer: A
Explanation: A REST principle where responses contain links to related resources.
Insight: Improves discoverability of REST APIs.

---

8. How is HATEOAS implemented in Spring Boot?

A. With Jackson annotations
B. Using `spring-boot-starter-hateoas` and `EntityModel`
C. Through Thymeleaf
D. By enabling JSON view

Answer: B
Explanation: Spring HATEOAS uses `EntityModel<T>`, `CollectionModel<T>`, and `Link`.
Insight: Helps clients understand actions they can take on a resource.

---

9. Which exception is thrown when a resource is not found?

A. `IOException`
B. `NullPointerException`
C. `ResourceNotFoundException`
D. `NoSuchElementException`

Answer: C *(custom)*
Explanation: Best practice is to define your own `ResourceNotFoundException`.
Insight: Map this to 404 with `@ExceptionHandler`.

---

10. Which annotation marks a method to handle DELETE requests?

A. `@RequestMapping(method = DELETE)`
B. `@Delete`
C. `@DeleteMapping`
D. `@HttpDelete`

Answer: C
Explanation: `@DeleteMapping` is shorthand for mapping DELETE HTTP method.
Insight: RESTful services should support DELETE for resource cleanup.

---

11. How do you represent a list of entities in Spring HATEOAS?

A. `ListViewModel<T>`
B. `CollectionModel<T>`
C. `LinkedList<T>`
D. `HateoasList<T>`

Answer: B
Explanation: `CollectionModel<T>` includes both the list and links for HATEOAS.
Insight: Can add pagination, self-links, and navigation actions.

---

12. What is `@ControllerAdvice` used for?

A. Advise beans
B. Apply advice to service layer
C. Define global exception handlers
D. Act as DAO

Answer: C
Explanation: `@ControllerAdvice` enables centralized exception handling for REST controllers.
Insight: Keeps your controller logic clean.

---

13. What content type does a REST API usually consume and produce?

A. `text/html`
B. `application/json`
C. `application/xml`
D. `multipart/form-data`

Answer: B
Explanation: JSON is the standard content type for REST APIs.
Insight: Use `@RequestMapping(produces = "application/json")` for explicit control.

---

14. How do you map a request body to a DTO object?

A. `@QueryParam`
B. `@RequestParam`
C. `@RequestBody`
D. `@PathVariable`

Answer: C
Explanation: `@RequestBody` binds request JSON to a Java object.
Insight: Ensure DTO fields have proper getters/setters for deserialization.

---

15. Which annotation handles validation on a DTO in REST API?

A. `@Validated`
B. `@Valid`
C. `@Check`
D. `@ModelAttribute`

Answer: B
Explanation: `@Valid` triggers bean validation on input objects.
Insight: Combine with `BindingResult` or `MethodArgumentNotValidException`.

---

16. What annotation defines HTTP response status for a custom exception?

A. `@ResponseStatus`
B. `@HttpStatus`
C. `@ExceptionCode`
D. `@ErrorCode`

Answer: A
Explanation: `@ResponseStatus(HttpStatus.NOT_FOUND)` can be set directly on an exception class.
Insight: Simplifies mapping exceptions to status codes.

---

17. What's the correct status code for a successful DELETE operation?

A. 200 OK
B. 201 Created
C. 204 No Content
D. 404 Not Found

Answer: C
Explanation: 204 indicates success with no response body.
Insight: Makes DELETE operations cleaner with minimal response.

---

18. Which library does Spring Boot use to serialize Java objects to JSON?

A. Gson
B. Jackson
C. Fastjson
D. Moshi

Answer: B
Explanation: Jackson is auto-configured in Spring Boot for JSON serialization.
Insight: You can customize it via `ObjectMapper` bean.

---

19. What is `ResponseEntity` used for?

A. Logging responses
B. Returning HTML
C. Representing full HTTP response with status, headers, and body
D. Injecting services

Answer: C
Explanation: `ResponseEntity` gives complete control over HTTP response.
Insight: Useful for conditional status codes and headers.

---

20. What is the correct status code when validation fails?

A. 403 Forbidden
B. 400 Bad Request
C. 409 Conflict
D. 422 Unprocessable Entity

Answer: B
Explanation: 400 indicates a bad request, typically due to input validation failure.
Insight: Spring returns this when `@Valid` fails and isn't caught.

---

21. How to return a location header for a newly created resource?

A. Return String
B. Use `ResponseEntity.created(URI)`
C. Set `@HeaderParam`
D. Use `@ResponseStatus`

Answer: B
Explanation: `ResponseEntity.created()` sets status 201 and Location header.
Insight: Helps clients follow RESTful hypermedia principles.

---

22. Why should DTOs be used instead of entities in API responses?

A. Entities are faster
B. DTOs are encrypted
C. DTOs offer better abstraction and avoid lazy loading issues
D. DTOs require less memory

Answer: C
Explanation: DTOs prevent JPA-specific problems and expose only required data.
Insight: Keeps API contracts stable despite internal changes.

---

23. How do you document REST APIs in Spring Boot?

A. `README.txt`
B. Swagger/OpenAPI (`springdoc-openapi-ui`)
C. JavaDocs
D. Excel Sheets

Answer: B
Explanation: Swagger auto-generates interactive API documentation.
Insight: Great for internal teams and external API consumers.

---

24. Which header should you include when sending JSON to an API?

A. `Accept: application/xml`
B. `Content-Type: application/json`
C. `Authorization: basic`
D. `Content-Disposition: inline`

Answer: B
Explanation: Informs the server of the payload's format.
Insight: Essential in API clients and Postman tests.

---

25. What does `@CrossOrigin` annotation handle in Spring Boot?

A. Prevents CSRF
B. Enables CORS for external domains
C. Enables OAuth2 login
D. Validates JWT

Answer: B
Explanation: `@CrossOrigin` allows cross-origin requests to a controller or method.
Insight: Useful for frontend apps calling backend APIs on different ports/domains.

25 MCQs — Spring Data JPA: DB Access

1. What does the `@Repository` annotation indicate in Spring?

A. A class handles REST endpoints
B. A bean is a controller
C. A class is a data access component
D. It performs security checks

Answer: C
Explanation: `@Repository` marks a bean for data access; it also translates database exceptions to Spring's `DataAccessException`.
Insight: Enables declarative exception handling and component scanning.

---

2. What is the root interface for all Spring Data JPA repositories?

A. `JpaDao`
B. `JpaRepository`
C. `CrudRepository`
D. `Repository`

Answer: D
Explanation: `Repository` is the base interface; others like `CrudRepository` and

`JpaRepository` extend it.

Insight: Extending `JpaRepository` gives access to rich JPA features.

---

3. Which interface provides pagination and sorting support?

A. `CrudRepository`
B. `JpaPagingRepository`
C. `PagingAndSortingRepository`
D. `JpaDao`

Answer: C
Explanation: `PagingAndSortingRepository` adds `findAll(Pageable)` and `findAll(Sort)`.
Insight: Use it when you want pagination without full `JpaRepository` features.

---

4. What does JPQL stand for?

A. Java Persistence Query Language
B. Java Page Query Language
C. JPA Processed Query Lookup
D. JPA Projection Query Language

Answer: A
Explanation: JPQL is an object-oriented query language for JPA entities.
Insight: Uses entity names and fields, not table names or columns.

---

5. What does the `@Query` annotation do?

A. Executes stored procedures
B. Maps database views
C. Allows writing custom JPQL or native SQL
D. Enables batch updates

Answer: C
Explanation: `@Query` defines JPQL/native SQL queries directly on methods.
Insight: Ideal for complex or customized queries beyond method naming conventions.

---

6. Which keyword is used in JPQL to select entities?

A. `TABLE`
B. `DOCUMENT`
C. `SELECT`
D. `FETCH`

Answer: C
Explanation: `SELECT` is used in JPQL to retrieve entity objects.
Insight: You can select partial objects via constructor expressions too.

---

7. What's the purpose of `@Param` in a `@Query` method?

A. Define pagination
B. Inject environment variables
C. Bind method parameters to query parameters
D. Apply validation

Answer: C
Explanation: `@Param("name")` maps method parameters to named query parameters like `:name`.
Insight: Improves readability and flexibility of dynamic queries.

---

8. What is a projection in Spring Data JPA?

A. Cached query result
B. A stored function
C. A way to map part of an entity to a DTO
D. A view on a database

Answer: C
Explanation: Projections allow fetching only required fields via interfaces or classes.
Insight: Reduces overhead and avoids loading unnecessary data.

---

9. How do you create an interface-based projection?

A. Using `@RepositoryRestResource`
B. Define a Java interface with getters matching entity fields
C. Using `@EntityGraph`
D. Registering in `application.properties`

Answer: B
Explanation: Spring dynamically maps selected columns into projection interfaces.
Insight: Can be used with nested projections too.

---

10. What does `Pageable` represent?

A. A request parameter binder
B. An SQL executor
C. Pagination and sorting information
D. A transaction manager

Answer: C
Explanation: `Pageable` is passed to repository methods to control page size and order.
Insight: Enables offset-limit-style pagination in a type-safe way.

---

11. How do you get total elements in a paginated query result?

A. Use `findAll()`
B. Count manually
C. Use `Page.getTotalElements()`
D. Use `Query.getSize()`

Answer: C
Explanation: `Page<T>` provides total pages, elements, and current slice.
Insight: Helps build UI pagination components dynamically.

---

12. What happens if a method name in a repository doesn't follow Spring naming conventions?

A. Throws an exception
B. Auto-generates SQL
C. You must annotate with `@Query`
D. Ignores the method

Answer: C
Explanation: If the method name is non-standard, you need `@Query` to define the query.
Insight: Naming conventions eliminate boilerplate in most use cases.

---

13. What does `findByNameAndAge` do?

A. Finds all names
B. Joins name and age columns
C. Fetches entities matching both `name` and `age`
D. Adds a LIKE filter

Answer: C
Explanation: Spring Data parses method names into conditions.
Insight: Supports chaining with `And`, `Or`, `Between`, `In`, etc.

---

14. Which keyword is used in JPQL to perform joins?

A. `UNION`
B. `MERGE`
C. `JOIN`
D. `LINK`

Answer: C
Explanation: JPQL supports `JOIN`, `LEFT JOIN`, `FETCH JOIN` for entity relationships.
Insight: `FETCH JOIN` is used to solve lazy loading issues.

---

15. What does `flush()` do in JPA?

A. Commits the transaction
B. Clears the cache
C. Synchronizes persistence context with database
D. Deletes all rows

Answer: C
Explanation: Forces pending entity changes to the DB, within the current transaction.
Insight: Use carefully in batch/bulk operations.

---

16. Which repository interface includes `saveAll()`?

A. `JpaRepository`
B. `PagingAndSortingRepository`
C. `CrudRepository`
D. All of the above

Answer: D
Explanation: `saveAll()` is available in all main Spring Data interfaces.
Insight: Preferred for inserting/updating collections in batch.

---

17. Which method returns an entity wrapped in `Optional<T>`?

A. `findOne()`
B. `getById()`
C. `findById()`
D. `searchOne()`

Answer: C
Explanation: `findById(id)` returns `Optional<T>` to handle nulls safely.
Insight: Promotes null-safety in Java 8+ codebases.

---

18. How do you ensure only a subset of fields are returned in a query?

A. Use DTOs or Projections
B. Define Entity Graph
C. Annotate with `@FetchSubset`
D. Use `LazyLoad`

Answer: A
Explanation: DTO projections fetch selected fields, improving performance.
Insight: Common in large tables or sensitive data handling.

---

19. What does the `count()` method do in Spring Data?

A. Counts requests per second
B. Returns the number of records in a table
C. Monitors batch operations
D. Tracks insert failures

Answer: B
Explanation: `count()` returns total number of entities managed by the repository.
Insight: Used in reporting and pagination metadata.

---

20. Which annotation lets you write native SQL queries?

A. `@SQL`
B. `@NativeQuery`
C. `@Query(nativeQuery = true)`
D. `@RawSQL`

Answer: C
Explanation: Add `nativeQuery = true` to `@Query` to write raw SQL.
Insight: Useful for database-specific optimizations.

---

21. How do you fetch paged results sorted by a field?

A. Use `Pageable` with `Sort.by("field")`
B. Append `ORDER BY` in SQL
C. Use `@OrderBy` annotation
D. Sort manually after query

Answer: A
Explanation: Construct `PageRequest.of(page, size, Sort.by(...))`.
Insight: You can sort by multiple fields or directions.

---

22. Which method helps avoid `LazyInitializationException`?

A. Use `@Lazy(false)`
B. Fetch eagerly
C. Use `@Transactional`
D. All of the above

Answer: D
Explanation: Lazy loading needs active persistence context or eager fetching.
Insight: Carefully balance performance vs. data availability.

---

23. What does `deleteById(id)` do if the entity is not found?

A. Throws an exception
B. Returns false
C. Logs a warning
D. Does nothing

Answer: A
Explanation: It throws `EmptyResultDataAccessException`.
Insight: Always check existence before delete in sensitive apps.

---

24. Which method is best for updating a specific field in bulk?

A. `save()`
B. `merge()`
C. Custom `@Modifying @Query`
D. `updateByField()`

Answer: C
Explanation: Use `@Modifying` with custom JPQL/SQL update query.
Insight: Required for performance-sensitive updates (e.g., flags, status).

---

25. Which Spring property enables SQL logging?

A. `spring.jpa.debug=true`
B. `logging.sql=true`
C. `spring.jpa.show-sql=true`
D. `hibernate.sql.log=true`

Answer: C
Explanation: Logs SQL to console when set to `true`.
Insight: Combine with `hibernate.format_sql=true` for readability.

25 MCQs — Transaction Management: Data Integrity

1. What does the `@Transactional` annotation do?

A. Encrypts data
B. Marks methods for dependency injection
C. Wraps method execution in a transaction
D. Makes a method asynchronous

Answer: C

Explanation: `@Transactional` defines transactional boundaries so all DB operations within either succeed or fail together.

Insight: Used in service or repository layers to maintain consistency.

---

2. What is the default propagation type for `@Transactional`?

A. `NEVER`
B. `REQUIRES_NEW`
C. `MANDATORY`
D. `REQUIRED`

Answer: D

Explanation: `REQUIRED` means the method will join an existing transaction or create a new one if none exists.

Insight: Good default that avoids nested transaction complexity.

---

3. What does `Propagation.REQUIRES_NEW` mean?

A. Uses parent transaction
B. Creates a new transaction, suspending existing one
C. Ignores transactions
D. Shares the existing Hibernate session

Answer: B

Explanation: Forces a new, independent transaction.

Insight: Useful when inner logic must persist regardless of outer failure (e.g., audit logs).

---

4. What happens if a `@Transactional` method throws a checked exception by default?

A. Transaction is always rolled back
B. Transaction is committed
C. It depends on rollback rules
D. Spring rethrows the exception

Answer: B

Explanation: By default, Spring only rolls back on unchecked (`RuntimeException`) or `Error`.

Insight: You must explicitly configure rollback for checked exceptions using `rollbackFor`.

---

5. How can you force rollback for a checked exception?

A. `@RollbackForChecked`
B. Catch and throw a runtime exception

C. `@Transactional(rollbackFor = CustomException.class)`
D. Use `@ExceptionHandler`

Answer: C
Explanation: `rollbackFor` attribute defines exceptions to trigger rollback.
Insight: Avoids accidental commits when business logic fails silently.

---

6. Which layer typically contains `@Transactional` annotations in a well-layered app?

A. Controller
B. Entity
C. Service
D. UI

Answer: C
Explanation: Service layer handles orchestration and business rules, ideal for transaction boundaries.
Insight: Keeps controller lightweight and testable.

---

7. What is `Propagation.SUPPORTS`?

A. Always creates a transaction
B. Fails if no transaction exists
C. Executes in transaction if one exists
D. Starts a read-only transaction

Answer: C
Explanation: SUPPORTS runs in a transaction if available but doesn't create one.
Insight: Suitable for read-only or optional transaction scenarios.

---

8. What is `Propagation.NEVER` used for?

A. Ensures a method always has a transaction
B. Prevents any transaction from existing
C. Suspends existing transactions
D. Rolls back on exception

Answer: B
Explanation: `NEVER` throws an exception if an active transaction exists.
Insight: Rare but useful for performance-critical non-transactional code.

---

9. What's the behavior of `Propagation.MANDATORY`?

A. Creates a new transaction
B. Always commits after method

C. Requires an existing transaction
D. Skips method if no transaction

Answer: C
Explanation: Fails with `IllegalTransactionStateException` if no transaction exists.
Insight: Good for enforcing calling context constraints.

---

10. How does Spring define a transaction rollback programmatically?

A. `@Rollback`
B. `@ErrorHandling`
C. Throwing an exception within a `@Transactional` method
D. Catching and logging the error

Answer: C
Explanation: A thrown runtime exception marks the transaction for rollback.
Insight: Always ensure exceptions are not swallowed if rollback is expected.

---

11. What is the effect of `@Transactional(readOnly = true)`?

A. Rolls back automatically
B. Prevents updates and inserts
C. Optimizes performance for reads
D. Disables connection pooling

Answer: C
Explanation: Hints underlying frameworks (like Hibernate) to skip dirty checks and unnecessary writes.
Insight: Use it in data retrieval services for performance tuning.

---

12. Can a private method be transactional in Spring?

A. Yes
B. No
C. Only if annotated twice
D. Only with AOP proxies enabled

Answer: B
Explanation: Spring proxies only public/protected methods unless using AspectJ weaving.
Insight: Annotating private methods has no effect — structure your logic accordingly.

---

13. What is the role of `TransactionManager` in Spring?

A. Builds SQL queries
B. Manages JDBC connections

C. Coordinates transaction boundaries
D. Parses JPQL

Answer: C
Explanation: Spring delegates transaction operations to a platform-specific `TransactionManager`.
Insight: Key abstraction to support JDBC, JPA, JMS, etc.

---

14. How do you define a rollback for multiple exception types?

A. `@Transactional(rollbackFor = {Exception1.class, Exception2.class})`
B. Use `@RollbackMulti`
C. Throw `Throwable`
D. Spring does it automatically

Answer: A
Explanation: Accepts a class array in `rollbackFor`.
Insight: Enables fine-grained control over transaction rollback logic.

---

15. Which transaction propagation type is ideal for background job logging that should persist even on failure?

A. `REQUIRED`
B. `REQUIRES_NEW`
C. `MANDATORY`
D. `SUPPORTS`

Answer: B
Explanation: Isolates logging or audit operations from main transaction failures.
Insight: Helps preserve debug/audit trails.

---

16. What is `TransactionTemplate` in Spring?

A. A DTO for rollback rules
B. A static context manager
C. A programmatic transaction management API
D. A JPA mapping file

Answer: C
Explanation: `TransactionTemplate` allows executing code within a transaction block programmatically.
Insight: Used where annotations are not viable (e.g., legacy systems, dynamic logic).

---

17. Which exception type triggers rollback by default?

A. `IOException`
B. `CheckedException`
C. `RuntimeException`
D. `ValidationException`

> **Answer**: C
> **Explanation**: Only unchecked exceptions cause rollback unless configured otherwise.
> **Insight**: Avoid catching `RuntimeException` unless you rethrow it.

---

18. What happens when `@Transactional` is applied to a method that internally calls another transactional method in the same class?

A. Both run in same transaction
B. Transaction is nested
C. Transaction is applied twice
D. Inner method's `@Transactional` is ignored

> **Answer**: D
> **Explanation**: Self-invocation bypasses proxy; the annotation is not intercepted.
> **Insight**: Extract inner method to a separate bean for isolation.

---

19. Can `@Transactional` be used on an interface?

A. Yes, always
B. No
C. Only with JDK dynamic proxies
D. Yes, but only Spring ignores it

> **Answer**: C
> **Explanation**: JDK proxies support interface-level annotations; CGLIB works on classes.
> **Insight**: Best to annotate implementation class for clarity.

---

20. Which attribute of `@Transactional` defines rollback exception classes?

A. `rollbackOnly`
B. `rollbackFor`
C. `onError`
D. `catchType`

> **Answer**: B
> **Explanation**: Specifies the exception types that trigger rollback.
> **Insight**: Combine with `noRollbackFor` for complete control.

---

21. How does Spring manage transaction demarcation internally?

A. Intercepts DB calls
B. Uses thread-local transaction context
C. Wraps methods with Java's `TransactionManager`
D. Declares static methods

Answer: B
Explanation: Transaction context is held per-thread using ThreadLocal.
Insight: Sharing threads across pools can break transaction integrity.

---

22. What happens if `@Transactional` is placed on both class and method?

A. Class-level always overrides
B. Method-level overrides class-level
C. Spring throws an exception
D. Both are ignored

Answer: B
Explanation: Method-level settings take precedence.
Insight: Allows general policy at class-level and overrides per method.

---

23. Which property disables Spring transaction management entirely?

A. `spring.tx.enabled=false`
B. Remove `@EnableTransactionManagement`
C. `spring.jpa.auto-commit=false`
D. `transaction.auto=false`

Answer: B
Explanation: Without `@EnableTransactionManagement`, Spring doesn't activate proxy-based transactions.
Insight: Default in Spring Boot is auto-enabled via auto-configuration.

---

24. What is the purpose of `@EnableTransactionManagement`?

A. Enables JDBC transactions
B. Turns on proxy-based AOP for `@Transactional`
C. Starts H2 database
D. Logs SQL statements

Answer: B
Explanation: Activates transaction handling using Spring AOP.
Insight: Required in non-Boot or manual configurations.

---

25. What happens if a transaction is committed but an exception occurs after commit?

A. Transaction is rolled back
B. Exception is suppressed
C. Data remains committed
D. Spring reopens the transaction

Answer: C
Explanation: Once committed, changes are permanent — post-commit exceptions don't affect DB state.
Insight: Handle post-commit errors gracefully to avoid inconsistent user states.

25 MCQs — Spring Security Basics: Protection Layer

1. What is the role of `AuthenticationManager` in Spring Security?

A. Encrypts credentials
B. Validates and authenticates user credentials
C. Manages sessions
D. Stores user profiles

Answer: B
Explanation: `AuthenticationManager` handles authentication by validating `Authentication` objects.
Insight: Acts as a gatekeeper to all secured resources.

---

2. What is the default authentication method in Spring Security?

A. JWT-based auth
B. Basic authentication
C. Form-based login
D. OAuth2

Answer: C
Explanation: By default, Spring Security sets up form-based login with a generated login page.
Insight: You can override this via `SecurityFilterChain`.

---

3. Which filter is responsible for handling username and password login?

A. `BasicAuthenticationFilter`
B. `JwtAuthenticationFilter`
C. `UsernamePasswordAuthenticationFilter`
D. `FilterSecurityInterceptor`

Answer: C
Explanation: It processes form login requests and delegates to `AuthenticationManager`.
Insight: Custom authentication flows often extend this filter.

---

4. What is the primary purpose of Spring Security filters?

A. Format data
B. Route API calls
C. Enforce security checks in request lifecycle
D. Log audit events

Answer: C
Explanation: Filters intercept requests and apply authentication and authorization logic.
Insight: Pluggable filters enable layered security like JWT, CSRF, and OAuth.

---

5. What does `HttpSecurity.csrf().disable()` do?

A. Enables encryption
B. Blocks all requests
C. Disables CSRF token validation
D. Bypasses login

Answer: C
Explanation: Disables CSRF protection, often needed in stateless APIs.
Insight: Safe to disable only when using tokens or same-origin API clients.

---

6. What is the role of `SecurityContextHolder`?

A. Stores database sessions
B. Manages application logs
C. Holds current user's `Authentication` info
D. Encrypts passwords

Answer: C
Explanation: Thread-local storage of current user context.
Insight: Useful in business logic to access user identity outside controllers.

---

7. What is returned by `AuthenticationManager.authenticate()` on success?

A. A new session
B. Null
C. A fully authenticated `Authentication` object
D. Username string

Answer: C
Explanation: Includes user details, granted authorities, and status.
Insight: Required to set into `SecurityContext` post-authentication.

---

8. Which class provides user credentials and authorities?

A. `AuthenticationManager`
B. `SecurityFilterChain`
C. `UserDetails`
D. `HttpSession`

> **Answer**: C
> **Explanation**: `UserDetails` defines username, password, and granted roles.
> **Insight**: Custom user services implement `UserDetailsService`.

---

9. Which interface is used to load users from a custom store?

A. `UserInfoService`
B. `UserLookup`
C. `UserDetailsService`
D. `UserSecurityProvider`

> **Answer**: C
> **Explanation**: Loads `UserDetails` from DB, LDAP, or external API.
> **Insight**: Customize it to plug in your domain model and data store.

---

10. What does `hasRole('ADMIN')` mean in an access rule?

A. User must have `ROLE_ADMIN` authority
B. Matches exact string "ADMIN"
C. Maps to a URL pattern
D. Always returns true for any role

> **Answer**: A
> **Explanation**: Prefix `ROLE_` is automatically added unless overridden.
> **Insight**: Keep consistent naming like `ROLE_USER`, `ROLE_MANAGER`.

---

11. What is the difference between `hasAuthority()` and `hasRole()`?

A. Same function
B. `hasRole()` auto-prefixes with `ROLE_`, `hasAuthority()` doesn't
C. One is for OAuth2
D. Only `hasRole()` supports JWT

> **Answer**: B
> **Explanation**: `hasAuthority("ROLE_ADMIN")` equals `hasRole("ADMIN")`.
> **Insight**: For fine-grained permissions, prefer `hasAuthority()`.

---

12. How are JWT tokens typically passed in HTTP requests?

A. Query parameters
B. Form fields
C. `Authorization` header with `Bearer` prefix
D. In cookies only

Answer: C
Explanation: Header `Authorization: Bearer <token>` is standard.
Insight: Stateless and scalable for APIs and mobile clients.

---

13. Which annotation restricts access to a method based on role?

A. `@SecureMethod`
B. `@RolesAllowed`
C. `@Secured`
D. `@PreAuthorize`

Answer: D
Explanation: `@PreAuthorize("hasRole('ADMIN')")` checks before method execution.
Insight: Works well with service-layer authorization.

---

14. What is the correct order of filters in Spring Security (simplified)?

A. Authentication → ExceptionTranslation → Authorization
B. Authorization → Authentication
C. FilterSecurity → CSRF → Exception
D. CSRF → Authentication

Answer: A
Explanation: Authentication first, then exceptions, then access control.
Insight: You can customize the chain using `SecurityFilterChain`.

---

15. How does Spring Security handle password encoding?

A. Stores plain text
B. Uses `@SecurePassword`
C. Uses `PasswordEncoder` interface
D. Encrypts in controller

Answer: C
Explanation: Implementations like `BCryptPasswordEncoder` ensure secure password hashing.
Insight: Always hash passwords before storing or comparing.

---

16. What is the purpose of `SecurityFilterChain` bean?

A. Filter logs
B. Define ordered security filter logic
C. Encrypt responses
D. Configure endpoints

Answer: B
Explanation: Core component for programmatically configuring Spring Security.
Insight: Replaces `WebSecurityConfigurerAdapter` in Spring Security 5.7+.

---

17. Which class is used to encode passwords securely in Spring Boot?

A. `PlainTextEncoder`
B. `HashPasswordTool`
C. `BCryptPasswordEncoder`
D. `AESPasswordEncoder`

Answer: C
Explanation: Secure, salted hashing based on BCrypt algorithm.
Insight: Industry standard for password security.

---

18. Which configuration disables all security in a Spring Boot app?

A. `spring.security.enabled=false`
B. Custom `SecurityFilterChain` allowing all requests
C. `@DisableSecurity`
D. Delete the SecurityConfig file

Answer: B
Explanation: Define a chain like `.authorizeHttpRequests().anyRequest().permitAll()`.
Insight: Useful in local/test environments but dangerous in production.

---

19. How does form-based login differ from JWT-based login?

A. Form login uses cookies/session; JWT uses token in headers
B. JWT is encrypted, form login is not
C. Form login is stateless
D. JWT only works with database auth

Answer: A
Explanation: Form login creates sessions, JWT is stateless.
Insight: JWT suits APIs, SPAs, and mobile apps.

---

20. Which endpoint is used by default for login in Spring Security form-based login?

A. `/api/login`
B. `/login`
C. `/signin`
D. `/user/authenticate`

Answer: B
Explanation: Spring Security uses `/login` by default for form submissions.
Insight: Customize using `.loginPage("/custom")`.

---

21. How does Spring Security identify if a request is authenticated?

A. Checks JWT token signature
B. Validates `Authentication` in `SecurityContextHolder`
C. Queries the database
D. Matches session ID

Answer: B
Explanation: Authentication object is stored in thread-local for each request.
Insight: You can inject it using `@AuthenticationPrincipal`.

---

22. What is a benefit of stateless authentication (JWT)?

A. Simpler to implement
B. Requires less security
C. Scales well across multiple servers
D. Works only with session memory

Answer: C
Explanation: Stateless tokens eliminate need for session replication.
Insight: Ideal for distributed systems and microservices.

---

23. Which exception indicates an unauthenticated request?

A. `AccessDeniedException`
B. `AuthenticationException`
C. `IllegalArgumentException`
D. `SessionExpiredException`

Answer: B
Explanation: Indicates user is not authenticated.
Insight: Handled by `AuthenticationEntryPoint`.

---

24. Which exception indicates a request is authenticated but forbidden?

A. `AuthenticationException`
B. `AuthorizationException`
C. `AccessDeniedException`
D. `JWTException`

Answer: C
Explanation: Indicates valid credentials, but insufficient permissions.
Insight: Handled by `AccessDeniedHandler`.

---

25. How do you disable the default login form?

A. `http.login().disable()`
B. `http.formLogin().disable()`
C. `http.defaultForm(false)`
D. Remove `SecurityAutoConfig`

Answer: B
Explanation: Disables the form login filter.
Insight: Required when implementing custom login endpoints or using JWT.

25 MCQs — Spring Security with JWT/OAuth2: Real-World SSO

1. What makes JWT-based authentication stateless?

A. Uses cookies only
B. Requires JDBC token store
C. Doesn’t store session on server
D. Disables CSRF

Answer: C
Explanation: JWT tokens carry all necessary info and don’t rely on server-side session tracking.
Insight: Makes it ideal for scalable microservices and mobile applications.

---

2. Which header is used to send JWT tokens in HTTP requests?

A. `X-JWT-Token`
B. `Set-Cookie`
C. `Authorization: Bearer <token>`
D. `JWT-Key`

Answer: C
Explanation: JWT is typically passed in the `Authorization` header using the `Bearer`

scheme.

Insight: This is widely adopted and supported by Spring Security filters.

---

3. What does Spring Security's `OAuth2LoginAuthenticationFilter` do?

A. Validates JWT signature
B. Processes form login
C. Handles OAuth2 login redirects and user info
D. Caches token claims

Answer: C
Explanation: Manages OAuth2 login flows like redirects and authorization code exchange.
Insight: Used for social login integrations (Google, GitHub, etc.).

---

4. What component decodes and verifies JWT in Spring Security?

A. `JwtUtil`
B. `JwtDecoder`
C. `AuthenticationManager`
D. `AccessTokenFilter`

Answer: B
Explanation: `JwtDecoder` parses and validates the token's signature and claims.
Insight: Often configured with public keys or JWKS endpoint.

---

5. What is a common method to secure REST APIs with Spring and JWT?

A. `session.enable()`
B. `httpBasic()`
C. Use a `OncePerRequestFilter` for token validation
D. Use cookies only

Answer: C
Explanation: `OncePerRequestFilter` intercepts each request, extracts and validates JWT.
Insight: Works well with stateless microservices and reverse proxies.

---

6. What does the `aud` claim in a JWT represent?

A. Token expiration
B. Token issuer
C. Intended audience
D. Algorithm used

Answer: C
Explanation: Indicates the audience (e.g., API or service) the token is intended for.
Insight: Useful for multi-tenant or multi-service ecosystems.

---

7. Which claim identifies the user in a JWT?

A. `iss`
B. `sub`
C. `exp`
D. `azp`

Answer: B
Explanation: `sub` is the subject, typically mapped to user ID or username.
Insight: Used in access logs and user tracing.

---

8. How is a JWT token typically signed?

A. Plaintext with Base64
B. AES encryption
C. HMAC or RSA algorithms
D. SSL handshake

Answer: C
Explanation: JWTs are cryptographically signed using HMAC (symmetric) or RSA/EC (asymmetric).
Insight: Asymmetric keys are preferred for distributed systems (public/private key separation).

---

9. What is the purpose of `OAuth2ResourceServer` in Spring Security?

A. Authenticates users with login form
B. Issues tokens
C. Validates and parses access tokens in APIs
D. Stores refresh tokens

Answer: C
Explanation: Enables token-based authentication for APIs by validating JWTs or opaque tokens.
Insight: It's used for resource-side protection, not issuing tokens.

---

10. What's the difference between access token and refresh token in OAuth2?

A. Access token is encrypted, refresh is plain
B. Access token is long-lived
C. Access token grants access, refresh token obtains new access tokens
D. There's no difference

Answer: C
Explanation: Refresh tokens are used to request new access tokens without re-authenticating.
Insight: Enables long-lived sessions with short-lived access tokens.

---

11. What does the `iss` claim in a JWT mean?

A. Issuer of the token
B. Intended role
C. Hashing algorithm
D. Issuing timestamp

Answer: A
Explanation: Identifies who issued the token (e.g., `https://auth.mikcore.net`).
Insight: Often used for validating trust boundaries.

---

12. Which flow is used for server-side OAuth2 integrations?

A. Password grant
B. Client credentials
C. Authorization code
D. Implicit flow

Answer: C
Explanation: Authorization Code flow is the most secure and widely used with server backends.
Insight: Now often enhanced with PKCE for mobile and SPA security.

---

13. What is PKCE in OAuth2 used for?

A. Shortening token length
B. Protecting against code interception
C. Signing JWTs
D. Managing sessions

Answer: B
Explanation: Proof Key for Code Exchange adds extra validation step in OAuth2 Authorization Code Flow.
Insight: Mandatory for public clients like mobile apps.

---

14. What is `Spring Authorization Server` used for?

A. Encrypt JWT tokens
B. Act as a resource server
C. Issue and manage OAuth2 tokens and user consent
D. Manage database credentials

Answer: C

Explanation: It's a modern, extensible OAuth2/OpenID Connect server built on Spring Security.

Insight: Ideal for enterprises replacing Keycloak or Auth0 with in-house auth server.

---

15. Which endpoint is typically used to request a token in OAuth2?

A. `/user/token`
B. `/oauth/token`
C. `/auth/request`
D. `/token/grant`

Answer: B

Explanation: Standard OAuth2 endpoint to obtain access and refresh tokens.

Insight: Customizable in Spring Authorization Server.

---

16. How is user identity validated in a resource server with JWT?

A. Token is decrypted
B. Token is parsed and validated via `JwtDecoder`
C. Token is matched to DB
D. Token is base64 decoded only

Answer: B

Explanation: JWT is validated using signature and claims via `JwtDecoder`.

Insight: Use JWK Set URI or static public key.

---

17. What does `@EnableOAuth2ResourceServer` do?

A. Starts JWT signing
B. Creates login form
C. Enables bearer token validation for resource endpoints
D. Provides user login service

Answer: C

Explanation: Configures Spring Boot to expect and validate incoming access tokens.

Insight: Used on APIs, not on token issuing services.

---

18. Which of the following is a reason to use OAuth2 instead of Basic Auth?

A. Easier to implement
B. Session-based
C. Supports delegated authorization
D. Works only in mobile apps

Answer: C
Explanation: OAuth2 allows users to authorize third-party access without sharing passwords.
Insight: Enables secure SSO, federated login, and delegated API access.

---

19. What is the role of the `UserInfo` endpoint in OpenID Connect?

A. Refreshes tokens
B. Returns user details based on access token
C. Logs out user
D. Exchanges ID token

Answer: B
Explanation: Returns user claims (email, name, etc.) post authentication.
Insight: Often used after login for frontend population.

---

20. What is the function of a JWT refresh token handler in Spring Security?

A. Encrypt the JWT
B. Extend expiration of an expired token
C. Issue new access token using valid refresh token
D. Validate CSRF

Answer: C
Explanation: Refresh tokens allow re-authentication without asking user to login again.
Insight: Requires custom controller or Spring Authorization Server support.

---

21. Which filter intercepts bearer tokens for authentication in Spring Resource Server?

A. `JwtAuthenticationFilter`
B. `BearerTokenAuthenticationFilter`
C. `OAuth2LoginAuthenticationFilter`
D. `AccessTokenFilter`

Answer: B
Explanation: Parses `Authorization` header and validates the bearer token.
Insight: This is auto-configured in Spring Boot for stateless APIs.

---

22. Which OAuth2 flow is recommended for machine-to-machine communication?

A. Authorization Code
B. Implicit
C. Client Credentials
D. Password Grant

Answer: C
Explanation: No user context required — client authenticates using client_id + secret.
Insight: Used in internal service calls and backend jobs.

---

23. In Spring Authorization Server, what is the default token format?

A. Encrypted Base64
B. Opaque UUID
C. Signed JWT (JWS)
D. Session ID

Answer: C
Explanation: Tokens are JWTs signed using asymmetric or symmetric keys.
Insight: Easy to validate at resource server with no DB call.

---

24. What happens if JWT validation fails in resource server?

A. Request continues as anonymous
B. User is redirected to login
C. HTTP 401 Unauthorized is returned
D. Token is renewed

Answer: C
Explanation: Authentication fails and 401 is sent to client.
Insight: Logs often contain root cause like `invalid signature`.

---

25. Why is storing JWTs in localStorage or sessionStorage a potential risk?

A. Can’t be deleted
B. Exposes token to XSS attacks
C. Breaks CSRF protection
D. Causes JWTs to expire faster

Answer: B
Explanation: Any XSS exploit can access storage and steal tokens.
Insight: Consider HttpOnly cookies or secure vaults for high-security applications.

25 MCQs — Testing Spring Apps: CI-Ready Apps

1. What does `@SpringBootTest` do?

A. Loads only controller beans
B. Starts full Spring Boot context for integration testing
C. Starts only the data layer
D. Runs tests in parallel

Answer: B
Explanation: It loads the full application context, including all components and configurations.
Insight: Best for end-to-end or full-stack integration testing.

---

2. What is the purpose of `@WebMvcTest`?

A. Test database configurations
B. Load full application context
C. Test Spring MVC controllers with minimal beans
D. Test messaging queues

Answer: C
Explanation: Focused test slice that loads only the web layer (e.g., controllers, filters).
Insight: Great for fast and isolated REST API tests.

---

3. Which annotation is used to test JPA repositories in isolation?

A. `@JpaTest`
B. `@HibernateTest`
C. `@DataJpaTest`
D. `@EntityScan`

Answer: C
Explanation: Loads only JPA-related components and uses an in-memory database.
Insight: Speeds up testing compared to full context loading.

---

4. What does `@MockBean` do in Spring tests?

A. Creates Mockito mocks and adds to context
B. Creates fake beans for production
C. Replaces beans in test with dummy objects
D. Disables bean validation

Answer: A
Explanation: Used to replace a real bean with a mock in the test ApplicationContext.
Insight: Useful for unit testing services while isolating dependencies.

---

5. When is `@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)` useful?

A. When testing JSPs
B. When mocking web layer
C. When starting embedded server for integration tests
D. When avoiding DB tests

Answer: C
Explanation: Starts the web server on a random port so real HTTP calls can be made.
Insight: Great for testing REST endpoints end-to-end.

---

6. Which of the following is used to test REST controllers in isolation without loading the full context?

A. `@DataJpaTest`
B. `@RestControllerTest`
C. `@WebMvcTest`
D. `@SpringBootTest`

Answer: C
Explanation: `@WebMvcTest` is scoped to only the controller layer.
Insight: Combine with `@MockBean` to stub service dependencies.

---

7. What does `MockMvc` help with?

A. Mocking database
B. Making HTTP requests to controller endpoints without starting server
C. Starting a test container
D. Creating real-time logs

Answer: B
Explanation: Mocks the servlet container and tests controllers using a fluent API.
Insight: Enables fast controller testing with assertions on response status and content.

---

8. What annotation is typically used with `@DataJpaTest` to load additional config like custom converters?

A. `@Import`
B. `@EnableJpa`
C. `@TestConfiguration`
D. `@PropertySource`

Answer: A
Explanation: `@Import` adds custom beans to the limited test slice.
Insight: Helps test only what you need with precise dependencies.

---

9. Which embedded database is commonly used in Spring Boot tests?

A. MySQL
B. Oracle XE
C. H2
D. MongoDB

Answer: C
Explanation: H2 is in-memory, lightweight, and auto-configured in test environments.
Insight: Makes tests fast and avoids the need for external DB setup.

---

10. What is the purpose of `TestRestTemplate`?

A. Unit testing repositories
B. Mocking authentication
C. Making real HTTP calls during `@SpringBootTest`
D. Generating random data

Answer: C
Explanation: Injected by Spring Boot to test REST endpoints during full app tests.
Insight: Best for integration tests using actual ports.

---

11. What does `@Transactional` do in tests by default?

A. Commits data to the database
B. Disables SQL logging
C. Rolls back transactions after each test
D. Caches all queries

Answer: C
Explanation: Ensures test data doesn't pollute the database.
Insight: Useful when using real DBs to maintain clean state.

---

12. Which test slice is used to test configuration properties mapping?

A. `@ConfigurationPropertiesTest`
B. `@SpringBootTest`
C. `@TestConfiguration`
D. There is no dedicated slice — use `@SpringBootTest`

Answer: D
Explanation: Properties mapping is tested via integration-style tests with full context.
Insight: Unit test mapping logic separately if you want full isolation.

---

13. How do you test a controller that uses `@PathVariable` and `@RequestParam`?

A. Use `MockMvc` and `@WebMvcTest`
B. Use `@SpringBootTest` with `RestTemplate`
C. Manually test via Postman
D. Use `@DataJpaTest`

Answer: A
Explanation: MockMvc can simulate full HTTP requests with parameters.
Insight: Assertions include headers, content type, and HTTP status.

---

14. What does `@TestConfiguration` do?

A. Marks a class as test-only configuration
B. Enables security in tests
C. Mocks database
D. Adds retry logic to tests

Answer: A
Explanation: It is like `@Configuration` but only active during tests.
Insight: Use for test-specific beans (e.g., fake service, mock encoder).

---

15. How can you verify a service method was called in a Spring test?

A. Use `System.out.println()`
B. Use `Mockito.verify()`
C. Use `@SpyBean`
D. Use AOP

Answer: B
Explanation: `verify()` checks if a mock interaction occurred.
Insight: Works well with `@MockBean` injected into test context.

---

16. What does `@AutoConfigureMockMvc` do?

A. Enables RestTemplate mocks
B. Auto-wires `MockMvc` in tests
C. Configures database
D. Starts web server

Answer: B
Explanation: Automatically sets up `MockMvc` bean for use in tests.
Insight: Required when using `@SpringBootTest` and `MockMvc`.

---

17. What is the difference between `@MockBean` and `@Mock`?

A. No difference
B. `@MockBean` is managed by Spring, `@Mock` is not
C. `@Mock` is for fields only
D. `@MockBean` cannot be used in Spring Boot

Answer: B
Explanation: `@MockBean` integrates with Spring context; `@Mock` is pure Mockito.
Insight: Use `@MockBean` in Spring Boot tests; `@Mock` in pure unit tests.

---

18. How to simulate authenticated user in Spring MVC tests?

A. Use `@SecurityTestUser`
B. Use `@WithMockUser`
C. Use `@EnableMockSecurity`
D. Add JWT token

Answer: B
Explanation: Adds a mock authentication context for testing secured endpoints.
Insight: Simplifies tests that require roles or usernames.

---

19. What does `@Nested` in JUnit 5 help with?

A. Organize tests in inner classes
B. Enable parallel test runs
C. Register DBs
D. Enable security contexts

Answer: A
Explanation: Groups related tests logically and shares setup.
Insight: Helps with readability in larger test classes.

---

20. Which annotation disables auto-rollback in test DB transactions?

A. `@DisableRollback`
B. `@Commit`
C. `@ForceFlush`
D. `@EnablePersist`

Answer: B
Explanation: Forces Spring to commit the transaction after test execution.
Insight: Useful for populating data fixtures or debugging.

---

21. How do you override application properties in tests?

A. `@TestOverride`
B. `@PropertyOverride`
C. `@TestPropertySource`
D. `@MockProperty`

Answer: C
Explanation: Provides a property file or inline overrides for test environments.
Insight: Useful for mocking secrets or feature toggles.

---

22. Which annotation is used for parameterized tests in JUnit 5?

A. `@ParameterizedTest`
B. `@TestParams`
C. `@MultiTest`
D. `@RunWithParams`

Answer: A
Explanation: Allows running tests with different input values.
Insight: Works great with DTO validation, enum testing, etc.

---

23. What is the benefit of using test slices like `@DataJpaTest` and `@WebMvcTest`?

A. Enable full E2E tests
B. Reduce test startup time and resource usage
C. Require less configuration
D. Enable debugging

Answer: B
Explanation: Test slices limit the context to what's necessary for that layer.
Insight: Faster tests = better CI feedback loop.

---

24. How to assert JSON values in a controller test using `MockMvc`?

A. Use `assertEquals()` only
B. Use `.andExpect(jsonPath(...))`
C. Use `@ResponseTester`
D. Compare manually

Answer: B
Explanation: `jsonPath()` allows querying and validating JSON in the response.
Insight: Very powerful for REST endpoint validation.

---

25. What does `@BeforeEach` in JUnit 5 do?

A. Run once before all tests
B. Run once after all tests
C. Run before each test method
D. Initialize application context

Answer: C
Explanation: Prepares shared setup before every test.
Insight: Ensures consistent test initialization.

25 MCQs — Spring Boot Actuator & Monitoring: Production Insight

---

1. What is the purpose of Spring Boot Actuator?

A. Perform unit tests
B. Manage transactions
C. Expose production-ready endpoints (health, metrics, info)
D. Enable OAuth2

Answer: C
Explanation: Spring Boot Actuator provides built-in endpoints for monitoring and management.
Insight: Used heavily in cloud deployments and service health dashboards.

---

2. Which dependency must be added to enable actuator features?

A. `spring-boot-starter-test`
B. `spring-boot-actuator`
C. `spring-monitoring-starter`
D. `spring-boot-logging`

Answer: B
Explanation: This starter includes all core Actuator endpoints.
Insight: Often combined with Prometheus and Grafana via Micrometer.

---

3. What is the default path prefix for all actuator endpoints?

A. `/api/actuator`
B. `/monitoring`
C. `/actuator`
D. `/health`

Answer: C
Explanation: All endpoints like `/actuator/health`, `/actuator/metrics` are exposed under this prefix.
Insight: Can be changed using `management.endpoints.web.base-path`.

---

4. Which endpoint provides service availability info?

A. `/actuator/info`
B. `/actuator/env`
C. `/actuator/metrics`
D. `/actuator/health`

Answer: D
Explanation: `/actuator/health` shows the current status of the application (e.g., `UP`, `DOWN`).
Insight: Used by Kubernetes, AWS ALB, or Cloud Foundry for readiness checks.

---

5. What tool does Spring Boot Actuator use internally to collect metrics?

A. Dropwizard
B. SLF4J
C. Micrometer
D. Logback

Answer: C
Explanation: Micrometer is a facade that bridges to monitoring systems like Prometheus, Datadog, etc.
Insight: Enables vendor-neutral instrumentation.

---

6. Which property enables all actuator endpoints by default?

A. `endpoints.all.enabled=true`
B. `management.endpoints.web.expose=*`
C. `actuator.expose=true`
D. `actuator.api.all=true`

Answer: B
Explanation: This exposes all web-based actuator endpoints.
Insight: Limit in production to expose only required endpoints for security.

---

7. How do you register a custom health indicator?

A. Extend `AbstractHealthCheck`
B. Implement `HealthCheckHandler`
C. Implement `HealthIndicator` interface
D. Add @CustomHealth on method

Answer: C
Explanation: Spring auto-detects beans implementing `HealthIndicator`.
Insight: Helps track DB status, external APIs, or internal queues.

---

8. What does the `/actuator/info` endpoint show?

A. Memory info
B. Actuator config
C. Application metadata like version, build, etc.
D. Health status

Answer: C
Explanation: Pulls info from `application.properties` or build info.
Insight: Good for runtime traceability and release tracking.

---

9. What file is used to populate build metadata in `/actuator/info`?

A. `info.yaml`
B. `build.gradle`
C. `pom.xml`
D. `META-INF/build-info.properties`

Answer: D
Explanation: Created during build with Maven/Gradle plugins.
Insight: Automate versioning pipelines with this data.

---

10. Which Micrometer registry is used for Prometheus integration?

A. `GraphiteMeterRegistry`
B. `SimpleMeterRegistry`
C. `PrometheusMeterRegistry`
D. `JMXMeterRegistry`

Answer: C
Explanation: Enables Prometheus to scrape metrics from Spring Boot.
Insight: Integrates with Kubernetes dashboards or Grafana charts.

---

11. What does the `/actuator/metrics` endpoint do?

A. Shows CPU usage
B. Returns list of all available metrics
C. Dumps JVM heap
D. Validates metrics configuration

Answer: B
Explanation: Lists all meter names; you can query specifics like `/actuator/metrics/jvm.memory.used`.
Insight: Drill-down into app performance in real time.

---

12. How do you add a custom metric counter in code?

A. `Metrics.addCounter()`
B. `meterRegistry.counter("my.custom.metric").increment()`
C. `@TrackMetric`
D. `Micrometer.counter().add()`

Answer: B
Explanation: Micrometer provides fluent APIs via `MeterRegistry`.
Insight: Use for business metrics like login attempts or payment failures.

---

13. What is `MeterRegistry` in Spring Boot?

A. DB connection handler
B. Health check configuration
C. Micrometer's central interface to register and track metrics
D. Logging factory

Answer: C
Explanation: `MeterRegistry` is auto-configured by Spring Boot for collecting all metrics.
Insight: Supports multiple backends via composite registries.

---

14. What is the default behavior of `/actuator/health` when DB is down?

A. Shows `UP`
B. Shows `DOWN`
C. Skips DB check
D. Shows `UNKNOWN`

Answer: B
Explanation: Spring includes health checks for DB, so if unavailable, it shows `DOWN`.
Insight: Can be customized via `HealthIndicator`.

---

15. How can you restrict actuator endpoints to specific IPs?

A. Use `.secureEndpoints()`
B. Use Spring Security config
C. Set `ipFilter=true`
D. It is not possible

Answer: B
Explanation: Secure endpoints via Spring Security rules (e.g., `.requestMatchers("/actuator/**").hasIpAddress(...)`).
Insight: Always secure actuator endpoints in production.

---

16. What does `@EventListener(ApplicationReadyEvent.class)` do?

A. Runs code after app is fully started
B. Enables metrics
C. Triggers health check
D. Loads all beans again

Answer: A
Explanation: Runs logic after the application context is ready.
Insight: Common for warm-up logic or logging system boot time.

---

17. Which of these is an application lifecycle event in Spring Boot?

A. `ApplicationReadyEvent`
B. `ServerLoadedEvent`
C. `WebReadyEvent`
D. `BootInitEvent`

Answer: A
Explanation: Published after the app is fully initialized and web server started.
Insight: Use it to start background tasks or diagnostics.

---

18. How can you expose only the `health` and `info` actuator endpoints?

A. `management.expose.limited=true`
B. `management.endpoints.web.expose=health,info`
C. `actuator.safeEndpointsOnly=true`
D. `management.limit.endpoints=true`

Answer: B
Explanation: Fine-grained control over what endpoints are available.
Insight: Minimum safe set for production is usually `health`, `info`.

---

19. Which actuator endpoint shows memory, CPU, and thread usage?

A. `/actuator/sys`
B. `/actuator/env`
C. `/actuator/metrics`
D. `/actuator/jvm`

Answer: C
Explanation: `/metrics/jvm.memory.used`, `/metrics/system.cpu.usage` etc. are available via Micrometer.
Insight: Helps correlate application slowdowns with resource pressure.

---

20. What property exposes Prometheus metrics endpoint?

A. `management.prometheus.enabled=true`
B. `management.endpoint.prometheus.enabled=true`
C. `actuator.metrics.prometheus=true`
D. `prometheus.export=true`

Answer: B
Explanation: Enables `/actuator/prometheus` for scraping by Prometheus.
Insight: Used with `prometheus.yml` scrape configs.

---

21. Which tag is automatically added to Micrometer metrics by default in Spring Boot?

A. `instanceId`
B. `spring.application.name`
C. `env.version`
D. `micrometer.source`

Answer: B
Explanation: Adds app name from properties, useful for identifying metrics source.
Insight: Add custom tags like region, zone for multi-region analysis.

---

22. What actuator endpoint can be used to inspect environment properties?

A. `/actuator/env`
B. `/actuator/config`
C. `/actuator/info`
D. `/actuator/debug`

Answer: A
Explanation: Shows properties from environment and configuration files.
Insight: Useful for diagnosing incorrect configuration.

---

23. Which dependency is required to enable Spring Boot build info generation?

A. `spring-boot-buildinfo-plugin`
B. `spring-boot-maven-plugin` or `spring-boot-gradle-plugin`
C. `build-info-compiler`
D. `actuator-metadata-plugin`

Answer: B
Explanation: These plugins generate `build-info.properties`.
Insight: Integrates with CI/CD pipelines to publish build versions.

---

24. What's the default HTTP status of `/actuator/health` when app is UP?

A. 202
B. 302
C. 200
D. 204

Answer: C
Explanation: Returns 200 OK with health status JSON payload.
Insight: Some platforms support `readiness` vs. `liveness` probes separately.

---

25. What's the difference between `readiness` and `liveness` probes in monitoring?

A. Liveness = app is dead, Readiness = app can't serve traffic
B. Both mean the same
C. Readiness checks DB only
D. Liveness checks logs

Answer: A
Explanation: Liveness = app must be restarted; Readiness = app is temporarily unavailable (e.g., loading cache).
Insight: Kubernetes uses this to decide restart vs. load balancing decisions.

25 MCQs — Scheduling & Async: Non-blocking Tasks

---

1. What is the purpose of `@Scheduled` in Spring?

A. Handle form validation
B. Define routes for REST controllers
C. Schedule method execution at fixed intervals
D. Manage database transactions

Answer: C
Explanation: `@Scheduled` marks a method to be invoked periodically based on time settings.
Insight: Commonly used for batch jobs, file scans, or cleanup tasks.

---

2. Which annotation enables Spring's scheduling support?

A. `@EnableScheduling`
B. `@SchedulingEnabled`
C. `@StartScheduler`
D. `@ScheduleOnBoot`

Answer: A
Explanation: This enables the detection of `@Scheduled` annotations.
Insight: Without it, scheduled methods won't run even if annotated.

---

3. What is the format of a Spring cron expression?

A. `second minute hour day-of-month month day-of-week`
B. `hour:minute:second`
C. `minute/hour/day`
D. `cron(* * *)`

Answer: A
Explanation: Standard cron format with six fields.
Insight: Can express complex scheduling like "every Monday at 2am".

---

4. What does `@Scheduled(fixedRate = 5000)` mean?

A. Executes every 5 seconds after the last start time
B. Executes once after 5 seconds
C. Executes 5 times total
D. Executes every 5 seconds after the last end time

Answer: A
Explanation: `fixedRate` runs at a fixed interval regardless of method execution time.
Insight: Use carefully when method takes long to complete.

---

5. What does `@Scheduled(fixedDelay = 5000)` do?

A. Delays execution forever
B. Runs every 5s after last invocation completes
C. Uses cron internally
D. Only works in `@Service` classes

Answer: B
Explanation: `fixedDelay` ensures the delay happens after task completion.
Insight: Use for long-running, sequential background tasks.

---

6. What annotation marks a method for asynchronous execution?

A. `@Background`
B. `@Async`
C. `@ScheduledAsync`
D. `@Parallel`

Answer: B
Explanation: `@Async` makes the method execute in a separate thread.
Insight: Enables non-blocking I/O, parallel tasks, or UI responsiveness.

---

7. Which annotation enables Spring's async method execution capability?

A. `@EnableThreads`
B. `@AsyncEnabled`
C. `@EnableAsync`
D. `@EnableParallelism`

Answer: C
Explanation: Activates Spring’s async execution using proxies.
Insight: Required in configuration classes or `@SpringBootApplication`.

---

8. What is the default return type supported by `@Async` methods?

A. `void`
B. `Future`
C. `CompletableFuture`
D. All of the above

Answer: D
Explanation: `@Async` supports `void`, `Future<T>`, `CompletableFuture<T>`, and `ListenableFuture<T>`.
Insight: Use `CompletableFuture` for fluent composition.

---

9. What happens if an `@Async` method returns `void`?

A. It throws an exception
B. It blocks until completion
C. It runs asynchronously, but no result is tracked
D. It returns `null` immediately

Answer: C
Explanation: Execution is fire-and-forget.
Insight: Good for logging, notifications, or one-way jobs.

---

10. How do you customize the thread pool for async tasks?

A. Set `max-threads` in application.properties
B. Override `TaskExecutor` bean
C. Use `@Async(maxPool=10)`
D. You cannot customize it

Answer: B
Explanation: Define a `@Bean` of type `TaskExecutor` and annotate with `@Primary`.
Insight: Avoid unbounded threads in production workloads.

---

11. Which interface is commonly used to define async thread pools in Spring?

A. `ExecutorService`
B. `TaskExecutor`
C. `ThreadManager`
D. `RunnablePool`

Answer: B
Explanation: `TaskExecutor` is a Spring abstraction over `Executor`.
Insight: Use `ThreadPoolTaskExecutor` for configuration flexibility.

---

12. What happens if an exception occurs inside an `@Async` method returning `CompletableFuture`?

A. It is ignored
B. It is swallowed
C. It is captured and available in `future.get()`
D. It is thrown to the caller immediately

Answer: C
Explanation: Exceptions are wrapped and can be accessed using future callbacks or `.exceptionally()`.
Insight: Essential for robust async programming.

---

13. What is the default task executor used by `@Async` if none is provided?

A. `ForkJoinPool`
B. `CachedThreadPool`
C. `SimpleAsyncTaskExecutor`
D. `ThreadPoolTaskExecutor`

Answer: C
Explanation: It's not backed by a real thread pool and creates new threads for every task.
Insight: Not suitable for production – always define a pool.

---

14. What annotation combination runs a scheduled task asynchronously?

A. `@Scheduled @Async`
B. `@EnableAsync @Async`
C. `@EnableScheduling @Scheduled`
D. All of the above

Answer: A
Explanation: Combining both allows the scheduled task to run in a separate thread.
Insight: Prevents blocking the default task scheduler thread.

---

15. What's the default timezone for Spring `@Scheduled(cron = "…")`?

A. UTC
B. JVM local timezone
C. System timezone
D. Asia/Kolkata

Answer: B
Explanation: By default, cron expressions use the JVM's local time.
Insight: Use `zone = "UTC"` to standardize scheduling across environments.

---

16. What is `ScheduledExecutorService` used for in Spring?

A. Bootstrapping application
B. Low-level cron management
C. Custom scheduling backend
D. Logging scheduler jobs

Answer: C
Explanation: You can register it as the scheduler backend with `@EnableScheduling`.
Insight: Used when precise control over scheduling threads is needed.

---

17. What happens if a scheduled method throws an exception?

A. Job halts permanently
B. Spring stops the app
C. Logs the exception and continues scheduling
D. Restarts the app

Answer: C
Explanation: By default, exceptions are logged but do not stop future executions.
Insight: Add error-handling logic to prevent silent failures.

---

18. How do you cancel a running `@Scheduled` task dynamically?

A. Remove annotation
B. Kill the process
C. Use `ScheduledFuture.cancel()`
D. Change cron expression at runtime

Answer: C
Explanation: If you schedule programmatically, `ScheduledFuture` allows dynamic

cancellation.

Insight: Useful for runtime control of long-running jobs.

---

19. How can you avoid overlapping executions of the same scheduled task?

A. Use `@Async`
B. Use distributed locks or `SchedulerLock` library
C. Use `@NonBlocking`
D. Use `Thread.sleep()`

Answer: B
Explanation: Spring does not prevent overlaps; you must manage concurrency manually.
Insight: Useful in clustered environments (e.g., Redis lock, ShedLock).

---

20. How can you ensure async exceptions are logged globally?

A. Use `@ExceptionHandler`
B. Configure `AsyncUncaughtExceptionHandler`
C. Log manually
D. Use `ThreadPoolExecutor#afterExecute()`

Answer: B
Explanation: Spring lets you define global async exception handlers via `AsyncConfigurer`.
Insight: Prevents silent failures in production environments.

---

21. Which is a valid Spring cron expression for "every Monday at 5:30 AM"?

A. `30 5 * * 1`
B. `0 30 5 * * MON`
C. `5 30 0 MON *`
D. `30 5 0 1 *`

Answer: B
Explanation: Format is `second minute hour day-of-month month day-of-week`.
Insight: Use cron expression testers to avoid misfires.

---

22. What is the advantage of using `CompletableFuture` over `Future`?

A. `CompletableFuture` is synchronous
B. It supports chaining and functional programming
C. It uses fewer threads
D. It blocks on `get()`

Answer: B
Explanation: Enables non-blocking, fluent async programming.
Insight: Use `.thenApply()`, `.thenCompose()`, `.exceptionally()` for reactive flows.

---

23. What is a good practice when creating custom thread pools in production?

A. Use unbounded pools
B. Use one thread per job
C. Set core pool size, max size, and queue size
D. Disable core thread timeout

Answer: C
Explanation: Prevents uncontrolled thread creation and OOM issues.
Insight: Monitor thread usage and tune for system load.

---

24. Which class helps schedule tasks programmatically in Spring?

A. `TaskManager`
B. `ScheduledThreadManager`
C. `TaskScheduler`
D. `AsyncManager`

Answer: C
Explanation: Interface to programmatically schedule future tasks.
Insight: Useful when schedule times are dynamic or configurable.

---

25. What happens if `@Scheduled(cron = "...")` is misconfigured?

A. Skips execution
B. Executes with default schedule
C. Throws `IllegalArgumentException` at startup
D. Logs warning only

Answer: C
Explanation: Spring fails fast on invalid cron expressions.
Insight: Always validate cron values before deployment.

25 MCQs — Spring Events & ApplicationContextPublisher: Decoupled Signals

---

1. What is the purpose of Spring's event mechanism?

A. Logging exceptions
B. Decoupling components using publish-subscribe pattern
C. Managing database entities
D. Scheduling tasks

Answer: B
Explanation: Spring's event mechanism allows one component to notify others without direct dependencies.
Insight: Ideal for audit logging, notifications, and state transitions.

---

2. Which interface is used to publish events in Spring?

A. `ApplicationEventPublisher`
B. `EventNotifier`
C. `MessageBroadcaster`
D. `EventContext`

Answer: A
Explanation: Spring injects this interface into beans to publish custom or built-in events.
Insight: Use it within services to raise domain signals cleanly.

---

3. How do you define a Spring event listener method?

A. `@EventHandler`
B. `@Observer`
C. `@Subscribe`
D. `@EventListener`

Answer: D
Explanation: `@EventListener` marks a method that will receive Spring events.
Insight: Supports filtering, async handling, and ordering.

---

4. What is a common use case for Spring application events?

A. Scheduling cron jobs
B. Decoupled audit logging after domain changes
C. Creating REST controllers
D. Defining DTOs

Answer: B
Explanation: Listeners can log or trigger actions without tightly coupling logic.
Insight: Reduces direct wiring of dependent components.

---

5. What is the base class for custom Spring events?

A. `SpringEvent`
B. `EventObject`
C. `ApplicationEvent` *(deprecated)*
D. No base class required in recent versions

Answer: D
Explanation: Since Spring 4.2, you can publish any object as an event — POJOs are supported.
Insight: No need to extend `ApplicationEvent` unless you rely on its timestamp or legacy behavior.

---

6. How are Spring events dispatched by default?

A. Parallel threads
B. Reactive pipelines
C. Synchronously in the publisher thread
D. Sent via HTTP

Answer: C
Explanation: Listeners are invoked in the same thread by default.
Insight: Use `@Async` for non-blocking listeners.

---

7. What happens if no listener is found for an event?

A. Throws exception
B. Logs a warning
C. Silently ignores
D. Retries publishing

Answer: C
Explanation: Events without listeners are ignored without error.
Insight: Safe to use events even with optional consumers.

---

8. How do you make a Spring event listener asynchronous?

A. `@EventListener(async = true)`
B. `@Async` on method + `@EnableAsync`
C. `@ParallelListener`
D. Use custom threads

Answer: B
Explanation: Combine `@EventListener` with `@Async` and enable async support.
Insight: Avoid blocking the main thread during slow operations like email or file processing.

---

9. Which Spring context method is used to publish events manually?

A. `ApplicationContext.publishEvent(Object event)`
B. `application.publish()`
C. `eventContext.fire()`
D. `Spring.publish()`

Answer: A
Explanation: The `ApplicationContext` itself implements `ApplicationEventPublisher`.
Insight: Often injected into services when direct access to context is needed.

---

10. What type of method signature does `@EventListener` expect?

A. No arguments
B. Return type only
C. One parameter = the event type
D. Two parameters (event, context)

Answer: C
Explanation: Spring maps the parameter to event type automatically.
Insight: Enables strong typing and automatic dispatch.

---

11. Can you listen to multiple event types in one method?

A. No
B. Yes, using multiple `@EventListener` annotations
C. Yes, with `@EventListener(classes = {...})`
D. Only with reflection

Answer: C
Explanation: `@EventListener(classes = {EventA.class, EventB.class})` enables multi-type listeners.
Insight: Useful for similar handling logic across events.

---

12. How do you prevent a listener from running for specific event conditions?

A. Throw an exception
B. Add condition with `@EventListener(condition = "...")`
C. Use `@SkipListener`
D. Remove the bean manually

Answer: B
Explanation: Supports SpEL conditions like `event.type == 'SUCCESS'`.
Insight: Helps avoid unnecessary processing.

---

13. What is the purpose of `ApplicationContextAware`?

A. Autowires Spring Boot configurations
B. Manually access `ApplicationContext`
C. Enables asynchronous execution
D. Loads YAML files

Answer: B
Explanation: Allows a bean to gain access to the `ApplicationContext` programmatically.
Insight: Common workaround in legacy or plugin-style architectures.

---

14. Which of the following events is published by Spring on startup?

A. `ContextStartedEvent`
B. `ApplicationStartedEvent`
C. `ApplicationReadyEvent`
D. All of the above

Answer: D
Explanation: Spring fires multiple lifecycle events during boot.
Insight: Hook into these to trigger diagnostics, background processes, etc.

---

15. What is `ApplicationEventMulticaster` used for?

A. Sends events over Kafka
B. Delivers events to all listeners
C. Logs application events
D. Handles RESTful errors

Answer: B
Explanation: Manages listener registration and dispatching.
Insight: Can be replaced to control threading behavior globally.

---

16. How does Spring determine listener order?

A. Random order
B. Based on @Order or `Ordered` interface
C. Alphabetical class name
D. Java reflection priority

Answer: B
Explanation: Listeners can use `@Order` or implement `Ordered` for precedence.
Insight: Critical when one listener depends on the output of another.

---

17. What happens when an async listener throws an exception?

A. It is logged and suppressed
B. All listeners stop
C. Publisher fails
D. Application exits

Answer: A
Explanation: Exceptions are swallowed by default, but you can configure global exception handling.
Insight: Consider logging failures explicitly in async tasks.

---

18. Can Spring events cross microservice boundaries?

A. Yes, automatically
B. Only with Kafka or external brokers
C. Yes, with `@EventBridge`
D. Spring converts them to REST calls

Answer: B
Explanation: Local events are in-process only; use messaging for distributed events.
Insight: Use Spring Cloud Stream, Kafka, or RabbitMQ for cross-service signaling.

---

19. What is the benefit of using events in DDD (Domain-Driven Design)?

A. Improves DB indexing
B. Isolates domain concerns
C. Simplifies REST routing
D. Avoids exception handling

Answer: B
Explanation: Events promote separation of concerns in bounded contexts.
Insight: Enables eventual consistency and decoupled workflows.

---

20. What's the default scope of event listeners in Spring?

A. Singleton
B. Prototype
C. Request
D. Stateless

Answer: A
Explanation: Event listeners are singleton beans unless otherwise specified.
Insight: No new listener is created per event.

---

21. Can you receive events in non-Spring beans?

A. Yes, always
B. No
C. Only with manual registration
D. Only if using `@ComponentScan`

> **Answer**: C
> **Explanation**: You can register external listeners using `ApplicationEventMulticaster`.
> **Insight**: Useful in plugin architectures or test-only listeners.

---

22. Which event can you listen to for graceful shutdown logic?

A. `ApplicationClosingEvent`
B. `ContextStoppedEvent`
C. `ContextClosedEvent`
D. `AppShutdownEvent`

> **Answer**: C
> **Explanation**: Fired when Spring context is closing.
> **Insight**: Use to release resources, close DBs, or flush logs.

---

23. How can you listen to all events regardless of type?

A. Register a listener for `Object.class`
B. Use `@EventListener(Object.class)`
C. Implement `ApplicationListener<ApplicationEvent>`
D. You cannot

> **Answer**: C
> **Explanation**: Catch-all listener receives all `ApplicationEvent` instances.
> **Insight**: Use with caution to avoid tight coupling or performance hits.

---

24. What is an example of a built-in Spring event?

A. `OrderCreatedEvent`
B. `ContextRefreshedEvent`
C. `JwtTokenExpiredEvent`
D. `UserSessionClosedEvent`

> **Answer**: B
> **Explanation**: Spring publishes core lifecycle events like `ContextRefreshedEvent`.
> **Insight**: Use for bean post-initialization hooks.

---

25. What interface should a custom listener implement for type-safe listening?

A. `ApplicationEvent`
B. `EventSubscriber`
C. `ApplicationListener<T>`
D. `SpringObserver<T>`

Answer: C
Explanation: `ApplicationListener<MyEvent>` receives only matching events.
Insight: Can be registered programmatically or picked up via component scan.

25 MCQs — Spring WebFlux (Reactive): High Throughput

---

1. What is the key difference between Spring MVC and Spring WebFlux?

A. WebFlux uses HTTP2
B. WebFlux is based on reactive programming and non-blocking I/O
C. WebFlux doesn’t support annotations
D. WebFlux only works with databases

Answer: B
Explanation: WebFlux leverages Project Reactor for non-blocking, asynchronous processing.
Insight: Ideal for streaming data, high-load APIs, or WebSockets.

---

2. What is `Mono` in Spring WebFlux?

A. A wrapper for arrays
B. A future-like type that returns 0 or 1 element asynchronously
C. A stream of multiple values
D. A synchronous wrapper

Answer: B
Explanation: `Mono<T>` represents an asynchronous computation with zero or one result.
Insight: Used for REST responses like user details or status checks.

---

3. What is `Flux` in Spring WebFlux?

A. Similar to a traditional list
B. Represents zero or more values in a reactive stream
C. Works only with JSON
D. Is a synchronous wrapper

Answer: B
Explanation: `Flux<T>` emits multiple values asynchronously, like a reactive stream.
Insight: Ideal for real-time feeds, streaming APIs, or large dataset responses.

---

4. Which module enables reactive REST support in Spring Boot?

A. `spring-boot-starter-web`
B. `spring-webflux`
C. `spring-reactive-core`
D. `spring-stream-reactive`

Answer: B
Explanation: This includes `WebFlux`, Reactor, and Netty-based server support.
Insight: Lightweight alternative to the traditional `spring-boot-starter-web`.

---

5. What is the default embedded server for Spring WebFlux?

A. Tomcat
B. Jetty
C. Undertow
D. Netty

Answer: D
Explanation: Netty is non-blocking and supports reactive backpressure natively.
Insight: Allows event-driven request handling without servlet container overhead.

---

6. Which type does `@GetMapping` return in a WebFlux controller?

A. `String`
B. `List`
C. `Mono<T>` or `Flux<T>`
D. `Optional`

Answer: C
Explanation: Reactive endpoints return `Mono` or `Flux` to allow async processing.
Insight: Enables responsive APIs that don't block threads.

---

7. What is `WebClient` used for in WebFlux?

A. Building HTML views
B. Making reactive, non-blocking HTTP calls
C. Sending email
D. Managing WebSockets

Answer: B
Explanation: Replaces `RestTemplate` in reactive environments.
Insight: Supports chaining, error handling, retries, and streaming.

---

8. What's a key benefit of WebClient over RestTemplate?

A. Easier syntax
B. Blocking model
C. Reactive, non-blocking I/O
D. XML support

Answer: C
Explanation: WebClient integrates with Project Reactor for non-blocking HTTP calls.
Insight: Enables parallel downstream calls without thread contention.

---

9. What does `.subscribe()` do on a `Mono` or `Flux`?

A. Cancels the stream
B. Collects the data synchronously
C. Triggers execution of the reactive pipeline
D. Stops the stream

Answer: C
Explanation: Without subscribing, the pipeline doesn't execute.
Insight: Spring handles subscription internally in WebFlux controllers.

---

10. What operator converts a Flux to a Mono?

A. `.flatten()`
B. `.cache()`
C. `.collectList()`
D. `.block()`

Answer: C
Explanation: Collects all emitted elements into a single `List<T>` inside a `Mono`.
Insight: Useful when aggregating multiple values before returning.

---

11. How do you define a reactive REST controller in WebFlux?

A. `@ReactiveController`
B. `@RestController` with `Mono/Flux` return types
C. `@WebFluxController`
D. `@AsyncController`

Answer: B
Explanation: The standard `@RestController` is used, but with reactive return types.
Insight: Keeps consistency between Spring MVC and WebFlux annotations.

---

12. What does `.block()` do on a `Mono`?

A. Blocks until result is available
B. Cancels execution
C. Subscribes with timeout
D. Schedules a new thread

Answer: A
Explanation: `block()` forces a synchronous wait, breaking reactivity.
Insight: Avoid in production, especially inside reactive flows.

---

13. Which annotation is used to enable WebFlux configuration?

A. `@EnableReactiveWeb`
B. `@EnableWebFlux`
C. `@WebFluxConfig`
D. `@EnableReactiveApp`

Answer: B
Explanation: Enables manual configuration of Spring WebFlux behavior.
Insight: Rarely needed when using Spring Boot autoconfiguration.

---

14. What's the use of `.flatMap()` in reactive programming?

A. Concatenates two strings
B. Maps and flattens nested `Mono` or `Flux`
C. Blocks the pipeline
D. Fetches data from DB

Answer: B
Explanation: Used to flatten and transform nested async types.
Insight: Core operator in chaining dependent reactive calls.

---

15. Which operator allows handling errors in a Mono/Flux pipeline?

A. `.filter()`
B. `.catch()`
C. `.onErrorResume()`
D. `.failSafe()`

Answer: C
Explanation: Allows fallback or recovery when an error occurs.
Insight: Enables resilient and fault-tolerant reactive pipelines.

---

16. What is backpressure in WebFlux?

A. Delayed HTTP response
B. Data pushed beyond consumer capacity
C. Thread overflow
D. Stack memory overflow

Answer: B
Explanation: Backpressure manages flow control to avoid overwhelming consumers.
Insight: Helps build stable streaming systems under load.

---

17. What happens if you return `Mono.empty()` in a controller?

A. Returns 204 No Content
B. Returns null
C. Throws exception
D. Returns 500

Answer: A
Explanation: Spring interprets `Mono.empty()` as 204 response.
Insight: Useful for DELETE or conditional responses.

---

18. What is `ServerResponse` used for in WebFlux functional endpoints?

A. Replace model attributes
B. Create REST response in functional-style routes
C. Handle DB transactions
D. Configure logging

Answer: B
Explanation: Used in functional routing (non-annotation based) to return responses.
Insight: Gives full control over headers, body, and status.

---

19. Which WebFlux component defines the route-to-handler mapping in functional style?

A. `RouterFunction`
B. `WebClient`
C. `RestTemplate`
D. `RequestPredicate`

Answer: A
Explanation: `RouterFunction` maps HTTP routes to handler functions.
Insight: Alternative to annotations — clean for DSL-style APIs.

---

20. What does `.concatWith()` do on a Flux?

A. Merges values randomly
B. Combines another Flux in sequence
C. Removes duplicates
D. Triggers async error

Answer: B
Explanation: Adds emissions from another publisher after the current one completes.
Insight: Enables chained processing of multiple data sources.

---

21. Which method retries a failed Mono up to N times?

A. `.onErrorRetry()`
B. `.retry(n)`
C. `.failSafe(n)`
D. `.catch().loop(n)`

Answer: B
Explanation: `.retry(n)` re-subscribes up to `n` times on error.
Insight: Add backoff logic to avoid tight retry loops.

---

22. What is the use of `.zipWith()` in Reactor?

A. Combines multiple Mono/Flux streams
B. Validates stream
C. Merges response headers
D. Checks compression

Answer: A
Explanation: `zipWith` pairs values from two publishers.
Insight: Useful in combining user info and roles, etc.

---

23. What's the return type for a streaming API endpoint in WebFlux?

A. `Stream<T>`
B. `CompletableFuture<T>`
C. `Flux<T>`
D. `Mono<byte[]>`

Answer: C
Explanation: Streaming data over time is represented by `Flux`.
Insight: Perfect for server-sent events and log feeds.

---

24. Which status code is returned when a Flux is empty?

A. 200 OK
B. 404 Not Found
C. 204 No Content
D. 500 Internal Error

Answer: A
Explanation: An empty Flux still returns 200 unless explicitly mapped.
Insight: Use `.switchIfEmpty()` to return a custom status or message.

---

25. What operator allows conditional transformation based on a predicate?

A. `.onErrorResume()`
B. `.switchIfEmpty()`
C. `.filter()`
D. `.transformWhen()`

Answer: C
Explanation: Filters out values not satisfying the predicate.
Insight: Helps shape the data flow early in the pipeline.

25 MCQs — Spring Boot with Kafka/RabbitMQ: Event-driven Architecture

---

1. What is the purpose of using Kafka or RabbitMQ with Spring Boot?

A. Manage databases
B. Handle synchronous REST calls
C. Enable asynchronous, decoupled communication between services
D. Generate Swagger docs

Answer: C
Explanation: Messaging platforms enable event-driven, non-blocking service-to-service communication.
Insight: Great for scalable microservices and real-time data pipelines.

---

2. What annotation is used to define a Kafka listener in Spring Boot?

A. `@KafkaListener`
B. `@KafkaConsumer`
C. `@StreamListener`
D. `@MessageListener`

Answer: A
Explanation: `@KafkaListener` designates methods that consume Kafka messages.
Insight: You can assign topics, group IDs, and container settings.

---

3. What is the equivalent annotation for RabbitMQ consumers in Spring AMQP?

A. `@RabbitConsumer`
B. `@RabbitHandler`
C. `@RabbitListener`
D. `@AMQPListener`

Answer: C
Explanation: Similar to Kafka, Spring AMQP uses `@RabbitListener` for consumer methods.
Insight: You can bind queues, routing keys, and exchanges declaratively.

---

4. What does a Kafka producer do?

A. Receives messages from consumers
B. Sends messages to a topic
C. Stores data in databases
D. Creates queues

Answer: B
Explanation: Kafka producers write messages to specified topics.
Insight: Producers are often service or API components generating events.

---

5. What property sets the Kafka broker address in Spring Boot?

A. `kafka.bootstrap.hosts`
B. `bootstrap.kafka.servers`
C. `spring.kafka.bootstrap-servers`
D. `spring.kafka.broker-url`

Answer: C
Explanation: This tells Spring where to connect the Kafka client.
Insight: Can be set to multiple comma-separated brokers for HA.

---

6. In RabbitMQ, what's the role of a queue?

A. Stores messages temporarily until consumed
B. Encrypts payloads
C. Maps URLs to services
D. Hosts a database

Answer: A
Explanation: Messages are delivered to queues, where consumers can fetch them.
Insight: RabbitMQ's push model differs from Kafka's pull-based model.

---

7. What is a "topic" in Kafka?

A. A type of message
B. A unique key
C. A logical channel for organizing messages
D. A schema validator

Answer: C
Explanation: Topics allow producers and consumers to organize and filter messages.
Insight: Used like "event categories" in pub-sub models.

---

8. What is the default serialization format in Spring Kafka?

A. XML
B. ByteBuffer
C. String (via StringSerializer)
D. JSON

Answer: C
Explanation: Spring Kafka uses `StringSerializer` and `StringDeserializer` by default.
Insight: You can override to use JSON, Avro, or custom formats.

---

9. Which Spring project provides Kafka support?

A. `spring-kafka-streams`
B. `spring-integration`
C. `spring-cloud-kafka`
D. `spring-kafka`

Answer: D
Explanation: `spring-kafka` offers annotation support, auto-config, templates, and listener containers.
Insight: Also integrates with Spring Boot autoconfiguration.

---

10. What class is used to send messages to Kafka in Spring Boot?

A. `KafkaConnector`
B. `KafkaPublisher`
C. `KafkaTemplate`
D. `KafkaStreamBuilder`

Answer: C
Explanation: `KafkaTemplate` is the primary producer utility.
Insight: Works similarly to `JdbcTemplate` or `RestTemplate`.

---

11. In RabbitMQ, how do you define a queue in Spring Boot?

A. Using `@EnableRabbit` only
B. Define it as a `@Bean` of type `Queue`
C. Use `application.yml`
D. It's created dynamically at runtime

Answer: B
Explanation: Spring AMQP detects `Queue`, `TopicExchange`, and `Binding` beans to configure infrastructure.
Insight: Declarative config keeps infra code clean.

---

12. What does `@EnableKafka` do?

A. Configures Kafka producers
B. Enables discovery of Kafka listener methods
C. Starts Kafka broker
D. Generates topic schemas

Answer: B
Explanation: Scans for and configures `@KafkaListener`-annotated beans.
Insight: Required when using plain Spring (not Spring Boot) config.

---

13. What is a Kafka "partition"?

A. Separate service
B. Thread of execution
C. Unit of parallelism within a topic
D. Security key

Answer: C
Explanation: Partitions allow topics to be split across brokers for scalability.
Insight: One consumer per partition ensures ordered processing.

---

14. What's the purpose of a consumer group in Kafka?

A. Encrypts messages
B. Balances load among multiple consumers
C. Ensures idempotency
D. Deletes old topics

Answer: B
Explanation: Consumers in a group coordinate to consume different partitions.
Insight: Parallel processing with high availability.

---

15. What does `KafkaListenerContainerFactory` configure?

A. Kafka producer settings
B. Consumer listener container concurrency and ack modes
C. JSON validation
D. Broker logging

Answer: B
Explanation: Defines how Kafka consumers behave (threads, acks, retry).
Insight: Customize for error handling and concurrency tuning.

---

16. What is Spring Cloud Stream used for?

A. UI generation
B. Exposing REST endpoints
C. Simplifying event-driven messaging across brokers
D. Starting embedded Kafka

Answer: C
Explanation: Abstraction layer over Kafka, RabbitMQ, etc., for unified messaging API.
Insight: Reduces boilerplate and vendor lock-in.

---

17. Which annotation is used in Spring Cloud Stream to bind to a Kafka or RabbitMQ consumer?

A. `@MessageConsumer`
B. `@InboundChannelAdapter`
C. `@StreamListener`
D. `@BindingListener`

Answer: C
Explanation: `@StreamListener` listens to input bindings in functional or annotation-based style.
Insight: Combine with `@EnableBinding` or `FunctionalRegistration`.

---

18. What property enables retries in Spring Kafka consumer?

A. `spring.kafka.retry.count`
B. `spring.kafka.consumer.enable-auto-retry=true`
C. `spring.kafka.listener.retry-template`
D. `spring.kafka.consumer.retry.enabled=true`

Answer: C
Explanation: You can use a `RetryTemplate` for retry behavior.
Insight: Prevents dropping messages due to transient failures.

---

19. What happens if a Kafka consumer fails with an exception?

A. Message is lost
B. Consumer stops permanently
C. Message is retried or sent to DLQ if configured
D. Topic is deleted

Answer: C
Explanation: You can configure retries, seek-to-current, or dead-letter topics.
Insight: Always set error handlers for resilience.

---

20. What is the RabbitMQ equivalent of a Kafka topic?

A. Queue
B. Binding
C. Exchange
D. Key

Answer: C
Explanation: Exchanges route messages to queues based on routing keys.
Insight: TopicExchange supports wildcard-based pub-sub patterns.

---

21. Which message pattern is more durable under failure, Kafka or RabbitMQ (default setup)?

A. RabbitMQ
B. Kafka
C. Both
D. Neither

Answer: B
Explanation: Kafka stores messages in a distributed, replicated log.
Insight: Kafka's design favors long-term persistence and reprocessing.

---

22. What is the default acknowledgment mode in Kafka listeners?

A. MANUAL
B. AUTO
C. NONE
D. TIMEOUT

Answer: B
Explanation: AUTO commits offsets after successful processing.
Insight: MANUAL gives better control for critical flows.

---

23. How do you publish a message using Spring AMQP?

A. `AmqpTemplate.convertAndSend(...)`
B. `RabbitPublisher.send(...)`
C. `MessageQueue.send()`
D. `RabbitTemplate.push(...)`

Answer: A
Explanation: `AmqpTemplate` is used to send objects to queues or exchanges.
Insight: Spring auto-converts Java objects to JSON or other formats.

---

24. What kind of exchange is best for topic-based routing in RabbitMQ?

A. DirectExchange
B. FanoutExchange
C. TopicExchange
D. HeaderExchange

Answer: C
Explanation: `TopicExchange` supports wildcard routing patterns like `*.order.*`.
Insight: Helps with pattern-matching message consumers.

---

25. What's the best way to ensure at-least-once delivery in Spring Kafka?

A. Use MANUAL ack mode and commit after processing
B. Disable partitioning
C. Use auto-commit and ignore failures
D. Use Flux instead of Kafka

Answer: A
Explanation: Manual acks let you control when to mark messages as consumed.
Insight: Prevents message loss during retries or partial failures.

25 MCQs — Circuit Breakers, Resilience4j, Modern Patterns & Pitfalls

---

1. What is the purpose of a circuit breaker in microservices?

A. Encrypt messages
B. Prevent cascading failures by cutting off failing services
C. Retry requests indefinitely
D. Auto-scale services

Answer: B
Explanation: Circuit breakers detect failures and prevent further attempts temporarily.
Insight: Helps maintain system stability and fast failure response.

---

2. Which of these is a Java library that implements circuit breaker, retry, and bulkhead patterns?

A. Hystrix
B. Resilience4j
C. Sentinel
D. Ribbon

Answer: B
Explanation: Resilience4j is a modern, lightweight fault tolerance library.
Insight: Built for Java 8+ and integrates easily with Spring Boot.

---

3. What happens when a circuit breaker is in the OPEN state?

A. All requests go through
B. Requests are queued
C. All requests are immediately failed
D. Requests are rerouted

Answer: C
Explanation: Requests are not passed to the downstream service.
Insight: Helps prevent overwhelming unstable services.

---

4. What is the default behavior of a Resilience4j retry?

A. Retry only on null
B. Retry only once
C. Retry on exceptions
D. Retry forever

Answer: C
Explanation: Retries are triggered on exceptions by default.
Insight: Configure max attempts and backoff intervals wisely.

---

5. What is a fallback method in circuit breaker logic?

A. Alternate service for logging
B. A backup path used when the main service fails
C. A retry handler
D. A monitoring tool

Answer: B
Explanation: Fallbacks provide default behavior during service failures.
Insight: Should be light, fast, and safe — never throw exceptions.

---

6. Which annotation enables Resilience4j's circuit breaker in Spring Boot?

A. `@CircuitBreaker`
B. `@Retryable`
C. `@Resilient`
D. `@FailSafe`

Answer: A
Explanation: Use `@CircuitBreaker(name = "serviceName")` to apply resilience logic.
Insight: Combine with `@Recover` or fallback logic for graceful degradation.

---

7. What is the CLOSED state of a circuit breaker?

A. Circuit is broken
B. All requests are rejected
C. All requests pass through normally
D. Retry limit reached

Answer: C
Explanation: This is the healthy state; everything is functioning.
Insight: Failure threshold triggers transition to OPEN.

---

8. What does the HALF_OPEN state mean?

A. All calls go through
B. Retry a limited number of requests to check recovery
C. Logs only metrics
D. All retries are paused

Answer: B
Explanation: A test state after the circuit is open for some time.
Insight: If test calls succeed, circuit closes again.

---

9. What's a good use case for the Bulkhead pattern?

A. API rate limiting
B. Isolating service failure domains by threads or semaphores
C. Encrypting traffic
D. Sharing DB connections

Answer: B
Explanation: Bulkheads restrict concurrent executions to protect shared resources.
Insight: Prevents a flood of calls from overwhelming the system.

---

10. What happens if fallback logic itself throws an exception?

A. It's retried
B. Application may crash or return 500
C. It is ignored
D. Circuit resets

Answer: B
Explanation: Unhandled exceptions in fallbacks result in failed responses.
Insight: Fallbacks must be exception-safe and fast.

---

11. Which of these is an anti-pattern in resilience design?

A. Timeout with fallback
B. Retry with backoff
C. Infinite retries
D. Circuit breaker per external call

Answer: C
Explanation: Infinite retries can amplify failures and lead to service collapse.
Insight: Always use capped retries with jitter and backoff.

---

12. Which annotation is used for retry behavior in Spring Retry or Resilience4j?

A. `@TryAgain`
B. `@Retryable`
C. `@Resend`
D. `@CircuitRetry`

Answer: B
Explanation: `@Retryable` specifies retry attempts, delay, and exceptions to retry on.
Insight: Use sparingly to avoid downstream overload.

---

13. What is the ideal approach when retrying calls to a slow downstream service?

A. Retry immediately
B. Retry with exponential backoff
C. Retry with static delay
D. Use fallback only

Answer: B
Explanation: Exponential backoff prevents retry storms and buys time.
Insight: Combine with jitter/randomness to prevent retry spikes.

---

14. What is "retry storm"?

A. Retry queue overloads
B. All clients retrying together, amplifying load
C. Kafka backlog
D. REST thread exhaustion

Answer: B
Explanation: Clients retrying at the same time overload the service.
Insight: Use jitter, backoff, and circuit breakers to mitigate.

---

15. What is a good default for circuit breaker failure threshold?

A. 1%
B. 25-50%
C. 90%
D. 100%

Answer: B
Explanation: Circuit should trip when a significant failure rate is observed.
Insight: Adjust based on error severity and SLA.

---

16. What does `@TimeLimiter` annotation in Resilience4j do?

A. Forces a method to timeout after specified duration
B. Logs request duration
C. Adds delay between retries
D. Measures memory usage

Answer: A
Explanation: Applies a timeout constraint to method execution.
Insight: Essential for ensuring fast failure in slow paths.

---

17. What is the major benefit of using `CompletableFuture` or `Mono` with resilience patterns?

A. Forces thread blocking
B. Enables better testability
C. Supports non-blocking fallbacks and timeouts
D. Disables retry

Answer: C
Explanation: TimeLimiter works only on async types like `CompletableFuture`.
Insight: Combine non-blocking types with circuit breaker and fallback strategies.

---

18. What is a common misuse of circuit breakers in real-world apps?

A. Using one breaker per downstream service
B. Sharing a single breaker across unrelated endpoints
C. Monitoring breaker metrics
D. Tuning breaker thresholds

Answer: B
Explanation: Shared breakers can cause false positives and shutdown healthy paths.
Insight: Use per-endpoint or per-client granularity.

---

19. What is the role of `RateLimiter` in Resilience4j?

A. Prevent retries
B. Limit concurrent threads
C. Control the number of calls per time window
D. Delay fallbacks

Answer: C
Explanation: Enforces throughput limits on requests per second.
Insight: Useful in public APIs and downstream protection.

---

20. What is a graceful degradation strategy?

A. Throw exception with stack trace
B. Return HTTP 500
C. Provide minimal but valid response or redirect to fallback
D. Retry until server recovers

Answer: C
Explanation: The idea is to return something meaningful instead of failing completely.
Insight: Common in e-commerce (e.g., showing stale prices if pricing service fails).

---

21. Which of the following is a resilience anti-pattern?

A. Distributed tracing
B. Coupling retry logic with business logic
C. Separate fallback class
D. Monitoring circuit states

Answer: B
Explanation: Mixing resilience logic in core business code leads to maintenance issues.
Insight: Always separate concerns via decorators or aspect wrappers.

---

22. Which dashboard allows monitoring of circuit states in Resilience4j?

A. Grafana + Micrometer
B. Spring Admin
C. RabbitMQ Console
D. Eureka UI

Answer: A
Explanation: Resilience4j metrics integrate with Micrometer and can be visualized using Grafana.
Insight: Track success rate, open circuits, and retry counts over time.

---

23. What does the `slidingWindowType` in Resilience4j configure?

A. Number of retries
B. Time to wait after OPEN state
C. Type of window for failure analysis (count/time)
D. Retry thread pool

Answer: C
Explanation: `COUNT_BASED` or `TIME_BASED` window helps in how failures are calculated.
Insight: Pick based on how bursty or consistent your traffic is.

---

24. What pattern helps isolate shared downstream resources by allocating execution pools?

A. Retry
B. Circuit breaker
C. Bulkhead
D. Gateway filter

Answer: C
Explanation: Bulkheads prevent a single service from exhausting all system resources.
Insight: Especially useful in high-traffic or multi-tenant apps.

---

25. What is a “fail-fast” system in microservices?

A. A system that hides errors
B. One that returns default success
C. Immediately rejects requests when issues are detected
D. Blocks requests for long retry periods

Answer: C
Explanation: Fail-fast avoids unnecessary retries and improves overall system throughput.
Insight: Keeps the system responsive under pressure.

25 MCQs — Docker & Kubernetes with Spring Boot

Topic: **K8s Deployment & Containerization Best Practices**

---

1. What is the purpose of a Dockerfile in Spring Boot projects?

A. Define database schema
B. Automate deployment in Jenkins
C. Package the app into a container image
D. Generate YAML configs

Answer: C
Explanation: Dockerfiles define steps to build a container image from your Spring Boot app.
Insight: Use multi-stage builds to reduce final image size.

---

2. What is the typical entrypoint for a Spring Boot Docker image?

A. `CMD ["java", "-jar", "app.jar"]`
B. `ENTRYPOINT ["sh", "boot.sh"]`
C. `RUN mvn spring-boot:run`
D. `RUN java Main`

Answer: A
Explanation: The image should run the Spring Boot app when started.
Insight: Use `exec` form (`[]`) to support signal forwarding in K8s.

---

3. What is a Kubernetes ConfigMap used for?

A. Store binary secrets
B. Mount persistent volumes
C. Store non-sensitive configuration (e.g., YAML, env vars)
D. Store logs

Answer: C
Explanation: ConfigMaps hold plain-text config for injection into containers.
Insight: Avoid putting secrets in ConfigMaps — use `Secret` instead.

---

4. Where should passwords and tokens be stored in Kubernetes?

A. Dockerfile
B. ConfigMap
C. Kubernetes Secret
D. application.yml

Answer: C
Explanation: Secrets are base64 encoded and intended for sensitive data.
Insight: Encrypt at rest or use external tools like HashiCorp Vault.

---

5. Which probe ensures the app is ready to serve traffic?

A. StartupProbe
B. InitProbe
C. ReadinessProbe
D. LivenessProbe

Answer: C
Explanation: Readiness probe gates traffic routing by checking app availability.
Insight: Prevents sending traffic to slow-starting services.

---

6. Which probe restarts the container when the app hangs or becomes unhealthy?

A. StartupProbe
B. TerminationProbe
C. LivenessProbe
D. PingProbe

Answer: C
Explanation: Liveness probe helps recover from runtime issues by restarting the pod.
Insight: Use with caution — false positives can cause restarts during GC.

---

7. Which tool packages Kubernetes YAML along with templates and values?

A. Kustomize
B. Helm
C. Skaffold
D. Tilt

Answer: B
Explanation: Helm simplifies deployment through versioned charts and templated config.
Insight: Supports values injection, lifecycle hooks, and upgrades.

---

8. What's the purpose of `application-k8s.yml` in Spring Boot?

A. For standalone JAR runs
B. Defines Helm values
C. Profile-specific config for Kubernetes deployments
D. CI/CD pipeline config

Answer: C
Explanation: Activated using `--spring.profiles.active=k8s` in container environments.
Insight: Keeps K8s-specific overrides separated from local/dev config.

---

9. What is the use of mounting a volume to `/config` in Spring Boot Docker images?

A. Share logs
B. Inject static files
C. Enable externalized config loading
D. Mount Helm charts

Answer: C
Explanation: Spring Boot can read config from `/config` automatically if mounted.
Insight: Avoid image rebuilds for config changes.

---

10. What is a good base image for Spring Boot Docker builds (production)?

A. `ubuntu:latest`
B. `openjdk:11`
C. `adoptopenjdk:slim` or `eclipse-temurin`
D. `spring:boot`

Answer: C
Explanation: Lightweight images reduce CVE surface area and startup time.
Insight: Consider `distroless` or `jlink`-based builds for security.

---

11. What command generates Kubernetes manifests using Spring Boot & Spring Cloud?

A. `spring k8s generate`
B. `kustomize init`
C. `spring-boot-admin deploy`
D. No built-in generator

Answer: D

Explanation: Spring Boot doesn't generate manifests; use Helm, Kustomize, or manually write them.

Insight: Tools like Jib or Skaffold help build images and deploy faster.

---

12. What is the recommended way to inject secrets into a Spring Boot app running on K8s?

A. Embed in `application.yml`
B. Inject as ENV vars or mounted files from Secret
C. Use inline JSON
D. Pass via command line

Answer: B

Explanation: K8s allows mounting secrets as env vars or files.

Insight: You can use `@Value("${SECRET_ENV_VAR}")` to inject safely.

---

13. What does the `imagePullPolicy: Always` directive do?

A. Rebuilds image on every deploy
B. Pulls latest image every time a pod starts
C. Pulls only if not cached
D. Pulls only on node restart

Answer: B

Explanation: Ensures you're always using the latest image tag.

Insight: Use with care in production — prefer SHA-pinned images.

---

14. What's the purpose of `resources.requests` in a K8s deployment?

A. Define how many replicas to create
B. Specify mandatory CPU/memory per pod
C. Set timeout for container startup
D. Define logging levels

Answer: B

Explanation: Requests reserve CPU/mem to avoid starvation.

Insight: Also used by scheduler to place pods appropriately.

---

15. How do you define a liveness probe in Spring Boot using Actuator?

A. Expose `/health` via actuator
B. Use `spring.liveness-probe.enabled=true`
C. Add a custom servlet
D. Define in controller manually

Answer: A
Explanation: Spring Boot Actuator exposes health endpoints used by K8s probes.
Insight: Customize health groups for readiness vs liveness.

---

16. What is mTLS in Kubernetes context?

A. Multi-Tenant Load Shifting
B. Manual TLS provisioning
C. Mutual TLS for service-to-service authentication
D. TLS termination at gateway

Answer: C
Explanation: mTLS ensures both client and server identities are verified.
Insight: Use with Istio or Linkerd for zero-trust networking.

---

17. Which service mesh is often used to enable mTLS in Kubernetes?

A. Helm
B. Istio
C. FluxCD
D. Kustomize

Answer: B
Explanation: Istio enables service-level encryption and mTLS by default.
Insight: Can enforce policy without code changes.

---

18. What is a Vault in Kubernetes?

A. ConfigMap for passwords
B. Plugin to access Docker Hub
C. External secret manager like HashiCorp Vault
D. Git storage

Answer: C
Explanation: Vaults manage dynamic secrets and secure credentials delivery.
Insight: Avoids hardcoding passwords or API keys in K8s resources.

---

19. How are certificates typically mounted in pods for HTTPS/mTLS?

A. Injected into image
B. Mounted as files from Kubernetes Secrets
C. Added in Dockerfile as ENV
D. Fetched from controller

Answer: B
Explanation: Certificates are often managed as secrets and mounted securely.
Insight: Use sidecar injector if managed by service mesh.

---

20. Which annotation enables Spring Boot to expose Actuator health endpoints?

A. `@EnableHealth`
B. `@EnableMetrics`
C. `@EnableAutoConfiguration`
D. No annotation needed

Answer: D
Explanation: Spring Boot automatically enables `/actuator/health`.
Insight: Use application properties to customize probes.

---

21. What's the function of `terminationGracePeriodSeconds` in pods?

A. Retry delay
B. How long K8s waits before killing a pod after SIGTERM
C. Time to readiness
D. DNS timeout

Answer: B
Explanation: Gives apps time to shut down gracefully.
Insight: Tune this for Spring apps to complete transactions during shutdown.

---

22. Which pattern is not recommended in Docker builds for Spring Boot?

A. Use `ADD` to extract remote archives
B. Use `COPY` with specific JAR path
C. Use multi-stage builds
D. Use Alpine as base image

Answer: A
Explanation: `ADD` downloads remote files — insecure and unpredictable.
Insight: Prefer `COPY` and deterministic builds.

---

23. How can Spring Boot read secrets mounted as files in Kubernetes?

A. Via command-line
B. `@Value("file:/etc/secrets/token")`
C. Use `SecretLoader` class
D. `application.yml` with `{{secret}}` placeholders

Answer: B
Explanation: File-based secrets can be read using Spring's `@Value`.
Insight: Combine with volume mounts and file permission tuning.

---

24. Which is a real risk of using environment variables for secrets?

A. Hard to access
B. Slower startup
C. Visible to all processes in pod
D. Requires mTLS

Answer: C
Explanation: ENV vars are readable by `ps` or `/proc` tools inside container.
Insight: File-based secrets are more secure than env-based.

---

25. What is the safest way to rotate secrets in Kubernetes?

A. Restart the pod manually
B. Use `kubectl edit secret`
C. Integrate with Vault and watch for changes
D. Use application restarts daily

Answer: C
Explanation: Vault agents and secret operators enable dynamic secret rotation.
Insight: Supports short-lived tokens and automated revocation.

25 MCQs — GraalVM & Native Images (Spring Boot 3)

Topic: **Native Ops – Performance & Optimization**

---

1. What is GraalVM?

A. A Spring Boot plugin
B. A JVM-based database
C. A high-performance polyglot VM supporting native image generation
D. A JSON parser

Answer: C
Explanation: GraalVM allows compiling Java applications into native executables.
Insight: Enables fast startup and low memory footprint.

---

2. What is the main benefit of building a native image with GraalVM?

A. Larger JAR files
B. Higher CPU usage

C. Faster startup and lower memory usage
D. Hot reloading

Answer: C
Explanation: Native images remove the need for JVM bytecode interpretation at runtime.
Insight: Perfect for cloud, CLI tools, serverless.

---

3. What does AOT mean in the context of Spring Boot 3?

A. After-on-time initialization
B. Ahead-of-Time compilation
C. Application-on-Test
D. Architecture Optimization Tools

Answer: B
Explanation: AOT compiles application context, beans, and configurations at build-time.
Insight: Reduces reflection and accelerates Spring startup.

---

4. What tool does Spring Boot 3 use internally for AOT processing?

A. Kotlin compiler
B. GraalVM Build Tools Plugin
C. `spring-aot-maven-plugin`
D. ByteBuddy

Answer: C
Explanation: Spring AOT plugin generates optimized code for native image build.
Insight: Replaces much of Spring's dynamic runtime processing.

---

5. What GraalVM feature compiles Java bytecode into a standalone executable?

A. JVM
B. JIT Compiler
C. Native Image
D. HotSpot

Answer: C
Explanation: Native Image transforms Java apps into OS-level executables.
Insight: No JVM needed at runtime.

---

6. Which of these is a limitation of native image builds?

A. Low memory usage
B. Full reflection support by default
C. Fast startup
D. Ahead-of-time GC configuration

Answer: B
Explanation: Reflection is limited — needs configuration in `reflect-config.json`.
Insight: Spring Boot AOT helps generate this automatically.

---

7. What Spring Boot 3 feature improves compatibility with GraalVM native builds?

A. Reactive Streams
B. Functional Beans
C. Spring AOT Engine
D. MVC AutoConfig

Answer: C
Explanation: The AOT engine transforms runtime config into static code.
Insight: Makes apps reflection-free and startup-friendly.

---

8. What command builds a native image using Maven in Spring Boot 3?

A. `mvn spring-boot:native`
B. `mvn native:build`
C. `mvn spring-boot:build-image`
D. `mvn -Pnative native:compile`

Answer: D
Explanation: This uses the native profile enabled via Spring Boot Native support.
Insight: Requires GraalVM and proper plugin setup.

---

9. What is the downside of native builds in CI/CD pipelines?

A. Long container startup
B. High memory usage
C. Long build time
D. Increased runtime dependency

Answer: C
Explanation: Building native images is CPU-intensive and slow compared to JVM builds.
Insight: Use caching and CI-native runners for speed.

---

10. What must be done to support reflection in GraalVM native image?

A. Nothing, it’s automatic
B. Add Spring Devtools
C. Create JSON-based metadata (`reflect-config.json`)
D. Rewrite the code in Kotlin

Answer: C
Explanation: Native image build-time analysis needs reflection metadata.
Insight: Spring AOT generates most of it automatically.

---

11. Which Spring Boot starter is required for native compilation support?

A. `spring-boot-starter-graalvm-native`
B. `spring-native`
C. No starter required in Boot 3
D. `spring-graal`

Answer: C
Explanation: In Spring Boot 3, native support is integrated directly.
Insight: Clean break from older `spring-native` experimental support.

---

12. What flag disables automatic runtime proxies and forces AOT resolution?

A. `--no-proxy-mode`
B. `--aot-only=true`
C. `--native`
D. `--spring.aot.enabled=true`

Answer: D
Explanation: This enables the AOT transformation mode.
Insight: Essential in non-Spring-aware CLI tools.

---

13. What native image limitation affects dynamic configuration loading?

A. Dynamic classpath scanning
B. Database access
C. File I/O
D. Logging

Answer: A
Explanation: Classpath scanning must be statically declared for native images.
Insight: AOT analysis removes all dynamic scanning behavior.

---

14. What's the typical startup time of a native Spring Boot app?

A. 5 seconds
B. 2 seconds
C. 100 ms – 400 ms
D. 20 seconds

Answer: C
Explanation: Native apps start under 500ms — ideal for serverless.
Insight: JVM startup might take 5–10 seconds in comparison.

---

15. What does GraalVM's `--no-fallback` flag do?

A. Avoids native build
B. Disables Java 8 support
C. Prevents fallback to JVM mode if native image build fails
D. Enables JIT

Answer: C
Explanation: Forces native image build to fail if unsupported features are used.
Insight: Best for catching compatibility issues early.

---

16. Where is GraalVM's native executable output by default?

A. `target/classes`
B. `target/native-image/`
C. `/tmp`
D. `~/.m2/`

Answer: B
Explanation: Most plugins output to this directory.
Insight: Use in Docker builds directly for Alpine-compatible distroless images.

---

17. Why is AOT important for Kubernetes and cloud-native apps?

A. Helps deploy to Azure
B. Improves YAML parsing
C. Reduces startup time, resource usage, and cold start delays
D. Adds new monitoring APIs

Answer: C
Explanation: Crucial for autoscaling and serverless models.
Insight: Cheaper and faster scale-out in production.

---

18. What problem does `native-image-agent` solve during development?

A. Automatic Docker image generation
B. Classpath conflict debugging
C. Captures runtime reflection usage for config generation
D. Measures memory

Answer: C
Explanation: This agent generates reflection config during test runs.
Insight: Helps cover edge cases that AOT misses.

---

19. Which JVM features are not supported by native images?

A. Logging
B. JMX, dynamic proxies, or bytecode generation at runtime
C. Scheduled tasks
D. Exception handling

Answer: B
Explanation: These require dynamic runtime resolution, blocked in native image.
Insight: Replace dynamic behavior with AOT equivalents.

---

20. What is Spring Boot's GraalVM alternative to reflection-based DI?

A. Java 21 Records
B. Static factory beans
C. Generated bean registration via AOT processor
D. Java Proxy API

Answer: C
Explanation: Spring AOT builds a static graph of beans to eliminate reflection.
Insight: Enables classpath-free startup.

---

21. What runtime configuration file is used to declare reflection needs for GraalVM?

A. `application.yml`
B. `reflect-config.json`
C. `bootstrap.properties`
D. `aot-settings.gradle`

Answer: B
Explanation: It declares classes and fields requiring reflection.
Insight: Needed when AOT inference fails or for 3rd-party libraries.

---

22. Which is true about GraalVM image sizes?

A. Always larger than JAR
B. Comparable to fat JARs
C. Typically smaller and self-contained
D. Requires Docker

Answer: C
Explanation: Native images are usually 20–60MB and have no JVM dependency.
Insight: Ideal for small cloud services or CLI tools.

---

23. What command-line tool is used to generate native image manually?

A. `java -native`
B. `native-image`
C. `mvn install-native`
D. `javac --aot`

Answer: B
Explanation: `native-image` is the core tool for compiling native executables.
Insight: Use flags like `--no-fallback`, `--static` for full control.

---

24. Why might native apps not work with certain 3rd party libraries?

A. Poor memory use
B. Dynamic code generation or reflection is unsupported without config
C. Too much logging
D. Requires cloud connectivity

Answer: B
Explanation: GraalVM restricts runtime introspection, breaking some libraries.
Insight: Look for "GraalVM compatibility" badges or community support.

---

25. What Gradle plugin is used to build GraalVM native images in Spring Boot 3?

A. `native-gradle`
B. `spring-native-gradle`
C. `org.graalvm.buildtools.native`
D. `native-compiler-plugin`

Answer: C
Explanation: Official Gradle plugin for native builds.
Insight: Works with Spring Boot Gradle plugin to AOT the app first.

25 Advanced MCQs — GraalVM Native Images with Real-World Insights (Set 2)

---

1. What is a major startup improvement seen in Spring Boot 3 native images compared to JVM mode?

A. 10% improvement
B. Near-zero startup with AOT (~100ms)
C. Slight increase due to reflection analysis
D. No difference

Answer: B
Explanation: Spring Boot 3 + AOT + GraalVM yields ~50–200ms startup vs 3–5 seconds JVM.
Insight: Game changer for serverless platforms like AWS Lambda, Cloud Run, Knative.

---

2. What is the most efficient garbage collector available for GraalVM native images?

A. CMS
B. G1
C. Serial GC
D. Static heap management via SubstrateVM

Answer: D
Explanation: Native images use SubstrateVM which has limited GC options — simple and fast.
Insight: Fine-tune heap size statically using `--initial-heap-size` and `--max-heap-size`.

---

3. What is a critical concern when using native images with dynamically loaded plugins or scripts?

A. They increase image size
B. They bypass Spring DI
C. They won't work unless declared ahead of time
D. They require Docker

Answer: C
Explanation: GraalVM eliminates classpath scanning; such behavior must be statically defined.
Insight: Re-architect plugin loading or use `native-image-agent` to detect reflection needs.

---

4. How can one detect which reflection calls happen at runtime in a dev environment?

A. Enable debug logging
B. Use `native-image-agent` during test runs
C. Add `@ReflectiveAccess`
D. Use Prometheus

Answer: B
Explanation: Agent tracks runtime reflection and produces JSON config.
Insight: Use `-agentlib:native-image-agent=config-output-dir=...` in local runs or integration tests.

---

5. What must be avoided inside `@PostConstruct` methods in native images?

A. Logging
B. Reflection-based instantiation
C. Null checks
D. Bean injection

Answer: B
Explanation: These will not work unless configured statically.
Insight: Prefer dependency-injected logic during AOT phase.

---

6. What type of logging configuration works best with native images?

A. log4j XML
B. SLF4J with JSON config
C. Precompiled `logback-spring.xml` with static class bindings
D. JDK Logging

Answer: C
Explanation: Avoid runtime configuration loaders.
Insight: Avoid dynamic log config reloading via filesystem in native images.

---

7. Why do JMX features fail in native images by default?

A. GraalVM drops JVM extensions
B. JMX is blocked due to reflection and dynamic proxy
C. Spring disables it in prod
D. Because of heap size

Answer: B
Explanation: JMX depends heavily on reflection and proxying — not supported by native images.
Insight: Replace with Micrometer/OpenTelemetry for monitoring.

---

8. What is the main reason to use `--initialize-at-build-time` in native builds?

A. Faster runtime memory allocation
B. Reduces file size

C. Forces certain classes to initialize during image creation
D. Enables async behavior

Answer: C
Explanation: Avoids runtime static init issues for incompatible libraries.
Insight: Useful for logging frameworks and JDBC drivers.

---

9. How does the native image approach affect Docker image sizes?

A. Requires large base images
B. Enables ~20–40MB ultra-small images
C. Adds multiple JVM layers
D. Requires Alpine

Answer: B
Explanation: No JVM is required; static binaries run in `scratch` or `distroless`.
Insight: Ideal for minimal, hardened containers.

---

10. What is an advantage of combining GraalVM native image with Docker multi-stage builds?

A. More startup memory
B. Smaller final images with CI caching
C. Adds log rotation
D. Adds test cases

Answer: B
Explanation: Compile in GraalVM container, export only final binary layer.
Insight: Combine with `distroless` or `scratch` base for security.

---

11. What breaks if you use dynamic proxies in Spring Boot with native images?

A. Scheduler
B. WebClient
C. Runtime autowiring
D. AOP or security proxies

Answer: D
Explanation: Proxies based on interfaces must be explicitly declared for native compilation.
Insight: Use compile-time proxies or static wrappers where possible.

---

12. Why is it hard to use Hibernate with GraalVM native images?

A. It’s deprecated
B. Uses bytecode enhancement and runtime reflection

C. Requires Oracle license
D. Needs XML configuration

Answer: B
Explanation: Hibernate scans and proxies entities at runtime.
Insight: Use Spring Data JDBC or query-only JPA models.

---

13. Which Spring Boot feature causes problems in native images?

A. `@Scheduled`
B. Devtools reload
C. `@RestController`
D. `@Slf4j`

Answer: B
Explanation: Devtools uses classloader tricks that are incompatible with AOT.
Insight: Remove Devtools in native builds.

---

14. What happens when a native image attempts to access a class not declared for reflection?

A. NullPointerException
B. ClassNotFoundException
C. Segfault
D. `UnsupportedFeatureException`

Answer: D
Explanation: GraalVM throws this when reflection targets are not configured.
Insight: Fix by adding entry to `reflect-config.json`.

---

15. What's the impact of `@ConfigurationProperties` in native images?

A. They are disabled
B. Work only if bound statically
C. Require profile activation
D. Enable YAML generation

Answer: B
Explanation: Need to be bound via AOT.
Insight: Use Spring Boot 3 AOT config binder to make them native-friendly.

---

16. Which metrics/telemetry tool is best suited for native image observability?

A. JMX
B. Micrometer + Prometheus

C. JDBC
D. Java Flight Recorder

Answer: B
Explanation: Lightweight, reflection-free metrics integration.
Insight: Use `micrometer-core` and `micrometer-registry-prometheus`.

---

17. What is the difference between JVM heap and native heap in this context?

A. JVM uses static memory
B. Native heap must be pre-sized
C. Native heap dynamically grows
D. JVM heap is smaller

Answer: B
Explanation: Native apps need heap size hints at build or runtime flags.
Insight: Use flags like `--initial-heap-size=64m`.

---

18. How is native image performance at runtime compared to JVM?

A. Always slower
B. Faster GC
C. Comparable or slightly slower in long-lived apps
D. Crashes under load

Answer: C
Explanation: No JIT → less optimized long-running throughput.
Insight: Great for short-lived apps, cold-start critical systems.

---

19. Why is `graalvm-ce-java17` preferred over `graalvm-ce-java11` for Spring Boot 3?

A. Supports YAML
B. Java 17 is minimum for Boot 3 native support
C. Better IDE support
D. Smaller image

Answer: B
Explanation: Spring Boot 3 requires Java 17 baseline.
Insight: Ensure version alignment in Maven/Gradle and Docker base.

---

20. Which file typically includes native-image specific configuration?

A. `.graalvmrc`
B. `application.properties`

C. `reflect-config.json`, `resource-config.json`
D. `pom.xml` only

Answer: C
Explanation: Used by GraalVM to declare required reflection and resources.
Insight: Tools like `native-image-agent` can generate them.

---

21. How does native compilation affect performance monitoring tools like New Relic or Dynatrace?

A. Works automatically
B. Native image disables Java agent support
C. Improves agent integration
D. Requires Micrometer

Answer: B
Explanation: Java agents cannot be attached to native images.
Insight: Use OpenTelemetry SDK directly in native code.

---

22. When building native image Docker container, what Linux base should you prefer?

A. Ubuntu
B. Alpine or `scratch`
C. CentOS
D. BusyBox

Answer: B
Explanation: Minimal base → smaller attack surface, faster deploys.
Insight: Combine with statically linked Graal binary.

---

23. What does `--static` flag do in native-image builds?

A. Uses static IP
B. Disables proxies
C. Links the binary statically, removing glibc dependencies
D. Adds Docker layer

Answer: C
Explanation: Ideal for portable containers using `scratch`.
Insight: Increases portability and hardens deployment.

---

24. Which deployment scenario benefits least from native compilation?

A. Serverless functions
B. Short-lived CLI tools
C. Always-on web servers
D. Cold-start APIs

Answer: C
Explanation: Throughput may be better on JVM for long-lived processes.
Insight: JVM still wins for warm, memory-rich, heavy-lifting workloads.

---

25. What is a common CI pipeline failure with native images?

A. Maven tests fail
B. Timeout due to long native compilation time
C. Kubernetes crash
D. Docker daemon dies

Answer: B
Explanation: Native image builds can take 5–10+ mins and may timeout CI runners.
Insight: Use caching, tuned runners, and split phases for performance.

25 MCQs — 12-Factor App with Spring Boot

---

1. What is the main goal of the 12-Factor App methodology?

A. Faster frontend rendering
B. Scalability and maintainability of modern web apps
C. Optimize JVM runtime
D. Enable desktop deployments

Answer: B
Explanation: 12-factor apps aim to be cloud-native, portable, and scalable across environments.
Insight: Ideal for PaaS environments like Kubernetes, Cloud Foundry, AWS ECS.

---

2. Which Spring feature supports Factor I: "Codebase – One codebase tracked in revision control"?

A. Spring Boot Devtools
B. Spring Initializr
C. Monorepo + Git SCM with branch-based deployment
D. Multiple WAR files

Answer: C
Explanation: One codebase per app; use Git-based versioning.
Insight: Use branching strategy (e.g., GitFlow) to manage releases per environment.

---

3. For Factor II: "Dependencies", how does Spring Boot encourage explicit dependency declaration?

A. Uses Maven/Gradle to define external libraries
B. Automatically bundles all libraries
C. Reads from lib directory
D. No dependency management

Answer: A
Explanation: Spring Boot manages dependencies via Maven POM or Gradle `build.gradle`.
Insight: Use dependency locking to pin versions in CI.

---

4. Factor III (Config): How should you externalize configuration in Spring Boot?

A. Hardcode in source
B. Use `@Value` and `application.properties` with profiles
C. Use `.env` only
D. Inline values in code

Answer: B
Explanation: Spring supports profile-based configs and environment variable overrides.
Insight: Use ConfigMaps/Secrets in K8s, Spring Cloud Config in distributed apps.

---

5. Factor IV (Backing services): How does Spring Boot treat services like DB, queues, caches?

A. As plugins
B. As hardcoded beans
C. As replaceable resources identified via config
D. As part of the app bundle

Answer: C
Explanation: Database, cache, and messaging services should be configured via properties.
Insight: Use env-based service discovery (e.g., Eureka, Consul).

---

6. What is a best practice for handling secrets in 12-factor Spring apps?

A. Bundle in WAR file
B. Add to `application.yml`
C. Load from environment variables or external secrets
D. Use inline literals in controllers

Answer: C
Explanation: Use env vars, Spring Cloud Vault, or K8s Secrets.
Insight: Never commit secrets to Git — even in `.gitignore`ed files.

---

7. Which of the following violates the 12-factor "Logs" principle in Spring apps?

A. Logging to stdout/stderr
B. Using a centralized log aggregator
C. Writing logs to `/var/logs/myapp.log` inside container
D. Streaming logs in JSON

Answer: C
Explanation: Apps should not manage or rotate logs — logs must stream.
Insight: Use Micrometer + Loki/Fluentd or ELK.

---

8. What does Spring Boot's actuator support that aligns with Factor IX (Disposability)?

A. Manual shutdown of JAR
B. Fast startup + graceful shutdown
C. UI-based restart
D. JVM monitoring

Answer: B
Explanation: Actuator exposes `/shutdown` and integrates clean shutdown hooks.
Insight: Crucial for autoscaling in Kubernetes.

---

9. For Factor V (Build, Release, Run), which Spring concept helps separation of build and run?

A. `spring.profiles.active`
B. `@PostConstruct`
C. Docker multi-stage builds
D. JAR-in-JAR pattern

Answer: A
Explanation: Separate runtime behavior using profiles.
Insight: Do not bundle environment-specific config into builds.

---

10. What enables Factor VI (Processes) in Spring Boot?

A. Use of `@Async` and non-blocking threads
B. Stateless behavior across requests
C. Session-based persistence
D. Shared caches across services

Answer: B
Explanation: Each request should be handled in isolation.
Insight: Stateless apps allow horizontal scaling without sticky sessions.

---

11. What Spring feature violates stateless process principle if misused?

A. `@Component`
B. `@SessionScope`
C. `@ConfigurationProperties`
D. `@Value`

Answer: B
Explanation: `@SessionScope` ties state to user session, breaking horizontal scalability.
Insight: Prefer token-based stateless auth (e.g., JWT).

---

12. What Spring Cloud component implements centralized config store for Factor III?

A. Spring Data
B. Spring Cloud Config Server
C. Eureka
D. Zipkin

Answer: B
Explanation: Config Server reads from Git or Vault and serves config via REST.
Insight: Allows live reloads with Spring Cloud Bus (RabbitMQ/Kafka).

---

13. How does Spring Boot encourage short-lived, disposable process execution (Factor IX)?

A. Long-lived HTTP connections
B. Resilient shutdown lifecycle and container health checks
C. JVM-only startup
D. Blocking main thread

Answer: B
Explanation: Graceful shutdowns via `@PreDestroy`, actuator probes.
Insight: Works well with `liveness` and `readiness` probes in Kubernetes.

---

14. Which pattern helps achieve horizontal scaling in Spring microservices?

A. Stateful session replication
B. Stateless REST APIs + database-backed persistence
C. ActiveMQ failover
D. WebSocket sticky sessions

Answer: B
Explanation: Statelessness enables scale-out across containers.
Insight: Store session state in Redis or DB if needed.

---

15. How can Spring Boot apps comply with Factor VIII (Concurrency)?

A. Thread pools
B. Immutable configuration
C. Run multiple app instances behind a load balancer
D. Shared mutable state

Answer: C
Explanation: Scale horizontally by running multiple copies of the app.
Insight: Use Kubernetes Deployments + autoscaling.

---

16. Which Spring pattern enables on-demand administrative tasks (Factor X: Admin Processes)?

A. `@Scheduled` jobs
B. Running CLI runners using `CommandLineRunner` or `ApplicationRunner`
C. Health checks
D. `@EventListener`

Answer: B
Explanation: Use `CommandLineRunner` to execute admin or batch operations at runtime.
Insight: Trigger via `kubectl exec`, SSH, or custom endpoint.

---

17. What violates Factor VII (Port binding) in Spring apps?

A. Embedded Tomcat via Spring Boot
B. Running app behind nginx
C. Expecting external web server to run Spring WAR
D. Running as `java -jar`

Answer: C
Explanation: Apps should be self-contained and expose HTTP via embedded servers.
Insight: Spring Boot's embedded Netty/Tomcat/Jetty supports this.

---

18. What helps maintain strict separation between build and run stages in Spring apps?

A. Different Maven profiles
B. Runtime environment overrides like `SPRING_DATASOURCE_URL`
C. Shared config files
D. Devtools hot reload

Answer: B
Explanation: Runtime config from ENV overrides build-time config.
Insight: Prevents rebuilds for new environment setups.

---

19. What tool helps simulate Factor XI (Logs) in local Spring Boot development?

A. Loki
B. FileAppender
C. Console logging to stdout with `docker logs`
D. JSON file streaming

Answer: C
Explanation: Standard output is captured by Docker and Kubernetes log systems.
Insight: FluentBit, ELK, or Loki scrape these logs in production.

---

20. What tool in Spring ecosystem helps autoscale based on metrics?

A. Spring Cloud Gateway
B. Prometheus
C. Kubernetes HPA + Micrometer metrics
D. Spring CLI

Answer: C
Explanation: Micrometer exposes metrics, K8s HPA scales based on thresholds.
Insight: Use metrics like CPU, memory, request rate.

---

21. Which principle suggests avoiding sticky sessions or in-memory sessions in Spring Boot?

A. Factor VI – Stateless Processes
B. Factor IX – Disposability

C. Factor III – Config
D. Factor I – Codebase

Answer: A
Explanation: Stateless design improves scale-out and resilience.
Insight: Use Redis-based session storage if persistence is needed.

---

22. Which property supports dynamic port assignment in Spring Boot (Factor VII)?

A. `server.port=8080`
B. `server.port=0`
C. `port=dynamic`
D. `spring.port.assign`

Answer: B
Explanation: Spring will bind to a random port, useful for tests or multi-instance runs.
Insight: Use actuator or logs to discover assigned port.

---

23. What is a real-world example of breaking Factor V: Build, Release, Run?

A. Same artifact used across all environments
B. Changing config during deployment
C. Using CI/CD to inject credentials
D. Committing passwords to Git

Answer: B
Explanation: Release artifacts should not be modified after build.
Insight: Separation ensures repeatable, safe deploys.

---

24. Which Spring module helps achieve Factor XII (Dev/prod parity)?

A. Spring Profiles + Docker + Flyway
B. Spring MVC
C. Spring Kafka
D. Spring Mail

Answer: A
Explanation: Profiles help simulate environments, Flyway applies migrations, Docker ensures parity.
Insight: Use same container image for dev/test/prod with different external config.

---

25. What is the correct flow of stages in a 12-factor compliant delivery pipeline?

A. Develop → Test → Build → Release → Run
B. Build → Release → Run
C. Release → Run → Build
D. Run → Test → Build

Answer: B
Explanation: 12-Factor advocates strict separation: Build → Release → Run
Insight: CI builds artifact → CD injects config → runs container.

25 Advanced MCQs — 12-Factor App with Spring Boot (Set 2)

---

1. What is the risk of committing `.env` files or `application-prod.yml` into version control?

A. Slower app boot
B. Enables live config change
C. Leaks sensitive credentials and violates config externalization
D. Breaks REST endpoints

Answer: C
Explanation: Configs must be injected externally, not stored in the repo.
Insight: Use `.gitignore` and secure config delivery like Vault or ENV.

---

2. What does the 12-Factor "Dev/Prod Parity" aim to minimize?

A. CI pipeline duration
B. Log sizes
C. Gaps between local and production environments
D. Code duplication

Answer: C
Explanation: Reduces surprises by keeping parity across dev/staging/prod.
Insight: Use Docker Compose for local, K8s manifests for dev/staging/prod.

---

3. What’s a violation of the “Backing Services as Attached Resources” rule?

A. External Redis accessed via URL
B. DB credentials injected at runtime

C. Writing to a local embedded DB in production
D. Using JDBC

Answer: C
Explanation: Backing services must be external and swappable via config.
Insight: Enables easy failover and decoupling of infra.

---

4. How should Spring Boot apps expose functionality according to the "Port Binding" principle?

A. Through Apache or Nginx proxies
B. Use embedded HTTP server (Tomcat/Jetty/Netty)
C. Use servlet container only
D. Through OS-level sockets

Answer: B
Explanation: Apps must self-contain and expose services directly.
Insight: Use `server.port` property for control.

---

5. In Kubernetes, how should you provide Spring Boot app configuration (Factor III)?

A. Use `application.properties` inside the JAR
B. With Docker ARG
C. Through ConfigMaps and Secrets
D. Hardcoded in Helm chart

Answer: C
Explanation: External config injection aligns with 12-Factor practices.
Insight: Use volume mounts or ENV injection.

---

6. What role do profiles like `dev`, `prod`, and `test` play in Spring Boot?

A. Classpath restriction
B. Static proxy creation
C. Runtime environment selection
D. Database rollback

Answer: C
Explanation: Profiles help isolate and inject environment-specific behavior.
Insight: Combine with `SPRING_PROFILES_ACTIVE`.

---

7. What's the best practice for health checks in 12-Factor Spring Boot apps?

A. Rely on JMX
B. Use `/actuator/health` endpoint
C. Poll the root `/` endpoint
D. Use log tailing

Answer: B
Explanation: Exposes readiness and liveness endpoints.
Insight: Kubernetes probes should point to this.

---

8. In a CI/CD pipeline, when should you inject secrets for production Spring apps?

A. At commit time
B. At build time
C. At runtime only
D. At unit test step

Answer: C
Explanation: Keeps build artifacts immutable and promotes separation.
Insight: Use vaults, secret stores, or Kubernetes secrets injection.

---

9. What does Factor IX: "Disposability" encourage in app design?

A. Heavy statefulness
B. Long startup time
C. Graceful shutdowns and quick boot
D. Dynamic class loading

Answer: C
Explanation: Improves resilience and autoscaling efficiency.
Insight: Use `@PreDestroy`, actuator `/shutdown`, and lifecycle hooks.

---

10. Which Spring Boot feature supports clean "Admin Process" execution (Factor X)?

A. `ApplicationRunner`
B. REST controller with shutdown logic
C. Background thread pool
D. Startup listeners

Answer: A
Explanation: Run commands independently (e.g., DB migrations, cleanup).
Insight: Expose tools via CLI flags or shell entrypoints.

---

11. What is an anti-pattern in config management for Spring Boot apps?

A. Using `@ConfigurationProperties`
B. Encrypting secrets
C. Injecting with `@Value`
D. Duplicating config values across environments

Answer: D
Explanation: Duplicated config creates drift and management issues.
Insight: Use shared config layers + environment overrides.

---

12. What 12-Factor principle does Spring Boot's executable JAR support?

A. Port Binding
B. Dev/Prod Parity
C. Logs
D. Stateless Process

Answer: A
Explanation: Enables apps to run as services on a port.
Insight: Also simplifies containerization.

---

13. What makes a Spring Boot app unsuitable for horizontal scaling?

A. Externalized configuration
B. Local filesystem-based session store
C. Micrometer metrics
D. Embedded Tomcat

Answer: B
Explanation: Shared filesystem/session storage violates statelessness.
Insight: Use Redis or token-based authentication.

---

14. What is the best log format for 12-factor Spring apps in cloud-native environments?

A. HTML logs
B. Text-only multi-line logs
C. JSON logs
D. Binary logs

Answer: C
Explanation: JSON logs are structured, machine-readable, and cloud-native.
Insight: Enables ingestion by ELK/Fluentd/CloudWatch.

---

15. What's the best way to change configuration in a live Spring Boot app deployed on K8s?

A. Redeploy with a new JAR
B. Update `application.yml` inside the image
C. Patch ConfigMap and restart pod
D. Edit properties in `/tmp`

Answer: C
Explanation: External config sources can be updated and rolled out safely.
Insight: Pair with `/actuator/refresh` for dynamic reload.

---

16. What principle does container image immutability reflect?

A. Logs
B. Build/Release/Run separation
C. Devtools
D. Scalability

Answer: B
Explanation: Immutable images enforce separation of concerns.
Insight: Every deployment should use the same artifact.

---

17. Why is session-based authentication discouraged in 12-factor cloud-native Spring apps?

A. Increases image size
B. Prevents log streaming
C. Causes state persistence and violates statelessness
D. Uses too much RAM

Answer: C
Explanation: Session stickiness blocks scalability and autoscaling.
Insight: Prefer token-based approaches like JWT.

---

18. In Spring Boot, how do you conditionally load a bean for a profile?

A. `@ActiveProfile`
B. `@Profile("prod")`
C. `@PropertySource`
D. `@Conditional`

Answer: B
Explanation: `@Profile` lets Spring selectively load beans.
Insight: Keeps behavior clean per environment.

---

19. What causes vendor lock-in and violates portability in 12-factor apps?

A. Using JPA
B. Tightly coupling to cloud-specific APIs
C. Docker base images
D. Logback JSON layout

Answer: B
Explanation: Use abstraction layers like Spring Cloud or Open Service Broker APIs.
Insight: Enables cloud migration.

---

20. What property ensures a Spring Boot app fails fast if config is invalid?

A. `spring.config.fail-fast=true`
B. `spring.profiles.fail=true`
C. `failOnStartup=true`
D. `config.load.mode=STRICT`

Answer: A
Explanation: Prevents startup with misconfigured config sources.
Insight: Important for CI/CD reliability.

---

21. What's a consequence of mixing multiple environment configs in the same `.yml` file?

A. Smaller image size
B. Memory leak
C. Confusion, misrouting, and drift
D. Faster boot

Answer: C
Explanation: Multiple env configs in a shared file can cause accidental deployment mismatch.
Insight: Use clear profile blocks or split files.

---

22. What Spring Boot command-line option is useful for override on the fly (Factor III)?

A. `--Dfile.encoding`
B. `--spring.config.import`
C. `--spring.datasource.url=...`
D. `--enable-refresh`

Answer: C
Explanation: Allows property overrides at runtime.
Insight: Useful in container entrypoint or Docker run args.

---

23. What is the danger of using a shared temp directory (`/tmp/`) in 12-factor apps?

A. Disk space overuse
B. Stateful behavior
C. Lock contention
D. All of the above

> **Answer**: D
> **Explanation**: Shared disk violates statelessness and can cause cross-instance issues.
> **Insight**: Use object stores (e.g., S3, MinIO) or in-memory temp buffers.

---

24. What does the principle “Logs are event streams” imply?

A. Logs must be binary
B. App must expose JSON metrics
C. Logs must stream to stdout for centralization
D. Logs should be discarded after use

> **Answer**: C
> **Explanation**: Streaming logs allow aggregation and analysis via external tools.
> **Insight**: Avoid log rotation inside the app.

---

25. In a secure Spring Boot 12-factor system, how should secrets be managed?

A. Commit them encrypted in source code
B. Store in cloud-native secret vaults (Vault, AWS Secrets Manager)
C. Inline in `application.yml`
D. Use `System.getEnv()` in code

> **Answer**: B
> **Explanation**: External secret stores provide encryption, auditability, and rotation.
> **Insight**: Integrate with Spring Cloud Vault or Kubernetes Secrets.

25 MCQs — Design Patterns & Anti-Patterns in Distributed Spring Apps

---

1. What is the Service Discovery Pattern in microservices?

A. Hardcoding endpoints
B. Registering and locating services dynamically
C. Using DNS round robin
D. Load-balancing with Nginx

Answer: B

Explanation: Services register with discovery tools like Eureka, allowing others to discover them dynamically.

Insight: This eliminates static configs and supports dynamic scaling.

---

2. Which Spring tool supports Service Discovery?

A. Spring JDBC
B. Spring Cloud Eureka
C. Spring Boot Starter Web
D. Spring Security

Answer: B
Explanation: Eureka allows clients to register and discover other microservices.
Insight: Integrates with `@EnableDiscoveryClient`.

---

3. Which of the following is a valid use of Circuit Breaker Pattern?

A. Prevent retrying broken service continuously
B. Optimize serialization
C. Block non-authenticated users
D. Reduce memory

Answer: A
Explanation: Circuit breakers prevent cascading failures when downstream services are down.
Insight: Use Resilience4j in Spring Cloud for declarative circuit breaking.

---

4. What is the API Gateway Pattern commonly used for in Spring Cloud?

A. Inter-service communication
B. DB caching
C. Centralizing access control and routing
D. Storing session state

Answer: C
Explanation: A gateway acts as an entry point for client requests, routing and filtering them.
Insight: Use Spring Cloud Gateway or Netflix Zuul.

---

5. What is an anti-pattern in distributed authentication?

A. Stateless JWT
B. Central OAuth2 identity provider
C. Session-based authentication with sticky load balancer
D. mTLS between microservices

Answer: C
Explanation: Sticky sessions violate statelessness, hinder autoscaling, and break failover.
Insight: Prefer token-based or federated auth.

---

6. What is the purpose of the Strangler Fig Pattern?

A. Kill the app on error
B. Replace old monolith piece-by-piece
C. Centralize message queues
D. Avoid database joins

Answer: B
Explanation: The Strangler Fig pattern allows gradual migration to microservices.
Insight: Use Spring Cloud Gateway to route legacy and new paths.

---

7. What is the Database per Service Pattern?

A. Every service shares one DB
B. Services use private schemas
C. Each service owns its DB and schema
D. Use a global DB with a single schema

Answer: C
Explanation: Isolates service data boundaries and supports autonomy.
Insight: Use events or APIs for cross-service queries.

---

8. What is a common anti-pattern with inter-service communication?

A. Using REST with load-balanced URLs
B. Direct synchronous HTTP calls forming dependency chains
C. Using Feign client for gateway
D. Using circuit breakers

Answer: B
Explanation: Deep synchronous chains lead to cascading failures and high latency.
Insight: Consider async messaging (Kafka) or CQRS projection.

---

9. What is CQRS Pattern in Spring context?

A. Command and Query in same controller
B. One DB per controller
C. Separate read and write models
D. Spring Security roles

Answer: C
Explanation: CQRS improves performance and scalability by splitting read/write paths.
Insight: Read from cache/DB; write via commands or events.

---

10. Which tool in Spring supports event-based design to avoid tight coupling?

A. Spring Retry
B. Spring Events (`ApplicationEventPublisher`)
C. Spring MVC
D. Spring Batch

Answer: B
Explanation: Spring Events support decoupled event listeners.
Insight: Combine with domain events or Kafka for cross-boundary triggers.

---

11. What is a data consistency anti-pattern in distributed Spring apps?

A. Distributed locking
B. Eventual consistency
C. Coordinating commits across services in one transaction
D. Using outbox pattern

Answer: C
Explanation: Distributed 2PC (two-phase commit) is fragile, complex, and often fails at scale.
Insight: Use Outbox pattern + Kafka for reliable propagation.

---

12. What pattern helps version APIs safely in Spring apps?

A. Class versioning
B. API Gateway Transformation
C. Versioning via URL path or headers
D. Spring Data Auditing

Answer: C
Explanation: `/v1/users`, `/v2/users`, or header-based versioning helps maintain backward compatibility.
Insight: Avoid removing old APIs suddenly.

---

13. Which is a valid resilience pattern in Spring Cloud?

A. Caching with TTL
B. Service Mesh
C. Retry with exponential backoff
D. Direct service binding

Answer: C
Explanation: Retry logic protects from transient errors and improves reliability.
Insight: Use `@Retryable` or Resilience4j Retry.

---

14. What is a design pattern for managing config across microservices?

A. Shared database
B. Spring Cloud Config
C. Runtime annotations
D. Internal YAML registry

Answer: B
Explanation: Centralized config allows profile-based environment segregation.
Insight: Works with Git, Vault, or database as config sources.

**15. What pattern should you use for propagating request context (like traceId) across services?

A. MDC + Zipkin
B. ThreadLocal + Logs
C. Redis cache
D. App context injection

Answer: A
Explanation: Use Mapped Diagnostic Context (MDC) to add contextual trace info in logs.
Insight: Spring Cloud Sleuth auto-injects it.

---

16. Which pattern allows separation of cross-cutting concerns like logging, retry, security?

A. Reactive pattern
B. Filter Chain
C. Proxy Pattern via AOP
D. Inheritance tree

Answer: C
Explanation: AOP lets you wrap logic around core concerns.
Insight: Use `@Aspect`, `@Around`, or `HandlerInterceptor`.

---

17. What is an anti-pattern when using Spring Boot with Docker in microservices?

A. Externalizing ports
B. Baking ENV in Docker image

C. Using entrypoint with `java -jar`
D. Minimal base image

Answer: B
Explanation: Baking ENV into the image violates portability and 12-factor config.
Insight: Inject ENV at container runtime (`docker run -e`, K8s secrets).

---

18. What pattern helps to defer feature releases in production safely?

A. Canary deployment
B. Retry pattern
C. Feature toggle
D. Sticky session

Answer: C
Explanation: Feature flags allow toggling features without redeployment.
Insight: Integrate with LaunchDarkly or Togglz.

---

19. Which is a common anti-pattern when exposing internal service APIs?

A. Use Feign
B. Register in Eureka
C. Expose internal endpoints to public Internet
D. Add JWT token

Answer: C
Explanation: Internal APIs should be protected via gateway or service mesh.
Insight: Restrict via firewall/VPC/private ingress.

---

20. What pattern supports observability in distributed apps?

A. Logging only
B. Centralized database
C. Distributed tracing with correlation IDs
D. Dynamic classpath scanning

Answer: C
Explanation: Enables tracking request flow across services.
Insight: Use Sleuth + Zipkin + Micrometer.

---

21. What is a valid fault isolation strategy in Spring Cloud apps?

A. Shared memory across pods
B. Single DB for all services

C. Running each service in its own container with circuit breakers
D. Shared cache bus

Answer: C
Explanation: Fault isolation ensures one service failure doesn't affect others.
Insight: Include `@CircuitBreaker`, retries, and timeouts.

---

22. What anti-pattern may occur when tightly coupling services with shared DTOs?

A. Faster response
B. Unwanted change propagation
C. Better memory usage
D. Faster build

Answer: B
Explanation: Sharing DTOs leads to brittle dependencies.
Insight: Define separate DTOs per service boundary.

---

23. Which pattern supports event-driven microservices using Spring Boot?

A. Feign client
B. Kafka + Spring Cloud Stream
C. In-memory queues
D. Thread join

Answer: B
Explanation: Spring Cloud Stream enables event-based communication via Kafka/RabbitMQ.
Insight: Promotes decoupled and scalable design.

---

24. What is a common pitfall in using async messaging systems like Kafka?

A. Latency
B. Ordering guarantees always hold
C. Failing to handle idempotency
D. Lack of config

Answer: C
Explanation: Async processing may lead to message duplication.
Insight: Ensure idempotent consumers.

---

25. What is the pattern to implement graceful degradation of a failing service?

A. Retry
B. Circuit Breaker

C. Fallback logic
D. Exponential backoff only

Answer: C
Explanation: Fallback responses ensure degraded but functional behavior.
Insight: Combine `@CircuitBreaker` with `fallbackMethod`.

25 MCQs — Advanced Design Patterns & Anti-Patterns in Distributed Spring Apps

---

1. Why is the “Shared Database per Microservice” pattern considered an anti-pattern?

A. It improves performance
B. It violates service boundaries and leads to tight coupling
C. It enhances data consistency
D. It simplifies failover

Answer: B
Explanation: Sharing a DB across services tightly couples them, introducing hidden dependencies and schema entanglement.
Insight: A change in one service's schema may unknowingly break another. Opt for database per service and publish events to share state.

---

2. Which Spring feature is essential for implementing the Bulkhead Pattern?

A. `@RequestMapping`
B. `@Scheduled`
C. ThreadPoolBulkhead from Resilience4j
D. `@Transactional`

Answer: C
Explanation: Bulkhead isolates failures by limiting concurrent access (like separate thread pools per service).
Insight: Prevents one slow dependency from exhausting all threads. Useful in high-latency scenarios like remote APIs.

---

3. What is the Saga Pattern primarily used for?

A. Event replay
B. Database connection pooling

C. Managing distributed transactions with compensating actions
D. Authentication fallback

Answer: C
Explanation: Saga sequences local transactions with rollback actions in case of failure.
Insight: Ideal for e-commerce checkout flows (e.g., order → payment → shipping).

---

4. Which Spring component allows implementing Compensating Transactions for Sagas?

A. `@Rollback`
B. `@Transactional(propagation = NEVER)`
C. Spring State Machine or event listeners
D. Feign Retry

Answer: C
Explanation: Compensation logic is often event-driven or workflow-based. Spring State Machine or Spring Events are flexible for chaining operations.
Insight: Design “undo” actions alongside each step in your service contract.

---

5. What is the Ambassador Pattern used for in microservices?

A. Delegating API calls
B. Managing remote service connection logic in a helper or sidecar
C. Promoting microservices in APIs
D. Scaling services linearly

Answer: B
Explanation: The Ambassador handles cross-cutting concerns like retries, logging, auth — often in a sidecar or proxy.
Insight: Useful with mTLS, observability agents, and retries outside core business logic.

---

6. Which is a correct application of the Sidecar Pattern in Spring?

A. Include logic directly in controller
B. Use actuator inside the main app
C. Deploy logging or monitoring logic as a separate container
D. Bundle configs into the WAR file

Answer: C
Explanation: A sidecar runs alongside the service to offload certain functions like log collection or metrics reporting.
Insight: Often used with tools like Envoy, Fluent Bit, or Prometheus exporters.

---

7. What is the Entity Aggregation Anti-pattern in distributed systems?

A. Combining multiple REST calls
B. Using too many microservices
C. Loading multiple bounded contexts in one aggregate root
D. Aggregating DTOs for frontend

Answer: C
Explanation: Aggregating unrelated entities violates DDD boundaries and hurts autonomy.
Insight: Leads to complex service contracts and coordination problems. Keep aggregates focused and cohesive.

---

8. What is the Event Carried State Transfer pattern?

A. Events carry commands
B. Events carry enough data to rebuild state in consumers
C. Events trigger updates without data
D. No need to persist events

Answer: B
Explanation: Events contain the full state needed for downstream services to act independently.
Insight: Enables loosely coupled projections and read models.

---

9. Which Spring concept supports event sourcing principles best?

A. Spring Kafka or Spring Cloud Stream
B. Spring Security
C. Spring Batch
D. Spring AOP

Answer: A
Explanation: Kafka enables append-only log-style data flow and replay for state rebuilding.
Insight: Can be used for rebuilding projections and supporting time travel in domain models.

---

10. What is the fat controller anti-pattern, and how can Spring help mitigate it?

A. Controller contains service logic → use `@Service` layer to extract business logic
B. Controller handles only routing
C. Controller contains only one method
D. Controller is in a separate package

Answer: A
Explanation: Avoid embedding logic in controllers; move to service/handler layer.
Insight: Promotes testability, single responsibility, and better layering.

---

11. Why is using `ThreadLocal` for request-scoped data in a reactive Spring WebFlux app an anti-pattern?

A. It improves performance
B. It blocks I/O
C. Reactive streams use non-blocking event loop, breaking ThreadLocal guarantees
D. It uses JMX

Answer: C
Explanation: Threads in reactive pipelines are not dedicated per request.
Insight: Use `Context` in Reactor or WebClient filters.

---

12. What design pattern helps avoid duplicate Kafka message processing?

A. Circuit Breaker
B. Bulkhead
C. Idempotency Token
D. Feign Retry

Answer: C
Explanation: Idempotent operations ensure safe reprocessing of messages.
Insight: Store processed message IDs or apply business-level idempotency keys.

---

13. What anti-pattern occurs when microservices share the same entity classes?

A. DTO leak
B. Broken encapsulation and brittle dependencies
C. Performance boost
D. Logging overload

Answer: B
Explanation: Changes in one domain ripple through all services.
Insight: Always define contract-specific DTOs per service.

---

14. Why is tight coupling to a messaging format like Avro/Protobuf across services dangerous?

A. Hard to scale
B. High performance
C. Contract changes ripple across consumers
D. No observability

Answer: C
Explanation: Changing one field may break several consumers.
Insight: Use schema registry and versioning.

---

15. What is the benefit of applying the Outbox Pattern with Spring Data and Kafka?

A. Logging
B. Avoiding distributed transaction by publishing events from the same DB commit
C. Skipping validation
D. Reducing microservices

Answer: B
Explanation: Save event and DB update in the same transaction; publish later asynchronously.
Insight: Guarantees consistency without 2PC.

---

16. How can tracing headers be propagated across services in Spring?

A. Manually add UUID
B. Use Sleuth or OpenTelemetry auto-propagation via `X-B3-*` or `traceparent`
C. MDC logging only
D. Avoid it

Answer: B
Explanation: These tools inject and forward headers automatically.
Insight: Enables end-to-end request tracing across service boundaries.

---

17. Why is the God Gateway anti-pattern dangerous in Spring Cloud Gateway setups?

A. It handles too little
B. Becomes a single point of failure and performance bottleneck
C. It’s too secure
D. Supports service discovery

Answer: B
Explanation: Doing too much (auth, transformation, routing) makes gateways bloated and brittle.
Insight: Offload concerns (e.g., auth to Keycloak; rate limit to mesh).

---

18. Which pattern helps scale reads independently from writes?

A. CQRS
B. Feign
C. Strangler
D. WebFlux

Answer: A
Explanation: Read/write paths can evolve independently using dedicated models.
Insight: Ideal for analytics, dashboards, and projections.

---

19. What is the Transactional Boundary Anti-pattern?

A. Wrapping multiple service calls in one transaction across network
B. Using retries
C. Logging too early
D. Using circuit breaker

Answer: A
Explanation: Distributed transactions across services break the isolation and consistency guarantees.
Insight: Design services to be independently consistent.

---

20. What does domain event publishing promote in Spring-based DDD systems?

A. Shared database
B. Decoupling and eventual consistency
C. Immediate sync updates
D. Logging

Answer: B
Explanation: Events convey state changes to other services asynchronously.
Insight: Encourages loosely-coupled, reactive architecture.

---

21. Why is implementing retry logic in consumers important in event-driven systems?

A. Avoids network errors
B. Prevents duplicate ID insertion
C. Helps recover from transient failures in downstream systems
D. Reduces code

Answer: C
Explanation: Network, DB, or third-party systems may fail temporarily.
Insight: Use Spring Retry or Resilience4j in consumer logic.

---

22. How should versioning be handled for Kafka topic schemas in Spring-based systems?

A. Add new field anytime
B. Delete unused fields
C. Register schemas in schema registry and evolve them carefully
D. Push code to consumers

Answer: C
Explanation: Ensures backward/forward compatibility.
Insight: Confluent Schema Registry helps enforce contracts.

---

23. What's a good pattern for aggregating calls to multiple downstream services in Spring?

A. Chain controllers
B. Service Aggregator pattern using `WebClient` with timeout and fallback
C. Use a filter
D. Use controller inheritance

Answer: B
Explanation: WebClient allows non-blocking fan-out/fan-in style aggregation.
Insight: Combine with Bulkhead, Retry, and timeout per call.

---

24. How does Backpressure relate to reactive systems like Spring WebFlux?

A. Prevents memory leaks by pausing data emitters when consumers are overwhelmed
B. Improves blocking IO
C. Adds latency
D. Disables concurrency

Answer: A
Explanation: Reactor supports backpressure to avoid OutOfMemory during fast producers and slow consumers.
Insight: Essential for streaming APIs and Kafka consumers.

---

25. What is a shared integration anti-pattern between Spring microservices?

A. Exposing REST API
B. Creating a shared client library for internal service logic
C. Using Feign
D. Using OAuth

Answer: B
Explanation: Shared clients introduce versioning lock, duplication, and hidden dependencies.
Insight: Instead, publish contracts (OpenAPI) and evolve independently.

Design Patterns vs Anti-Patterns (Cheat Sheet)

Category	Design Pattern	Anti-Pattern	Explanation / Real-World Insight
Service Communication	API Gateway Pattern	God Gateway / Backend-for-Frontend in Gateway	Gateway should not absorb all logic like auth, UI shaping, etc. Split UI & service orchestration properly.
Service Registration	Service Discovery (Eureka, Consul)	Hardcoded URLs / Static Host Configs	Discovery enables dynamic scaling and failover.
Reliability & Fault Tolerance	Circuit Breaker (Resilience4j)	Synchronous Chain Without Timeout	Prevent cascading failures. Always add timeouts and fallbacks.
Resilience & Containment	Bulkhead (ThreadPoolBulkhead)	Unbounded Threads / Shared Resources	Use isolated resource pools per service/endpoint.
Data Sharing	Database per Service Pattern	Shared Database Schema Across Microservices	Leads to schema coupling and cross-service regression.
Transaction Handling	Saga Pattern (Compensation & Coordination)	Distributed 2PC (XA Transactions)	Prefer eventual consistency over global locking.
State Sharing	Event-Carried State Transfer	Chained REST Calls with Full Data Fetch	Events allow async, independent state updates.
Inter-Service Integration	Feign/RestTemplate + Retry + Timeout	Deep Synchronous Chains (REST Chains)	Use async messaging or fallback when possible.
Data Flow	CQRS (Read/Write Separation)	One Entity for Everything (Anemic Domain Model)	Separate read models improve performance and clarity.
Decoupling	Domain Events (Spring Events / Kafka)	Tight Coupling with Shared DTOs	Events are better than direct service dependency.
API Evolution	Versioned APIs via Path or Header	Breaking Changes on Shared APIs	Always preserve backward compatibility.
Deployment	Sidecar & Ambassador Patterns	Logic in Docker ENTRYPOINT or Gateway	Offload logging, retries, observability to sidecars.
Scaling & Isolation	Stateless Services + Horizontal Pod Autoscaler	Sticky Sessions + Shared Memory	Statelessness simplifies scaling and recovery.
Observability	Distributed Tracing (Sleuth + Zipkin/OpenTelemetry)	Isolated Logs / No Correlation IDs	Always pass trace/context headers downstream.

Category	Design Pattern	Anti-Pattern	Explanation / Real-World Insight
Secrets Management	Config Server + Vault + Spring Profiles	Committing Secrets to Git or Environment	Use Kubernetes secrets, Vault, or AWS Secrets Manager.
Messaging	Outbox Pattern + Idempotent Consumers	Non-idempotent Consumers + No Offset Check	Duplicate messages are common. Always guard processing.
Performance	Async Processing (`@Async`, Kafka, Reactor)	Blocking IO in WebFlux or Shared Pools	Use non-blocking I/O, Reactor threads, and tune pools.
Monitoring	Spring Boot Actuator + Micrometer	Custom Logging Hacks for Monitoring	Use metrics, health, liveness/readiness endpoints.
Auth & Security	OAuth2 / JWT + Resource Server	Session Auth in Microservices	Token-based stateless auth scales better across nodes.
Config Management	Spring Cloud Config / Profile-based Settings	Per-service Config in WAR/JAR	Externalize configuration for environment parity.
Failure Recovery	Retry + Exponential Backoff	Blind Retry Loops or No Retry Logic	Retry smartly with delay/jitter, not endlessly.
Domain Modeling	Bounded Contexts + Aggregates	Entity Aggregation in Shared Model	Avoid single mega-domain model across services.
DevOps & Build	Dockerfile + Health Checks + ENV Configs	Fat Docker Images with Embedded Secrets	Minimal images with external secrets = best practice.
Security	Zero Trust + mTLS + Fine-Grained RBAC	Overtrust Internal Network or Global Admin Role	Every service should be authenticated + authorized.

Pro-Tips for Interviews & Architecture Reviews

These help **justify your design choices**, **show system-level awareness**, and **demonstrate real-world trade-offs** during interviews or technical design reviews.

---

1. Focus on boundaries — services, teams, and responsibilities

Explanation:
Each microservice should encapsulate a well-defined **business capability**, managed by a **small team**. Boundaries should reflect both **domain logic** and **team ownership**.

Example:

- **Anti-pattern**: A “user-service” that handles login, registration, role assignment, profile photos, and 2FA.
- **Improved**:
  - `auth-service`: login, token issuance
  - `profile-service`: user details and preferences
  - `roles-service`: RBAC and admin delegation

Interview Tip: Mention DDD and the "Team Topologies" principle. Say: *"We aligned our microservices with bounded contexts to ensure team autonomy and fast-paced delivery."*

---

2. Avoid “centralized” solutions unless explicitly orchestrated

Explanation:
Centralized logic (like a mega-gateway or one service calling all others) leads to **tight coupling**, **single points of failure**, and **scaling bottlenecks**.

Example:

- API Gateway that does:
  - All auth logic
  - DTO transformation
  - Rate limiting
  - DB joins for dashboards
- Split:
  - Spring Cloud Gateway → only route, filter
  - Backend-for-frontend (BFF) → aggregates data for specific UIs

Interview Tip: *"We decoupled gateway concerns and pushed aggregation to BFFs per channel (web/mobile) using WebClient with retry/fallback logic."*

---

3. Apply backpressure in async systems (Kafka, WebFlux)

Explanation:
Without **backpressure**, fast producers (e.g., Kafka topic) can overwhelm slow consumers, causing memory overflow or thread starvation.

Example:

- Kafka listener with unbounded thread pool or sync DB calls
- Use:

- `@KafkaListener(concurrency = 3)` + manual commit
- `Flux.limitRate(50)` in WebFlux
- Reactive DB drivers (R2DBC)

Interview Tip: *"We hit a bottleneck in a Kafka consumer due to lack of backpressure; we mitigated it using concurrent partitions + manual offset commit + idempotency checks."*

---

4. Favor fail-fast, recover-later over wait-forever

Explanation:
Blocking calls or infinite retries stall the thread pool, blocking downstream systems. **Fail-fast with fallback** improves system responsiveness and user experience.

Example:

- REST template call without timeout: hangs for 60s
- Feign/WebClient with:
  - Timeout: 2s
  - Retry: max 3
  - Fallback: default response or cache

```
java
CopyEdit
@CircuitBreaker(name = "inventory", fallbackMethod = "fallbackInventory")
public Inventory getInventory() {
   return
webClient.get().uri("/inventory").retrieve().bodyToMono(Inventory.class)
         .timeout(Duration.ofSeconds(2))
         .block();
}
```

Interview Tip: *"We adopt fail-fast patterns with short timeouts, graceful degradation via fallbacks, and alerting to catch persistent failures."*

---

5. Use health, metrics, and trace as first-class citizens

Explanation:
In production, **observability is non-negotiable**. You need to trace requests, monitor system health, and expose useful metrics.

Example:

- Spring Boot Actuator:
  - `/actuator/health`
  - `/actuator/metrics`
  - `/actuator/prometheus` (for Grafana)

- Spring Sleuth + Zipkin/OpenTelemetry
  - Trace ID injected across services
  - `X-B3-TraceId` or `traceparent` headers

Demo Scenario:

- Trace a request from `api-gateway` → `orders-service` → `payment-service`
- If payment fails, trace shows delay in external provider

Interview Tip: *"We embedded actuator endpoints into all services and hooked into Prometheus with Micrometer for service-level metrics and SLOs."*

---

6. Design for graceful degradation, not just resilience

Explanation:
Resilience protects from failure. **Graceful degradation** provides partial functionality even if dependencies fail.

Example:

- Payment service down → checkout flow still reserves cart and shows message: “We’ll notify you when payment succeeds.”
- Recommendation engine down → product page loads with no recommendations, not a 500 error

Tech Patterns:

- Circuit Breaker (`@CircuitBreaker`)
- Fallback (`fallbackMethod`)
- Cache/last-known state
- Alerting via Prometheus/Grafana

Interview Tip: *"Every critical path in our services includes a degradation fallback — from pricing cache to offline email retries."*

---

7. Log structured data, not just raw strings

Explanation:
Use **structured logs** (JSON or key-value format) for parsing by ELK/Fluentd.

Example:

```
json
CopyEdit
{
  "timestamp": "2024-01-01T12:00:00Z",
  "level": "INFO",
  "traceId": "abc123",
```

```
  "userId": "u123",
  "event": "OrderPlaced"
}
```

Interview Tip: *"Structured logs let us correlate user actions across services and enabled dashboards on user behavior."*

---

8. Treat local dev and CI environments as production-like

Explanation:
Avoid surprises by simulating production locally: use Docker Compose, environment parity, and secrets injection.

Example:

- Local `.env` + Spring Profiles + Docker secrets
- CI pipeline builds container, runs test with `Testcontainers` or ephemeral DBs

Interview Tip: *"Our team had minimal drift between staging and production thanks to Dockerized CI pipelines and environment parity."*

---

9. Prefer contracts over shared libraries

Explanation:
Shared code (DTOs, Feign clients) introduces coupling. Publishing **API contracts** promotes independence.

Example:

- OpenAPI (Swagger) docs exported → clients generate stubs
- Sharing Java DTO classes via internal JARs

Interview Tip: *"We generated clients via OpenAPI and evolved versions independently to avoid dependency hell."*

---

10. Design for observability from day 1, not as an afterthought

Explanation:
Add **traceability**, **metrics**, and **alerts** during development — not after outages occur.

Example:

- Include `traceId` in all logs
- Create health dashboards before Go-Live
- Define alert thresholds per service

Interview Tip: *"Our SLOs were backed by real-time dashboards that tracked latency, errors, and throughput by service and trace ID."*

Section 1: AWS IAM, KMS & Security Controls (10 MCQs)

---

1. What is the primary function of an IAM Role in AWS?

A. To store secrets securely
B. To provide temporary, assumable permissions to entities
C. To create EC2 instances
D. To manage billing preferences

Answer: B
Explanation: IAM Roles provide **temporary credentials** that can be assumed by users, services, or applications.
Real-world: Lambda assumes an IAM role to read from S3 or write to DynamoDB. STS (Security Token Service) is behind the scenes.

---

2. What is the difference between IAM Policy and IAM Role?

A. Roles define actions, policies define identity
B. Policies are permission sets; roles are identities that can assume them
C. They are identical
D. Policies store passwords

Answer: B
Explanation: IAM Policies are **JSON-based permission documents**. IAM Roles are **assumable identities** that policies attach to.
Real-world: A Kubernetes pod in ROSA can assume an IAM Role via IRSA (IAM Roles for Service Accounts) to access S3 securely.

---

3. Which AWS KMS feature ensures cryptographic separation between services?

A. Shared secrets
B. Key aliases
C. Customer Managed Keys (CMKs)
D. IAM MFA

Answer: C
Explanation: Customer Managed Keys (CMKs) allow you to control key policies, rotation, and grants — offering stronger isolation.
Real-world: Encrypt RDS, S3, or Lambda environment variables using different CMKs per service for tenant-level separation.

---

4. How can you restrict a Lambda function to read only from a specific S3 bucket?

A. Use a public-read ACL
B. Attach an S3 bucket policy
C. Create an IAM Role with a policy scoped to the bucket
D. Make bucket private and hope for best

Answer: C
Explanation: IAM Role attached to Lambda must include `s3:GetObject` for only the target bucket ARN.
Real-world: Helps implement least-privilege in event-driven data pipelines.

---

5. What is a key benefit of using AWS KMS key grants?

A. They support manual rotation
B. Fine-grained control of encryption use for a specific principal
C. They expose key material to the app
D. They are used for IAM role switching

Answer: B
Explanation: Grants let you delegate temporary use of KMS keys to services like Lambda or S3, without giving full key access.
Real-world: A Step Function may grant access to KMS to decrypt secrets only during execution.

---

6. What is the purpose of `Condition` in IAM policies?

A. Deny all access
B. Filter policies based on variables like time, IP, resource tags
C. Encrypt policies
D. Define policies for CloudTrail

Answer: B
Explanation: `Condition` allows you to enforce **context-aware access**, e.g., IP whitelisting, encryption enforced.
Real-world: `"Condition": {"StringEquals": {"s3:x-amz-server-side-encryption": "aws:kms"}}` ensures all writes to S3 are encrypted.

---

7. Which AWS service enables external KMS key usage (BYOK)?

A. IAM
B. Secrets Manager
C. CloudHSM
D. AWS KMS with External Key Store (XKS)

Answer: D

Explanation: XKS (External Key Store) lets you use **keys managed outside AWS**, giving regulatory or HSM-bound use cases flexibility.

Real-world: Useful for banking & compliance systems needing in-country encryption control.

---

8. What AWS IAM feature allows rotating access keys for a user?

A. MFA
B. Access Key Expiration
C. IAM Lifecycle Policy
D. Access Key Rotation API

Answer: D

Explanation: IAM supports key rotation either manually or via scripting, but access keys are discouraged in favor of roles.

Real-world: Rotate keys every 90 days minimum if using long-term access keys for automation.

---

9. Why is IAM policy scoping important in Kubernetes (e.g., ROSA)?

A. Because Kubernetes has its own IAM
B. Because Secrets need to be public
C. To limit pod access via IAM Roles for Service Accounts (IRSA)
D. So developers can bypass AWS Auth

Answer: C

Explanation: IRSA ensures pods get access only to permitted AWS resources using scoped IAM policies.

Real-world: A Kafka consumer pod can read from S3 but not write to RDS via IAM role attached to its service account.

---

10. What happens if a KMS-encrypted S3 object is accessed without permission to the key?

A. Access is granted via IAM
B. Access fails even if S3 policy allows it
C. It’s decrypted in plaintext
D. CloudTrail deletes the object

Answer: B

Explanation: S3 permissions and KMS permissions are evaluated **independently**.

Real-world: You may see 403 errors even with correct S3 policy if KMS permissions are missing.

Section 2: S3, Lambda, Step Functions, and Event-Driven Architecture (10 MCQs)

---

11. Which AWS service can directly trigger a Lambda function upon new object upload?

A. EC2
B. S3
C. Route 53
D. CloudTrail

Answer: B
Explanation: S3 can emit an event notification (e.g., `s3:ObjectCreated:*`) to Lambda when a new file is uploaded.
Real-world: Automatically process uploaded documents (e.g., resize images, run virus scan) without polling.

---

12. What happens if a Lambda function processing an S3 event fails repeatedly without a DLQ?

A. It retries forever
B. S3 deletes the object
C. The event is lost unless a DLQ is configured
D. Lambda auto-heals the function

Answer: C
Explanation: If no DLQ (Dead Letter Queue) or retry mechanism is defined, unhandled errors result in lost events.
Insight: Always attach a DLQ (SQS/SNS) to ensure observability of failures in production.

---

13. What is the primary advantage of Step Functions over chaining Lambdas manually?

A. It's cheaper
B. No IAM is needed
C. Built-in state management, retries, parallelism, and visual flow
D. It uses Java only

Answer: C
Explanation: Step Functions allow complex workflows with error handling, branching, parallel execution, and timeout control.
Real-world: AML/KYC processing pipeline → validate identity → OCR docs → score → store → notify.

---

14. Which of the following Lambda concurrency models helps prevent overload?

A. Reserved concurrency
B. Inline caching
C. Lambda Layers
D. Multithreaded mode

Answer: A

Explanation: Reserved concurrency caps the number of concurrent executions, protecting downstream systems.

Real-world: Prevent flooding RDS with 100s of simultaneous connections from spike in file uploads.

---

15. How can Step Functions interact with AWS services like RDS, S3, or DynamoDB without custom Lambda logic?

A. Use `AWS::Bash`
B. Use Service Integrations (AWS SDK integration)
C. Use Glue crawlers
D. Use AWS CLI inside Lambda

Answer: B

Explanation: Step Functions have direct integrations with services using AWS SDK without needing Lambda.

Real-world: Update a DynamoDB table → invoke a SageMaker job → publish to SNS, all without writing Lambda code.

---

16. Which of the following helps maintain event ordering in event-driven pipelines with S3 → Lambda → Kinesis?

A. S3 Object ACL
B. FIFO SQS Queues
C. Kinesis Partition Keys
D. IAM Trust Policies

Answer: C

Explanation: Kinesis ensures ordering within a shard. Choosing partition keys wisely can preserve logical order.

Insight: Partition by `userId` if order matters per user (e.g., financial events).

---

17. How does Lambda reuse containers across invocations?

A. Via concurrency settings
B. Via cold start cache

C. Via execution context reuse (warm starts)
D. Via Dockerfile volumes

Answer: C
Explanation: Lambda maintains a **warm execution context** across invocations to reduce latency.
Real-world: You can cache DB connections or ML models between invocations to save time.

---

18. What happens when you hit the account-level concurrency limit for Lambda?

A. Functions wait in a queue
B. New invocations are throttled (429)
C. AWS adds more VMs
D. Nothing happens

Answer: B
Explanation: Exceeding concurrency limits leads to throttling (HTTP 429 or dropped events depending on source).
Insight: Use metrics to monitor concurrency and scale limits accordingly.

---

19. What pattern ensures files uploaded to S3 are not partially processed by downstream Lambda?

A. Pre-signed URLs
B. Object Lock
C. S3 Event Notifications + object size checks
D. Use versioning

Answer: C
Explanation: S3 triggers Lambda as soon as an object is created. Add logic to ensure file size is complete or use a post-upload marker file.
Real-world: Ingestion of large logs → check metadata or split upload/finalize steps.

---

20. What is the default Lambda timeout limit?

A. 1 minute
B. 15 minutes
C. 5 minutes
D. 10 minutes

Answer: B
Explanation: Maximum timeout for Lambda is 15 minutes. Suitable for long-running processes like ML inference or batch aggregation.
Insight: Step Functions should be used for complex workflows instead of increasing timeouts indiscriminately.

Section 3: AWS MSK (Kafka), RDS, and Integration Patterns (10 MCQs)

---

21. In AWS MSK (Managed Kafka), what ensures high availability of topic partitions?

A. Using FIFO topics
B. Increasing partition size
C. Replication factor
D. Enabling DLQ

Answer: C
Explanation: Replication factor (typically 3) ensures partitions are copied across brokers for HA.
Real-world: In a production-grade ROSA setup, each topic is configured with a replication factor of 3 to ensure tolerance for broker failure.

---

22. What happens if a Kafka consumer group has more consumers than partitions?

A. Messages are dropped
B. Some consumers are idle
C. New partitions are automatically created
D. Load is evenly balanced

Answer: B
Explanation: Kafka assigns **one consumer per partition**. Extra consumers in a group remain idle.
Real-world: If a topic has 3 partitions and 5 consumer pods, 2 pods will sit idle unless more partitions are added.

---

23. What is the recommended pattern for integrating RDS with microservices in AWS?

A. Use long-lived JDBC connections
B. Use connection pools like HikariCP
C. Use EC2 NAT Gateway
D. Share the same DB schema across services

Answer: B
Explanation: RDS supports a limited number of concurrent connections. Using a pooler (like HikariCP) reduces overhead.

Real-world: In a Spring Boot app deployed on OpenShift, HikariCP with health checks ensures optimal DB usage under load.

---

24. Which of the following is a valid reason to place Kafka in AWS (MSK) outside OpenShift ROSA?

A. ROSA cannot run Java
B. Kafka on OpenShift is always free
C. Kafka scales better when isolated from app runtime clusters
D. Kafka doesn't support Kubernetes

Answer: C
Explanation: Kafka benefits from being isolated in a dedicated VPC/subnet for performance and fault isolation.
Real-world: In a high-throughput payment system, Kafka is provisioned in AWS MSK, and OpenShift pods connect via VPC peering.

---

25. In Kafka, which delivery guarantee ensures a message is never lost even if consumer crashes?

A. At-most-once
B. At-least-once
C. Best-effort
D. None

Answer: B
Explanation: At-least-once delivery ensures messages are reprocessed if acknowledgment fails. Requires idempotent processing to avoid duplication.
Real-world: Kafka → Lambda → RDS pipelines often use **idempotency keys** in DB writes to prevent duplicates.

---

26. Which RDS feature allows instant disaster recovery across regions?

A. Read Replica
B. Multi-AZ
C. Cross-region automated snapshot
D. Elastic IP

Answer: C
Explanation: Automated snapshot replication across regions allows restoration in another AWS region in case of full region failure.
Real-world: Financial apps may replicate snapshots of encrypted RDS PostgreSQL to comply with BCP (business continuity plan).

---

27. What is the best way to prevent a noisy Lambda from exhausting RDS connections?

A. Use Elastic IPs
B. Use RDS Proxy
C. Throttle RDS manually
D. Increase instance size

Answer: B
Explanation: RDS Proxy maintains a **connection pool** that safely manages concurrency and credential caching.
Real-world: In event-driven ingestion pipelines, Lambda bursts are common — RDS Proxy absorbs the spike.

---

28. How can Kafka help with event replay in a microservice system?

A. Kafka deletes messages immediately
B. Kafka stores message offset, so replay is manual
C. Consumers can reset their offset to reprocess events
D. It sends the event again automatically

Answer: C
Explanation: Consumers can reset the committed offset to an earlier value to **reconsume messages**.
Real-world: Replaying fraud signals or logs for post-incident analysis in banking systems.

---

29. Why is it important to partition Kafka topics effectively?

A. To make logs look nice
B. To improve message ordering
C. To scale consumer throughput and parallelism
D. To enable S3 integration

Answer: C
Explanation: More partitions = more parallelism. Helps you scale Kafka consumers horizontally.
Real-world: In a ROSA-based AML processing system, partitioning by `customerId` enabled tenant-aware scaling.

---

30. What is a downside of using at-least-once delivery in Kafka without idempotency?

A. Messages are lost
B. Messages can be duplicated
C. System crashes
D. Kafka topics get locked

Answer: B

Explanation: If a message fails after being processed but before committing, it may be retried — leading to duplicates.

Real-world: Always ensure writes to DB are **idempotent** (e.g., use `upsert` or deduplication key).

Section 4: Kubernetes Core + ROSA-Specific Concepts (10 MCQs)

---

31. What is the purpose of a liveness probe in Kubernetes?

A. To check if the service is scaled
B. To verify if the container is ready to serve traffic
C. To detect if a container is dead and restart it
D. To log resource usage

Answer: C

Explanation: Liveness probes detect hung containers. If the check fails repeatedly, K8s restarts the pod.

Real-world: A Java microservice with a memory leak might hang — the liveness probe triggers a pod restart to self-heal.

---

32. Which OpenShift feature enables managing cluster resources as code with version control?

A. Helm Charts
B. OpenShift GitOps (ArgoCD)
C. Jenkins Operator
D. ROSA CLI

Answer: B

Explanation: OpenShift GitOps (built on ArgoCD) enables declarative app management, rollback, and audit via Git-based workflows.

Real-world: Finance team uses GitOps to deploy AML rules, config maps, and role bindings from a secure Git repo.

---

33. What is the default behavior when a Kubernetes pod fails its readiness probe?

A. It is deleted
B. It is restarted
C. It is removed from the Service endpoint list
D. It scales down the ReplicaSet

Answer: C

Explanation: Readiness probe tells the Service whether a pod is ready to serve traffic. Failing = removed from load balancer.

Real-world: During app startup, readiness probes delay routing traffic until the service is fully initialized.

---

34. How are secrets stored by default in Kubernetes?

A. As encrypted blobs
B. As environment variables only
C. Base64-encoded in etcd
D. Not stored at all

Answer: C

Explanation: Kubernetes stores secrets base64-encoded in etcd. By default, this is **not encrypted** unless configured.

Real-world: On ROSA, configure encryption at rest for etcd or use external secret managers like HashiCorp Vault.

---

35. In OpenShift (ROSA), how are service accounts used to access AWS resources securely?

A. SSH keys in pods
B. S3 bucket policy injection
C. IAM Roles for Service Accounts (IRSA)
D. Static credentials

Answer: C

Explanation: IRSA lets OpenShift pods **assume AWS IAM roles** securely using OIDC federation.

Real-world: A pod reading from S3 assumes a dedicated IAM role via service account annotation + IRSA config.

---

36. What is the role of a Kubernetes ConfigMap?

A. Secret manager
B. Logs aggregator
C. Inject non-sensitive config (e.g., YAML, JSON) into pods
D. Persistent volume driver

Answer: C

Explanation: ConfigMaps store non-secret config (like env vars, files) and can be mounted into pods.

Real-world: A fraud rules engine pod loads JSON rules via ConfigMap mounted as a volume.

---

37. What is the benefit of using OpenShift Secrets over Kubernetes Secrets?

A. Automatic encryption at rest and RBAC integration
B. More secure by design
C. Enables use of Helm
D. Supports binary files only

Answer: A
Explanation: OpenShift enhances Secret security by encrypting them at rest and enforcing access via RBAC.
Real-world: OpenShift Secrets can store database credentials securely and integrate with OpenShift Console and oc CLI.

---

38. In Kubernetes, how do you control who can access what in the cluster?

A. Pod affinity rules
B. Horizontal Pod Autoscaler
C. RBAC: Role-Based Access Control
D. VolumeMounts

Answer: C
Explanation: RBAC defines roles and bindings that govern user/service access to K8s objects.
Real-world: Grant read-only access to a developer for ConfigMaps in the `dev` namespace using a RoleBinding.

---

39. What's the difference between ClusterRole and Role in Kubernetes RBAC?

A. Role is global, ClusterRole is namespaced
B. Role is for logs only
C. ClusterRole is for cluster-wide access; Role is for a namespace
D. There is no difference

Answer: C
Explanation: ClusterRole grants permissions across namespaces; Role is scoped to a single namespace.
Real-world: Use ClusterRole to define admin access; bind with RoleBinding at namespace level as needed.

---

40. What's a secure way to inject passwords into a pod running on OpenShift?

A. Hardcode in Dockerfile
B. Use environment variables in YAML
C. Mount an OpenShift Secret as a file or env var
D. Store in Git

Answer: C
Explanation: OpenShift Secrets can be mounted into pods and managed securely using RBAC.

Real-world: Store DB password in a secret → mount into Spring Boot pod → read via `@Value` or `System.getenv()`.

Section 5: Cross-Service Integration — Secrets, CI/CD, KMS, mTLS, Vaults (10 MCQs)

---

41. How can you securely pass secrets from AWS to OpenShift (ROSA) pods during CI/CD?

A. Use base64-encoded ConfigMaps
B. Use Jenkins global variables
C. Use AWS Secrets Manager + IRSA + projected volumes
D. Use environment variables in the Dockerfile

Answer: C
Explanation: IRSA enables OpenShift service accounts to assume IAM roles to pull secrets from AWS Secrets Manager securely.
Real-world: A ROSA-based app fetches DB credentials via IRSA-annotated pod + init container using `aws secretsmanager get-secret-value`.

---

42. In GitHub Actions, how do you safely store and use OpenShift cluster credentials?

A. Add to Dockerfile
B. Use GitHub Secrets + environment variables
C. Inline in `.yml` workflows
D. Use `kubectl` aliases

Answer: B
Explanation: GitHub Secrets encrypts sensitive data (tokens, kubeconfigs) for workflows.
Real-world: Use `oc login` with tokens stored in GitHub Secrets for deploying apps to ROSA.

---

43. What is the purpose of `oc extract secret/<name>` in OpenShift?

A. Exports logs
B. Debugs secrets
C. Pulls secret content into the local shell for inspection
D. Creates a new secret

Answer: C
Explanation: `oc extract` reads secret content (e.g., DB password, certificate) into stdout or

a file.

Real-world: Helps developers inspect TLS certs or credentials during staging/debug.

---

44. What AWS service can encrypt secrets, RDS data, and S3 objects with customer-managed keys?

A. IAM
B. KMS
C. ECS
D. WAF

Answer: B

Explanation: AWS KMS (Key Management Service) supports envelope encryption using CMKs (Customer Managed Keys).

Real-world: A Spring Boot app writes logs to an encrypted S3 bucket using a tenant-specific CMK key.

---

45. What is the primary benefit of mTLS between microservices?

A. Fast routing
B. Static IP support
C. Auth + encryption at transport level
D. More logging

Answer: C

Explanation: Mutual TLS (mTLS) ensures both client and server authenticate each other + encrypts traffic in transit.

Real-world: OpenShift Service Mesh (Istio) can auto-inject mTLS between pods in zero-trust environments.

---

46. What is a secure way to store certificates for mTLS in OpenShift?

A. Embed in JAR file
B. Add to Docker CMD
C. Use OpenShift Secret (TLS type)
D. Keep in Git repo

Answer: C

Explanation: TLS secrets (`type: kubernetes.io/tls`) store `tls.crt` and `tls.key`, mountable into pods.

Real-world: A ROSA microservice uses a mounted TLS secret for client-side cert validation in mTLS.

---

47. What is the role of Vault (e.g., HashiCorp) in a ROSA + AWS setup?

A. Scheduling containers
B. Serving frontend
C. Centralized secret lifecycle and rotation
D. Running RDS

Answer: C
Explanation: Vault provides dynamic secrets, secret versioning, and auto-expiry.
Real-world: Vault issues short-lived PostgreSQL credentials or AWS temporary credentials to microservices.

---

48. What helps GitHub Actions securely authenticate to AWS (e.g., for deploying to ECR)?

A. Shared credentials file in repo
B. IAM user hardcoded in script
C. OIDC-based GitHub-AWS trust relationship
D. Kubeconfig injection

Answer: C
Explanation: GitHub can use **OIDC to assume AWS IAM roles** without access keys. Safer than long-lived secrets.
Real-world: Set up IAM trust policy with GitHub's OIDC provider to push Docker images to ECR.

---

49. What is the benefit of combining AWS KMS with Secrets Manager or OpenShift Secrets?

A. Easier dashboards
B. Unlimited storage
C. Fine-grained encryption and audit trail
D. Native multi-region support

Answer: C
Explanation: Secrets encrypted with KMS benefit from **envelope encryption**, key rotation, and audit logging.
Real-world: Audit which user decrypted what secret — critical in financial compliance apps.

---

50. Why would a developer prefer declarative YAML + GitOps over imperative `kubectl` commands?

A. It's harder to debug
B. It reduces automation
C. Provides traceability, versioning, and repeatable deployments
D. Git isn't supported by OpenShift

Answer: C

Explanation: GitOps uses Git as the **source of truth**, enabling review, audit, and rollback.

Real-world: Finance CI/CD pipelines push YAML to Git, and ArgoCD syncs apps across dev/staging/prod environments.

Section 1: Advanced Kafka Usage

1. What is the key property to enable idempotent Kafka producer behavior?

A. `enable.exactly.once`
B. `acks=0`
C. `enable.idempotence=true`
D. `retries=0`

Answer: C

Explanation: `enable.idempotence=true` ensures producer retries do not cause duplicate records, preserving exactly-once semantics for a single partition.

Real-world: Used in financial systems where retrying messages must not result in duplicate transactions.

---

2. Which setting must be used with `enable.idempotence=true` to achieve transactional guarantees?

A. `transaction.id`
B. `group.id`
C. `auto.commit=true`
D. `partitioner.class`

Answer: A

Explanation: Defining a `transaction.id` allows Kafka producers to participate in transactions using the Kafka transactional API.

Real-world: Used in order-processing systems to atomically update Kafka and a database.

---

3. What happens if a consumer group rebalances while a long-running message is being processed?

A. The message is lost
B. The consumer is killed
C. Processing may be duplicated unless offsets are committed manually after processing
D. Kafka delays rebalance

Answer: C

Explanation: To avoid duplication, commit offsets **after** successful processing. Otherwise, the same message may be reprocessed.

Real-world: Use `ackMode=MANUAL_IMMEDIATE` in Spring Kafka for precise control.

---

4. Which Kafka topic type retains the latest message per key and is often used for caching purposes?

A. Ephemeral topic
B. Compacted topic
C. Stream topic
D. Fan-out topic

Answer: B

Explanation: Compacted topics retain only the latest message per key by removing older records.

Real-world: Product catalog updates, user preferences, or last known device location tracking.

---

5. What does the Kafka producer setting `acks=all` do?

A. Ignores acknowledgments
B. Waits for only leader acknowledgment
C. Waits for acknowledgment from all in-sync replicas
D. Causes retries to fail

Answer: C

Explanation: Ensures high durability by waiting for all ISR replicas to acknowledge a write.

Real-world: Required for data integrity in systems handling compliance events or financial transactions.

---

6. What Kafka feature enables message flow from one topic to another using processing logic?

A. Kafka Connect
B. Kafka Streams
C. Kafka MirrorMaker
D. Kafka Compaction

Answer: B

Explanation: Kafka Streams is a client library for real-time transformation of Kafka topics using stateful/stateless operators.

Real-world: Detect suspicious transactions by joining streams of user login and payment topics.

---

7. In Spring Kafka, how do you implement a retry strategy with backoff for failed message processing?

A. Use `@KafkaRetryable`
B. Use `RetryTemplate` in KafkaListenerContainerFactory
C. Use Kafka Connect transformer
D. Set `retries=0`

Answer: B
Explanation: `RetryTemplate` with exponential backoff allows you to retry message processing before sending to DLQ.
Real-world: Retry up to 3 times with 1s, 2s, and 5s gaps for transient DB/network failures.

---

8. What is the main benefit of using dead-letter topics in Kafka consumer error handling?

A. Persist failed records for manual inspection
B. Delete corrupted messages
C. Skip schema validation
D. Encrypt messages

Answer: A
Explanation: DLQs allow storing failed messages so they can be analyzed and replayed.
Real-world: In a fraud detection system, corrupt or invalid records are routed to DLQ for forensics and patching.

---

9. What is the purpose of a Kafka Schema Registry?

A. To manage topic ACLs
B. To persist consumer offsets
C. To store and validate Avro/Protobuf schemas
D. To balance partitions

Answer: C
Explanation: Schema Registry stores schemas for producers and consumers and validates compatibility across versions.
Real-world: Used in event-driven systems where versioned data models evolve (e.g., `customer-v1`, `customer-v2`).

---

10. In a Kafka cluster with many consumers and topics, which component handles group coordination?

A. Zookeeper
B. Kafka Broker Leader
C. Consumer Coordinator
D. Schema Registry

Answer: C

Explanation: Kafka assigns a coordinator broker per consumer group to manage partition assignments and heartbeat.

Real-world: Essential to prevent consumer timeouts in large clusters — tuning session and heartbeat timeouts matters.

11. What does Kafka's "exactly-once semantics" guarantee?

A. One message is delivered to one consumer only
B. No message is ever lost
C. One message is processed only once, even with retries
D. Every consumer receives a copy of every message

Answer: C

Explanation: Kafka's EOS ensures a message is neither lost nor duplicated when properly configured with idempotent producer and transactional writes.

Real-world: Crucial in billing systems to avoid overcharging users.

---

12. How do you ensure messages with the same key go to the same partition?

A. Use a custom header
B. Set the message key when producing
C. Configure round-robin partitioner
D. Increase consumer group size

Answer: B

Explanation: Kafka's default partitioner uses the message key to determine partition, ensuring ordering per key.

Real-world: Ensures all updates for a `customerId` go to the same partition to maintain order.

---

13. What does Kafka’s linger.ms configuration do in the producer?

A. Waits before retrying a failed request
B. Waits for more records to batch before sending
C. Delays consumer polling
D. Controls heartbeat interval

Answer: B

Explanation: `linger.ms` controls batching efficiency by introducing a small delay to accumulate more records.

Real-world: Tuning this improves throughput for high-volume systems with frequent small messages.

---

14. What happens if a Kafka topic has more consumers than partitions?

A. Messages are lost
B. Extra consumers sit idle
C. Kafka creates new partitions
D. Consumers form sub-groups

Answer: B
Explanation: Consumers in a group are assigned one partition max — any extras remain idle.
Real-world: Helps diagnose underutilized pods in autoscaling Kafka consumers.

---

15. What is the purpose of Kafka Connect?

A. Manage consumer groups
B. Provide reactive Kafka clients
C. Integrate external systems (e.g., DB, Elasticsearch) with Kafka
D. Compress messages in transit

Answer: C
Explanation: Kafka Connect uses connectors to move data between Kafka and external sources/sinks.
Real-world: Postgres → Kafka (source connector), Kafka → S3 (sink connector).

---

16. Which Kafka Stream operation is used to perform stateful windowed joins?

A. `filter()`
B. `groupByKey()`
C. `join()` with `TimeWindows`
D. `flatMap()`

Answer: C
Explanation: `join()` with `TimeWindows.of(...)` enables joining two streams within a defined time window.
Real-world: Login event + payment within 3 seconds → trigger fraud check.

---

17. What Kafka feature allows replaying a topic from a specific offset?

A. Kafka Replay Module
B. Offset Seek
C. Kafka Timer
D. GroupBalancer

Answer: B
Explanation: Consumers can `seek(offset)` or reset offsets to replay events from a point in time.
Real-world: Used for reprocessing after a bug fix in payment status handler.

---

18. How does Kafka ensure replication of data across brokers?

A. Round-robin broker assignment
B. ISR (in-sync replicas) mechanism
C. Consumer groups
D. DLQ fallback

Answer: B
Explanation: ISR tracks which brokers have fully synced copies of a partition.
Real-world: Used in high availability clusters to recover from broker failure.

---

19. What happens if the schema in Schema Registry is incompatible with incoming Avro data?

A. Message is dropped silently
B. Consumer hangs
C. Producer throws a serialization exception
D. Schema is auto-updated

Answer: C
Explanation: Producers will fail serialization if schema validation fails — helps catch breaking changes early.
Real-world: Avoids pushing broken messages due to field type mismatch in evolving schemas.

---

20. What is a major limitation of Kafka when used for event sourcing with large aggregates?

A. Low throughput
B. Hard to maintain partitions
C. Replaying all events can be slow
D. High memory usage

Answer: C
Explanation: Replaying thousands of events to rebuild a large object graph can be time-consuming.
Real-world: Use snapshotting with event sourcing to mitigate performance issues.

---

21. Which Kafka Streams feature allows saving state across processing steps?

A. Stateless transformation
B. KTable
C. Kafka Consumer cache
D. Avro encoding

Answer: B

Explanation: KTable stores compacted state and updates as changes come in — enabling aggregation and joins.

Real-world: Maintain the latest account balance as a KTable in financial apps.

---

22. What is the function of a custom partitioner in Kafka producer?

A. Filters messages
B. Encrypts payload
C. Controls how messages are routed to partitions
D. Manages schema compatibility

Answer: C

Explanation: Custom partitioners allow producers to override partition routing logic based on message content.

Real-world: Route all `VIP customers` to a dedicated partition for priority processing.

---

23. What Kafka feature allows producers to include metadata like correlation ID in messages?

A. KafkaMetadataStore
B. KafkaHeader
C. ProducerProps
D. KafkaJoinKey

Answer: B

Explanation: `ProducerRecord` supports headers, which are often used to carry tracing metadata (e.g., correlation IDs).

Real-world: Used with OpenTelemetry for trace propagation across microservices.

---

24. How can Kafka ensure message order across consumers?

A. Use compacted topics
B. Use a single partition per key and one consumer
C. Use fetch groups
D. Set `acks=0`

Answer: B

Explanation: Kafka maintains order **only within a single partition**. If a key is consistently routed to one partition, order is preserved.

Real-world: Ensures order of fund transfers for an account is not broken.

---

25. Why is it important to manually commit offsets after processing in critical systems?

A. To avoid duplicate partition creation
B. To improve broker availability
C. To guarantee message has been fully processed before acknowledging
D. To reduce CPU usage

Answer: C
Explanation: Manual offset control lets the app **acknowledge only after processing is successful**, ensuring reliability.
Real-world: In loan approval systems, DB update must succeed before marking message as processed.

Section 2: Distributed Transaction Patterns

1. What is the main reason to avoid 2PC (Two-Phase Commit) in distributed microservices?

A. It lacks consistency
B. It doesn't support rollback
C. It's synchronous, complex, and blocks resources
D. It requires GraphQL

Answer: C
Explanation: 2PC blocks participants and is tightly coupled, making it unsuitable for resilient microservices.
Real-world: Legacy systems using XA datasources have difficulty scaling in cloud-native environments.

---

2. Which pattern achieves distributed consistency through compensating actions?

A. 2PC
B. XA transactions
C. SAGA Pattern
D. Outbox Pattern

Answer: C
Explanation: SAGA uses local transactions and compensating actions for rollback — better for asynchronous coordination.
Real-world: In a flight booking system, if hotel reservation fails, the flight is cancelled via a compensating action.

---

3. In a SAGA choreography, how do services communicate state?

A. Through direct REST calls
B. Shared database
C. Publishing events to a message broker
D. Using XA protocol

Answer: C
Explanation: In choreography, services emit and react to domain events — no central orchestrator.
Real-world: Order service emits `OrderPlaced → PaymentService → ShippingService`.

---

4. Which of the following best describes the Outbox Pattern?

A. Stores all domain events in Kafka
B. Logs SQL errors
C. Writes events to a DB table and forwards them asynchronously
D. Skips event delivery when DB is unavailable

Answer: C
Explanation: Events are written to an outbox table within the same DB transaction, later forwarded to a broker.
Real-world: Ensures order confirmation email isn't lost even if Kafka is temporarily down.

---

5. What problem does the Transactional Outbox solve?

A. Message reordering
B. Ensures DB update and message publish are atomic
C. Circular dependencies
D. Missing schema validations

Answer: B
Explanation: Guarantees DB and event state changes are consistent and non-divergent.
Real-world: Used when an update in RDS must result in a message sent to Kafka reliably.

---

6. In Orchestration SAGA, what component is responsible for managing the transaction flow?

A. Kafka topic
B. Orchestrator service
C. Consumer group
D. Message serializer

Answer: B

Explanation: Orchestration centralizes logic in a controller that calls other services and handles failures.

Real-world: BPMN engines like Camunda or a custom Spring Boot SAGA orchestrator.

---

7. What is the main trade-off of eventual consistency in distributed systems?

A. High latency
B. Partial availability
C. Temporary inconsistency between services
D. Tight coupling

Answer: C

Explanation: Systems may not reflect real-time changes immediately but converge over time.

Real-world: A read on Shipping may show status “PENDING” even after Payment succeeded.

---

8. What does the term "idempotency" mean in distributed APIs?

A. Prevents data replication
B. Each request has a different result
C. Multiple identical calls have the same effect
D. Data is eventually deleted

Answer: C

Explanation: Idempotency ensures retrying a call won’t cause side-effects (e.g., duplicate orders).

Real-world: PUT `/order/{id}` ensures update is safe even if client retries due to timeout.

---

9. What is the role of a “compensation service” in a SAGA?

A. Handles tracing of services
B. Monitors orchestrator
C. Performs undo logic for a previously successful step
D. Logs performance metrics

Answer: C

Explanation: It implements a logical rollback by reversing the effects of previous actions.

Real-world: Cancel invoice, issue refund, or restock item.

---

10. Which design is an anti-pattern for distributed transactions?

A. Using Outbox Pattern
B. Emitting domain events
C. Sharing database tables across services
D. Using correlation IDs

Answer: C

Explanation: Shared databases violate service boundaries, create tight coupling and hinder autonomy.

Real-world: Leads to integration hell and broken teams in a large retail platform.

11. Which strategy ensures that a transaction either commits to both the database and Kafka or rolls back both?

A. Kafka ACL
B. Idempotent producer
C. Transactional Outbox
D. Spring AOP

Answer: C

Explanation: Transactional outbox ensures both DB and message are committed atomically using a single local DB transaction.

Real-world: A bank's transaction posting system uses it to send reliable accounting events to Kafka.

---

12. Why are distributed locks considered an anti-pattern in microservices?

A. They improve consistency
B. They scale across nodes
C. They introduce latency, single point of failure, and violate service independence
D. They speed up event processing

Answer: C

Explanation: Distributed locks are fragile and don't scale. They can lead to deadlocks or inconsistent states in failures.

Real-world: Better to use idempotency and compensating actions than rely on locks for resource access.

---

13. In Spring Boot, how can you identify and trace transaction boundaries in a microservice system?

A. Use `@RequestMapping`
B. Enable `spring.jpa.show-sql=true`
C. Apply `@Transactional` and use tracing tools like Sleuth/OpenTelemetry
D. Enable actuator endpoints

Answer: C

Explanation: `@Transactional` defines method-level transaction scopes, while distributed tracing tools help visualize span propagation.

Real-world: Sleuth tags DB span and Kafka producer span for end-to-end correlation.

---

14. Which of these ensures exactly-once delivery across DB + Kafka in Spring Boot?

A. Kafka Streams
B. KafkaTemplate
C. Transactional Kafka Producer + Outbox
D. Consumer Groups

Answer: C

Explanation: Outbox + transactional producer avoids duplicates when DB is updated and message is published in tandem.

Real-world: In a CRM system, customer onboarding event is persisted and pushed without loss or duplication.

---

15. What's a drawback of SAGA with Orchestration compared to Choreography?

A. More code duplication
B. Increased latency due to centralized control
C. Eventual consistency
D. Requires direct DB access

Answer: B

Explanation: Orchestration introduces a central coordinator, which adds latency and may become a bottleneck.

Real-world: A SAGA orchestrator calling 5 services serially vs. parallel invocations in Choreography.

---

16. What does "atomicity" mean in the context of distributed systems?

A. Each service runs its own DB
B. A process either fully completes or fully fails without partial side effects
C. Each node has a replica
D. Services use shared state

Answer: B

Explanation: Atomicity ensures that either all operations succeed or none — critical for financial operations.

Real-world: Funds should never be debited if credit to the recipient fails.

---

17. Why is retry logic dangerous without idempotency in distributed transactions?

A. It increases throughput
B. It can trigger multiple side-effects like duplicate billing
C. It lowers latency
D. It simplifies rollback

Answer: B
Explanation: Retrying non-idempotent actions can cause incorrect updates (e.g., double-insert in DB).
Real-world: An email campaign sends the same email 5 times on timeout.

---

18. What does "compensating transaction" typically NOT do?

A. Log the error
B. Rollback a previous successful action
C. Undo side effects like sending emails
D. Commit a new business step

Answer: D
Explanation: Compensation undoes effects; it is not used for normal forward business progress.
Real-world: Undo hotel booking or issue refund if payment fails later in flow.

---

19. What is the recommended message sequencing strategy in distributed pipelines?

A. Use hash-based partitioning
B. Use timestamps
C. Use a global monotonic counter or version
D. Use random UUIDs

Answer: C
Explanation: Sequence IDs or versions help identify the order of processing and resolve out-of-order execution.
Real-world: Track item state with increasing version numbers to handle late-arriving messages.

---

20. In a SAGA pattern, what’s the challenge when using a long compensation chain?

A. High CPU usage
B. Non-determinism in rollback order
C. Tight service coupling
D. Large payloads

Answer: B
Explanation: Multiple compensations must be applied in **reverse order**, which increases

complexity.

Real-world: Failure at step 5 means triggering 4 → 3 → 2 → 1 in reverse.

---

21. How can a service detect a duplicate event in the Outbox pattern?

A. By ignoring all repeated events
B. Using a deduplication ID or hash key
C. Kafka automatically blocks it
D. Using time-based filtering

Answer: B
Explanation: A unique event ID is persisted, and duplicates are checked before processing.
Real-world: Payments carry a transaction ID; if seen before, processing is skipped.

---

22. What is the major limitation of using XA (distributed transaction) in microservices?

A. It doesn't support NoSQL
B. It’s not supported by Kafka
C. It introduces high coupling and global transaction manager bottlenecks
D. It requires eventual consistency

Answer: C
Explanation: XA protocols are not microservices-friendly due to tight coordination and lack of fault tolerance.
Real-world: Replaced with eventual consistency and retry in cloud-native systems.

---

23. In distributed transaction design, which is more important than consistency for availability?

A. Low code
B. Event sourcing
C. Partition tolerance
D. Idempotency

Answer: D
Explanation: Idempotency ensures that retries or replay don’t result in data corruption — essential for high availability.
Real-world: Idempotent APIs allow safe retries without concern for exact transaction timing.

---

24. What technique is often used to manage ordering guarantees in a distributed SAGA?

A. Locking tables
B. Global transaction coordinator

C. Sequence tokens or event versioning
D. Timestamp-only ordering

Answer: C
Explanation: Tokens ensure actions are processed in the right order, especially during compensation.
Real-world: Order status updates like CREATED → APPROVED → SHIPPED must follow strict sequencing.

---

25. What is a major pitfall when implementing distributed rollback logic manually?

A. Using too many databases
B. Too many logs
C. Side-effects from compensations may fail themselves and require further handling
D. Messages are not traceable

Answer: C
Explanation: Compensating actions (e.g., refund API) may fail due to business or technical issues — leading to partial rollbacks.
Real-world: Flight cancel refund fails because payment gateway is offline — needs retry + alert.

Section 3: Reactive Stack Deep-Dive (Spring WebFlux & Beyond)

1. What is the key difference between `Mono` and `Flux` in Project Reactor?

A. Mono is asynchronous, Flux is synchronous
B. Mono represents 0–1 items, Flux 0–N items
C. Mono uses threads, Flux uses memory
D. Mono is blocking, Flux is non-blocking

Answer: B
Explanation: Mono is for a single item (like a REST response); Flux for multiple (like a stream of DB records).
Real-world: `Mono<User>` for user profile API, `Flux<Order>` for order history.

---

2. What happens if a `Flux` emits more items than a slow subscriber can consume?

A. Nothing — subscriber will ignore it
B. Flux drops all data
C. Backpressure is applied
D. Flux switches to Mono

Answer: C

Explanation: Backpressure is a mechanism to avoid overwhelming the subscriber — it requests data at its pace.

Real-world: Streaming millions of records from DB to client while avoiding memory bloat.

---

3. What is the role of `Schedulers.parallel()` in a reactive pipeline?

A. Blocks the main thread
B. Schedules blocking DB calls
C. Runs processing in separate worker threads
D. Prevents retries

Answer: C

Explanation: Parallel scheduler allows multi-threaded execution — useful for CPU-heavy tasks.

Real-world: Run image processing or document parsing in a reactive data enrichment step.

---

4. Which HTTP client is reactive and non-blocking in Spring?

A. RestTemplate
B. WebClient
C. ApacheHttpClient
D. HttpURLConnection

Answer: B

Explanation: `WebClient` is the non-blocking replacement for RestTemplate in reactive apps.

Real-world: Used to call third-party APIs in streaming data flows.

---

5. How do you return a stream of results from a WebFlux controller?

A. Return `List<T>`
B. Return `ResponseEntity<List<T>>`
C. Return `Flux<T>`
D. Use `@Async`

Answer: C

Explanation: `Flux<T>` allows Spring to stream data progressively to the client.

Real-world: Streaming real-time logs, alerts, or sensor data to dashboards.

---

6. What will happen if you call a blocking JDBC call inside a reactive controller?

A. It works fine
B. It blocks the entire event loop and degrades performance

C. It retries 3 times
D. It gets parallelized automatically

Answer: B
Explanation: Blocking calls inside non-blocking threads block the Netty event loop → thread starvation.
Real-world: Use R2DBC instead of JDBC in WebFlux apps.

---

7. What is the correct way to switch threads in reactive streams?

A. `.subscribeOn()`
B. `.observeOn()`
C. `.publish()`
D. `.queueOn()`

Answer: A
Explanation: `subscribeOn` controls the thread for the source execution, often combined with `Schedulers.boundedElastic()` for blocking I/O.
Real-world: Switching Mongo reactive call to elastic scheduler to prevent Netty blocking.

---

8. What is the purpose of `StepVerifier` in reactive programming?

A. Load testing tool
B. DB query performance benchmark
C. Unit testing tool for reactive streams
D. OAuth verifier

Answer: C
Explanation: `StepVerifier` from Reactor Test allows you to validate emitted items, errors, and completion.
Real-world: Used to test `Flux.fromIterable(List.of("A", "B")).map(...).filter(...)` logic step-by-step.

---

9. In reactive retry patterns, what is the correct operator to retry on failure?

A. `.block()`
B. `.repeat()`
C. `.retry()`
D. `.timeout()`

Answer: C
Explanation: `.retry(n)` re-executes a stream in case of error.
Real-world: Retry calling a flaky external payment service 3 times before giving up.

---

10. What is `onErrorResume()` used for in reactive streams?

A. Retrying DB transactions
B. Swallowing all errors
C. Handling error by switching to fallback stream
D. Delaying execution

Answer: C
Explanation: `onErrorResume()` lets you gracefully fallback to another `Mono` or `Flux` on error.
Real-world: Fallback to cached value or mock response on service failure.

11. What happens when `.block()` is called in a WebFlux controller?

A. The method returns immediately
B. The controller becomes reactive
C. It blocks the event loop and causes performance degradation
D. It triggers retry logic

Answer: C
Explanation: `.block()` forces synchronous behavior and blocks the reactive thread — this violates the reactive model.
Real-world: Use `.block()` only in tests or CLI utilities, not in production WebFlux code.

---

12. Which of the following is a valid reactive database option for Spring?

A. Spring JDBC
B. JPA with Hibernate
C. R2DBC
D. JDBC Template

Answer: C
Explanation: R2DBC provides non-blocking reactive access to relational databases like PostgreSQL, MySQL.
Real-world: Ideal when integrating a reactive REST API with a PostgreSQL DB in WebFlux.

---

13. What does `.flatMap()` do in a reactive pipeline?

A. Converts a stream to blocking
B. Maps each value to a new stream and flattens the result
C. Returns null
D. Applies retry logic

Answer: B

Explanation: `flatMap()` transforms each element into a `Publisher` and flattens the resulting stream.

Real-world: Fetch multiple product reviews per product ID and flatten into a single `Flux<Review>`.

---

14. What reactive operator is used to limit rate of data emission?

A. `.filter()`
B. `.take()`
C. `.delayElements()`
D. `.sample()`

Answer: C

Explanation: `delayElements(Duration.ofMillis(x))` introduces delay between emissions.

Real-world: Throttle outgoing email alerts in an alerting system.

---

15. Which transport protocol is used under the hood by WebFlux for non-blocking I/O?

A. Tomcat
B. Jetty
C. Undertow
D. Netty

Answer: D

Explanation: Spring WebFlux uses Project Reactor and Netty for non-blocking, asynchronous execution.

Real-world: Useful for high-concurrency workloads like streaming APIs or WebSockets.

---

16. What is the result of returning `Flux.empty()` from a controller?

A. 400 Bad Request
B. 204 No Content
C. 500 Internal Server Error
D. Null Pointer Exception

Answer: B

Explanation: `Flux.empty()` emits no items and completes — WebFlux translates this to 204.

Real-world: Used in endpoints like `/healthcheck` or `/ping` when no result is needed.

---

17. Why is `subscribe()` rarely used in WebFlux business logic?

A. It blocks the stream
B. It bypasses reactive pipeline lifecycle management
C. It crashes the server
D. It stops retries

Answer: B
Explanation: `subscribe()` executes eagerly and disconnects from the reactive chain lifecycle — Spring handles subscription internally.
Real-world: Use `subscribe()` for side-effects or fire-and-forget logic (e.g., audit logs), but never for control flow.

---

18. How do you apply a timeout in WebFlux for an external call?

A. Use `@Timeout`
B. Use `WebClient.create().timeout()`
C. Use `.timeout(Duration.ofSeconds(x))`
D. WebFlux doesn't support timeout

Answer: C
Explanation: `Mono.timeout(Duration)` cancels the stream if no item is emitted in the time window.
Real-world: Cancel external API calls after 3s to avoid hanging the entire request chain.

---

19. What does `.concatMap()` provide that `.flatMap()` does not?

A. Synchronous behavior
B. Concurrent processing
C. Ordered execution
D. Multiple subscriptions

Answer: C
Explanation: `.concatMap()` preserves the order of elements — processes them one at a time.
Real-world: Important when streaming logs or audit trails in time order.

---

20. When is `boundedElastic` scheduler recommended in reactive systems?

A. For CPU-bound tasks
B. For I/O-blocking operations like DB or file reads
C. For messaging
D. Never use it

Answer: B
Explanation: `boundedElastic()` is used for blocking I/O within a reactive context — it

has a dynamic but capped thread pool.

Real-world: Reading files from disk, querying JDBC as fallback in a reactive system.

---

21. What tool can help visualize and debug reactive sequences?

A. Spring DevTools
B. StepVerifier
C. BlockHound
D. Sleuth

Answer: C

Explanation: BlockHound detects blocking calls on reactive threads at runtime and throws errors.

Real-world: Helps spot `.block()` sneaking into reactive chains in development.

---

22. What does `map()` do in a reactive stream?

A. Modifies control flow
B. Transforms each element
C. Flattens a publisher
D. Adds delay

Answer: B

Explanation: `map()` applies a transformation to each emitted item.

Real-world: `Flux<String>` → uppercase each name before returning.

---

23. Why is using JDBC in WebFlux discouraged?

A. It’s reactive
B. It’s fast
C. JDBC blocks threads, violating the reactive non-blocking model
D. JDBC can’t run in containers

Answer: C

Explanation: JDBC is a blocking API and doesn’t play well with Netty or other non-blocking runtimes.

Real-world: Introduces thread starvation under load — use R2DBC instead.

---

24. What is `zip()` used for in reactive programming?

A. Compress streams
B. Combine multiple streams by index
C. Retry failed streams
D. Delay execution

Answer: B
Explanation: `zip()` combines values from multiple publishers as tuples (like a zipper).
Real-world: Combine user info and permissions in a single call.

---

25. What is a major anti-pattern in reactive systems?

A. Avoiding `.block()`
B. Using backpressure
C. Mixing blocking and non-blocking code in the same flow
D. Using `Mono.empty()`

Answer: C
Explanation: Mixing blocking logic in a reactive pipeline causes thread contention, poor scalability, and hard-to-debug performance drops.
Real-world: `Mono.fromCallable(() -> restTemplate.getForObject(...))` is an anti-pattern.

Section: Observability & Tracing

1. What does a "Trace ID" represent in distributed tracing?

A. A unique request ID across all systems involved
B. ID for a specific microservice
C. Logging configuration file
D. Metric name in Prometheus

Answer: A
Explanation: A trace ID identifies a single request that may pass through multiple services and systems.
Real-world: A POST request from a mobile app triggers user, payment, and notification services — all share the same Trace ID.

---

2. Which tool is commonly used to visualize distributed traces in Spring Boot apps?

A. Logstash
B. Jaeger
C. SonarQube
D. Kafka UI

Answer: B
Explanation: Jaeger helps visualize traces, spans, and request lifecycles across microservices.
Real-world: Used to diagnose bottlenecks by tracking time taken across chained services.

---

3. In OpenTelemetry, what is a “span”?

A. Log entry in a database
B. HTTP response header
C. A single unit of work with start and end times
D. Entire user session

Answer: C
Explanation: A span represents an operation like DB query or API call, linked to a larger trace.
Real-world: "sendEmail()" or "fetchCustomerDetails()" each become spans under a trace.

---

4. How can correlation IDs be propagated in Spring Boot apps?

A. Via hardcoded UUIDs
B. Using filters/interceptors to inject into headers/logs
C. Static constants
D. Through Kafka partitions

Answer: B
Explanation: Spring allows injecting correlation IDs into `MDC` via filter/interceptor and propagating via headers.
Real-world: A traceable `X-Correlation-ID` appears in logs and travels across service calls.

---

5. What does Micrometer provide in Spring Boot apps?

A. Dependency injection
B. JSON serialization
C. Metrics collection abstraction for multiple backends
D. Logging to disk

Answer: C
Explanation: Micrometer supports metrics export to Prometheus, Datadog, New Relic, etc.
Real-world: Service exposes `/actuator/prometheus` endpoint for Prometheus scraping via Micrometer.

---

6. Which annotation enables Spring Boot actuator health endpoints?

A. `@Actuator`
B. `@HealthEndpoint`
C. Spring Boot auto-configures it with `spring-boot-starter-actuator`
D. `@Metrics`

Answer: C
Explanation: Simply adding `spring-boot-starter-actuator` enables default health, info, and metrics endpoints.

Real-world: `/actuator/health`, `/actuator/metrics`, and custom health checks for DB or Kafka.

---

7. Why is tenant-based metric tagging important in multi-tenant applications?

A. To calculate log size per tenant
B. For security access
C. To segregate metrics for billing, SLAs, and analytics
D. To display tenant logo

Answer: C
Explanation: Adding tags like `tenantId` in metrics enables per-tenant dashboards, billing, and anomaly detection.
Real-world: In SaaS, response times and error counts can be filtered by each tenant in Grafana.

---

8. What is a common anti-pattern when implementing custom metrics?

A. Tagging by tenant ID
B. Incrementing counters
C. Using unbounded cardinality in tags
D. Using summaries

Answer: C
Explanation: Metrics with too many unique tag values (e.g., user ID, UUID) can overwhelm Prometheus and cause memory issues.
Real-world: Avoid tagging metrics with high-cardinality fields like IP address or request ID.

---

9. How do you propagate trace context in an async task in Spring?

A. Use static variables
B. Copy headers manually
C. Use `ThreadLocal`
D. Use context propagation tools like OpenTelemetry's `ContextPropagators`

Answer: D
Explanation: Trace context is lost across threads unless explicitly propagated using OpenTelemetry or Project Reactor's context.
Real-world: Submit to executor pool → trace headers must be manually restored or auto-propagated.

---

10. What's the difference between tracing and logging?

A. Tracing is for real-time metrics only
B. Logging shows static snapshots; tracing shows end-to-end request flow

C. Logging replaces tracing
D. They are the same

Answer: B
Explanation: Tracing maps a complete flow of a request across services; logs are static messages tied to context.
Real-world: Traces help root cause 400ms delays by showing where latency occurs — unlike logs alone.

11. How can Jaeger trace data be sent from a Spring Boot app using OpenTelemetry?

A. Directly through JDBC
B. Exported using OpenTelemetry SDK and gRPC/HTTP exporter
C. Through log files
D. Via Kafka

Answer: B
Explanation: OpenTelemetry SDK exports spans using OTLP over gRPC or HTTP to Jaeger Collector.
Real-world: Spring Boot + OpenTelemetry auto-config + OTLP exporter pushes traces to Jaeger running in Docker.

---

12. What does `B3` refer to in distributed tracing?

A. Binary security framework
B. Zipkin's HTTP header-based trace context format
C. A JVM tuning flag
D. A Micrometer annotation

Answer: B
Explanation: B3 is a set of HTTP headers (`X-B3-TraceId`, `X-B3-SpanId`, etc.) used to propagate trace context across services.
Real-world: Spring Cloud Sleuth used B3 format before moving to W3C Trace Context.

---

13. What type of metric is used to measure response latency over time?

A. Gauge
B. Timer
C. Counter
D. Tag

Answer: B
Explanation: Timer tracks the duration of operations and also counts how often they occur.

Real-world: API response latency (`http.server.requests`) is tracked using timers in Micrometer.

---

14. Why is `MeterFilter` used in Spring Micrometer?

A. To modify request payloads
B. To add error handling
C. To filter/transform metrics before publishing
D. To enable Actuator endpoints

Answer: C
Explanation: `MeterFilter` lets you control, deny, or transform metric names/tags globally.
Real-world: Remove sensitive or high-cardinality tags from being exposed in Prometheus.

---

15. What's the role of `MDC` in logging?

A. Stores microservice deployment config
B. Passes metadata between Kafka messages
C. Stores per-thread context like correlation ID
D. Encrypts logs

Answer: C
Explanation: `Mapped Diagnostic Context (MDC)` allows injecting correlation ID or tenant info into every log line.
Real-world: Helps trace logs for request `REQ-1234` across multiple log entries.

---

16. What does a correlation ID typically represent?

A. Service version
B. Error ID
C. Unique identifier of a request for end-to-end traceability
D. Request payload

Answer: C
Explanation: Correlation ID helps track a single request as it flows through distributed services.
Real-world: Generated by API Gateway or client, passed in `X-Correlation-ID` header.

---

17. What is the role of `Histogram` in metrics collection?

A. To capture only total counts
B. To store event samples over time
C. To provide a frequency distribution of values
D. To log request payload

Answer: C

Explanation: Histograms divide a metric into buckets to observe frequency — useful for latency SLAs.

Real-world: Bucketed response times: `<50ms`, `<100ms`, `<200ms`, `<500ms`, etc.

---

18. What is span sampling in OpenTelemetry used for?

A. For sending metrics only on Sundays
B. To reduce network overhead by sampling only some spans
C. To duplicate spans
D. For retry logic

Answer: B

Explanation: Sampling reduces volume of tracing data by selecting a subset of spans to record/export.

Real-world: In high-load production systems, 10% of requests are traced.

---

19. What OpenTelemetry component is used to receive and process trace data centrally?

A. Tracer
B. Collector
C. Prometheus
D. Sleuth

Answer: B

Explanation: OpenTelemetry Collector is a vendor-agnostic service to receive/process/export telemetry data.

Real-world: Collector receives spans from services, enriches them, and sends to Jaeger or OTEL backend.

---

20. Which Prometheus metric type grows monotonically?

A. Gauge
B. Summary
C. Counter
D. Histogram

Answer: C

Explanation: A counter only increases (or resets on restart), used for things like HTTP requests served.

Real-world: `http_server_requests_total` counts HTTP calls.

---

21. Which best describes the difference between logs, metrics, and traces?

A. All are logs in JSON
B. Metrics = real-time health, Logs = detail, Traces = request flow
C. Metrics = large files, Traces = configuration, Logs = charts
D. No difference

Answer: B
Explanation: Metrics give real-time stats, logs offer details/debug info, traces follow request flow.
Real-world: CPU high (metric) → trace slow span → log exception.

---

22. What OpenTelemetry SDK concept wraps operations like DB query, HTTP call, etc.?

A. View
B. EventEmitter
C. Span
D. Route

Answer: C
Explanation: Spans define units of work in traces, with timing, tags, and relationships.
Real-world: "get-customer-from-db", "validate-payment", etc.

---

23. What is a major benefit of integrating tracing with log correlation?

A. Easy to debug across systems with trace ID embedded in logs
B. Faster DB queries
C. Shorter logs
D. No code needed

Answer: A
Explanation: Logs enriched with Trace/Span IDs allow searching across services for a single transaction.
Real-world: Search Kibana by `trace.id=abc123` to see full lifecycle.

---

24. What configuration enables exposing metrics at `/actuator/prometheus`?

A. `management.endpoints.web.exposure.include=prometheus`
B. `spring.security.prometheus.enable=true`
C. `prometheus.port=8080`
D. No config needed

Answer: A
Explanation: Actuator endpoint must be included in the list of exposed endpoints.
Real-world: Used in K8s with Prometheus scraping from each service pod.

---

25. In a multi-tenant system, how should correlation ID and metrics be managed?

A. Generate once, ignore tenant
B. Generate per thread
C. Correlation ID in headers/logs, tenant ID in tags
D. Same correlation ID for all users

Answer: C
Explanation: Correlation IDs track requests; tenant info enables slicing metrics/logs by customer.
Real-world: Metrics like `api.latency{tenant="acme"}` and logs with `X-Correlation-ID`.

Scenario-Based Mock Questions

. What would you do if a REST endpoint intermittently returns 500 errors in production?

- **Situation**: `/api/user/{id}` occasionally fails in live with `500 Internal Server Error`.
- **Walkthrough**:
  - Check logs with correlation ID via MDC or trace ID (Jaeger/OpenTelemetry).
  - Confirm downstream DB or service call isn't timing out.
  - Add `@ControllerAdvice` to return meaningful error info.
  - Use circuit breaker (`Resilience4j`) to isolate cause.
  - Add retry with exponential backoff if failure is transient.

---

2. What if service A is calling service B and both are slow? How do you debug latency?

- **Situation**: Response time is 2.5 seconds. No obvious error.
- **Walkthrough**:
  - Enable distributed tracing (OpenTelemetry + Jaeger).

- Break down spans: check service A → B RPC delay.
- Check DB query time, DNS, serialization, and queuing latency.
- Add metrics via Micrometer: start timers at HTTP request and DB layer.
- Optimize: cache where needed, pre-aggregate data, or reduce payload size.

---

3. What would you do if one tenant's API usage degrades others in a multi-tenant SaaS?

- **Situation**: Tenant X sends too many requests and degrades shared resources.
- **Walkthrough**:
  - Add tenant-aware metrics with Micrometer (`@Timed(extraTags="tenant", "#tenantId")`).
  - Apply rate limiting (Bucket4j or Istio Envoy-based).
  - Isolate tenant workloads via K8s namespaces or queues.
  - Separate tenant-specific storage or service tiers for high volume customers.

---

4. What if Kafka consumer is skipping messages randomly?

- **Situation**: Some messages not processed, no error logs.
- **Walkthrough**:
  - Confirm auto-commit is disabled or delayed (`enable.auto.commit=false`).
  - Check rebalance events — consumer group may be unstable.
  - Add DLQ and retry logic using Spring Kafka RetryTopic mechanism.
  - Use idempotent consumers to prevent duplicates on retries.

---

5. What if one of your APIs returns data too slowly only under heavy load?

- **Situation**: `/api/search` takes 5ms under dev load, but 4s under prod traffic.
- **Walkthrough**:
  - Use Prometheus + Grafana to chart latency over time.
  - Analyze GC and thread pool (Tomcat, Netty) saturation.
  - Add query caching or pagination.
  - Consider moving logic to async `@Async`, `WebFlux`, or Kafka batch processing.

---

6. What if a Kubernetes pod keeps restarting with OOMKilled?

- **Situation**: Application pod fails due to memory exhaustion.
- **Walkthrough**:
  - Inspect memory usage via `kubectl top pod`.
  - Tune JVM options (`-Xmx`, GC tuning).
  - Add proper `resources.requests` and `resources.limits` in deployment.
  - Investigate memory leaks with tools like Eclipse MAT or VisualVM.

---

7. What if sensitive configs (like DB password) are committed to Git?

- **Situation**: `.yml` with credentials was pushed by mistake.
- **Walkthrough**:
  - Immediately rotate the secrets (DB creds, JWT keys, etc.).
  - Use tools like GitGuardian or TruffleHog to scan for leaked secrets.
  - Replace inline secrets with `@Value("${db.password}")` backed by Spring Vault / AWS Secrets Manager.
  - Add `.gitignore`, `.gitattributes`, and commit hooks to prevent recurrence.

---

8. What if your Spring Boot app takes 40 seconds to start?

- **Situation**: Startup time is unacceptable in production.
- **Walkthrough**:
  - Use `--debug` and `spring-boot-startup-tracker` to profile bean init.
  - Delay heavy initializations using `SmartLifecycle` or `@Lazy`.
  - Replace unused `@ComponentScan` with targeted base packages.
  - Move to GraalVM Native Image if startup is critical (e.g., Lambdas).

---

9. What if Dev environment works, but test/QA fails due to missing configs?

- **Situation**: Service fails in QA but not in local.
- **Walkthrough**:
  - Compare Spring profiles: `application-dev.yml` vs `application-qa.yml`.
  - Verify externalized config (Config Server or Vault) is mounted properly.

- Use `@ConfigurationProperties` for environment-specific properties.
- Include health check on startup for critical config validation.

---

10. What if Kafka producer retries indefinitely on send failure?

- **Situation**: Kafka publish stalls under network glitch.
- **Walkthrough**:
  - Check `retries`, `max.in.flight.requests.per.connection`, and `acks` config.
  - Wrap producer with custom retry + fallback logic (e.g., DLQ fallback).
  - Log and alarm on send failure metrics.
  - Avoid infinite retry loop — add circuit breaker or TTL policy.

---

11. What if DB transactions partially persist after service crash?

- **Situation**: DB has inconsistent data after service failure mid-operation.
- **Walkthrough**:
  - Wrap logic in `@Transactional`, avoid non-transactional external API inside.
  - Use `TransactionalEventListener` for post-commit async calls.
  - In distributed scenarios, adopt Saga pattern (via Kafka or state machine).
  - Audit using transaction logs and recovery hooks.

---

12. What if an exception handler fails and crashes the app?

- **Situation**: An unhandled error inside `@ExceptionHandler`.
- **Walkthrough**:
  - Guard all handlers with fallback logic (`try/catch`).
  - Move to a global `@ControllerAdvice` class for central control.
  - Log all exceptions — never suppress unless intended.
  - Add `@ResponseStatus` to return clear codes.

---

13. What if a client misuses your open public API and floods it?

- **Situation**: Bots or users cause heavy API load.
- **Walkthrough**:

- Add API Gateway with rate limits and API keys.
- Use Spring Cloud Gateway filters to enforce tenant-level quotas.
- Enable threat protection (WAF, CAPTCHA, throttling).
- Add caching or snapshot APIs for frequently accessed data.

---

14. What if your observability stack shows 200ms delay in 10% of requests?

- **Situation**: Prometheus/Grafana shows a performance tail.
- **Walkthrough**:
  - Trace slow spans using Jaeger.
  - Identify specific endpoints or data paths causing slowness.
  - Enable thread dump capture on threshold.
  - Compare GC behavior and async thread pool saturation.

---

15. What if your system must scale for a 10x load in a new region within 2 weeks?

- **Situation**: Business demands urgent regional launch.
- **Walkthrough**:
  - Use infra as code (Terraform/Helm) to replicate infra.
  - Make config fully environment-driven (e.g., profile, secrets, endpoints).
  - Use K8s HPA + cloud autoscaling for elasticity.
  - Prefill region-specific caches or CDNs.
  - Test failover + chaos scenarios beforehand.

16. What would you do if JWT tokens expired too frequently and users complained of logout?

- **Situation**: Users report forced logouts after 5 minutes.
- **Walkthrough**:
  - Analyze JWT expiry settings (`exp`, `nbf`) in auth server.
  - Issue short-lived **access tokens** + long-lived **refresh tokens**.
  - Build refresh token endpoint (if using OAuth2/OIDC).
  - Secure refresh tokens with `HttpOnly`, rotation, and inactivity expiry.

---

17. What if your Spring Boot microservices are failing mutual TLS handshakes?

- **Situation**: Services behind Istio/Linkerd show TLS handshake errors.
- **Walkthrough**:
  - Confirm both services trust the same CA and validate certificates.
  - Use `spring.security.ssl.trust-store` and `key-store` properly.
  - Test with `openssl s_client` or `curl --cert`.
  - Avoid self-signed certs in prod unless added to trust store.

---

18. What if config values fail to load in a Spring Cloud Config environment on container restart?

- **Situation**: Dockerized services fail due to `ConfigServer unreachable`.
- **Walkthrough**:
  - Ensure the `bootstrap.yml` is in place — not `application.yml` (prevents early config loading).
  - Enable retry with exponential backoff on `ConfigClient`.
  - Fall back to local `application.yml` or use `failFast=false`.
  - Monitor Config Server health and scale it if needed.

---

19. What would you do if K8s autoscaling spawns pods that fail health checks and get stuck in CrashLoopBackOff?

- **Situation**: HPA triggers new pods that fail and restart endlessly.
- **Walkthrough**:
  - Examine readiness/liveness probes (`tcpSocket`, `httpGet`, startup delay).
  - Check pod logs: app might be slow to start or misconfigured.
  - Delay HPA thresholds or configure PodDisruptionBudget.
  - Add pre-stop hook to cleanly drain connections.

---

20. What if consuming large Avro/Protobuf Kafka messages fails with deserialization errors?

- **Situation**: Consumers throw `SchemaMismatchException` or `EOFException`.
- **Walkthrough**:
  - Use **Schema Registry** and pin versions explicitly.

- Set `max.partition.fetch.bytes` and `fetch.message.max.bytes`.
- Validate producer and consumer use same schema evolution rules.
- Fallback to JSON with version tagging if needed.

---

21. What if downstream systems are slow and you're losing Kafka messages during spikes?

- **Situation**: Consumer lags increase, some messages skipped.
- **Walkthrough**:
  - Decouple using buffer/queue + backpressure (`max.poll.records`, `max.poll.interval.ms`).
  - Scale consumers via K8s HPA or additional partitions.
  - Use retry topic or DLQ to isolate failure cases.
  - Apply circuit breakers if downstream causes cascading delays.

---

22. What if developers bypassed Spring Security filters for internal tools?

- **Situation**: Devs disabled JWT or security for `/admin` endpoints in dev/prod.
- **Walkthrough**:
  - Enforce policy-as-code: deny unsafe filters in CI using static analysis (Checkmarx, SonarQube).
  - Enforce RBAC in production profile; never allow fallback/noop security beans.
  - Use Spring Profiles with `@Profile("dev")` to isolate open endpoints.

---

23. What if your service must deliver alerts in real-time across channels (SMS, Email, WebSocket)?

- **Situation**: Alerting needs to reach different channels depending on priority.
- **Walkthrough**:
  - Use Kafka topic per channel (e.g., `alert.sms`, `alert.email`) and multi-topic consumers.
  - Apply Spring Cloud Stream with multiple binders or `@KafkaListener`.
  - Use Routing Slip pattern if orchestration needed.
  - WebSocket gateway with STOMP can stream to frontend users.

---

24. What if a service emits high-cardinality metrics and causes Prometheus performance issues?

- **Situation**: Metric labels are dynamic (e.g., `userId`, `sessionId`).
- **Walkthrough**:
  - Avoid high-cardinality tags: use coarse dimensions (`tenant`, `region`, `featureFlag`).
  - Aggregate metrics at app layer or push sanitized metrics via push gateway.
  - Drop dynamic labels using Micrometer `MeterFilter.denyUnless()`.

---

25. What if your system integrates a third-party API that's rate-limited and causes cascading failures?

- **Situation**: External API throttles you, failures bubble up to user.
- **Walkthrough**:
  - Add bounded queues and bulkhead pattern (isolation).
  - Use Resilience4j's `RateLimiter` + circuit breaker.
  - Retry with backoff and exponential delay.
  - Fall back with cache, stub response, or UI message to user.

36. What if a Co-Operative Bank core system is hosted on-prem but fintech modules must be cloud-native?

- **Situation**: Core CBS runs on AS400, but new AML/KYC modules must live on AWS.
- **Walkthrough**:
  - Introduce **asynchronous message bridge** (Kafka or MQ) for CBS ↔ Cloud comms.
  - Use **DMZ/bastion** with **IP whitelisting**, **TLS termination**, and **VPC endpoints**.
  - Encrypt all data-in-transit and use **data minimization** techniques.
  - Use **service mesh + gateway** to control cross-environment traffic.

---

37. What if RBI restricts production workload to "GovCloud" but UAT and testing are on public cloud?

- **Situation**: Test in AWS Commercial, Prod must go to AWS GovCloud (India).
- **Walkthrough**:
  - Use **Terraform workspaces or environments** to separate state and targets.

- Ensure all sensitive infra uses **KMS with HSM-backed keys**.
- Configure separate pipelines with `env: govcloud` tag-based IAM enforcement.
- Avoid public AMIs — use hardened private AMIs for GovCloud.

---

38. What if daily batch settlement jobs run on-prem but APIs are served from the cloud?

- **Situation**: Overnight reconciliation processes are delayed due to network latency.
- **Walkthrough**:
  - Deploy **secure SFTP bridge** or use **AWS Direct Connect/VPN** for consistency.
  - Time API ingestion to allow for delayed batch reconciliation.
  - Introduce a **job dispatcher** that emits job status events to the cloud for monitoring.
  - Design APIs to reflect **eventual consistency**, not strict sync.

---

39. What if a blacklisted IP from RBI's threat list accesses your login endpoint in cloud logs?

- **Situation**: Login requests from known bad IPs detected in logs.
- **Walkthrough**:
  - Integrate threat intel feeds into WAF or API Gateway access control.
  - Apply **GeoIP filtering** + **anomaly-based IP rate limiting**.
  - Store all failed login attempts in a tamper-proof log (e.g., KMS-encrypted S3 with write-once policy).
  - Trigger auto-alert via Lambda/SNS when suspicious IPs hit login endpoints.

---

40. What if CoOp bank still uses flat-file transfers over FTP, but fintech platform demands Kafka?

- **Situation**: Legacy FTP jobs must feed modern Kafka-based pipelines.
- **Walkthrough**:
  - Introduce **poller agent** that watches FTP directory and converts to Kafka messages.
  - Apply **schema validation** before publishing to ensure format integrity.
  - Store original files in S3 as raw backup for audit trail.
  - Maintain **file ID + job metadata in a lookup DB** to avoid duplication.

---

41. What if a TCoOp bank's CBS requires batch settlement per branch, but regulators demand real-time logs?

- **Situation**: Settlement per branch done EOD, regulators want hourly access logs.
- **Walkthrough**:
  - Introduce **near-real-time audit service** that mirrors all user logins/accesses.
  - Use **Fluentd or Filebeat** to ingest logs from branches into a central SIEM/S3.
  - Push logs through **event bridge (Kafka)** for compliance dashboard.
  - Tag logs with **branch ID + operator ID** for traceability.

---

42. What if customer onboarding requires Aadhaar validation, but cloud-hosted app cannot reach UIDAI due to IP restrictions?

- **Situation**: UIDAI endpoint only accessible from registered government IPs.
- **Walkthrough**:
  - Use **hybrid relay proxy** deployed on registered on-prem IP.
  - Make cloud call UIDAI via this proxy over mutual TLS.
  - Log request-response trail with signature and timestamp.
  - Encrypt Aadhaar data using **RBI-prescribed cryptography standard (e.g., AES-256 GCM)**.

---

43. What if RBI requires DR copy of customer data to be in India, but your workloads run multi-region?

- **Situation**: AWS runs across Mumbai and Singapore.
- **Walkthrough**:
  - Use **region-aware data policies**: All RDS + S3 in Mumbai are master.
  - Singapore only serves metadata/masking tokens.
  - DR replication within India region (Mumbai → Hyderabad) via **multi-AZ or Route53 failover**.
  - Enforce **S3 bucket policy denying PutObject outside India region**.

---

44. What if different banks in a CoOp ecosystem want to whitelist only specific APIs from your microservices?

- **Situation**: Some clients want `/verifyKYC`, others want `/validatePAN`.
- **Walkthrough**:

- Use **Spring Cloud Gateway** or **API Gateway** with dynamic route filters.
- Introduce **client ID-based route mapping** via configuration DB.
- Apply **resource-level JWT scopes** (e.g., `scope:kyc.read`) per API.
- Add **API versioning + rate limiting** to prevent shared abuse.

---

45. What if a regulator mandates immutable logs of all financial transactions and staff operations?

- **Situation**: Requirement for WORM (Write Once, Read Many) store.
- **Walkthrough**:
  - Use **AWS S3 Object Lock** in Compliance Mode.
  - Log financial events using **structured JSON + checksums**.
  - Store `SHA256` hash of each event in **immutable audit index** (e.g., QLDB, Ledger DB).
  - Rotate audit trail hash chains for integrity verification.

---

46. What if your payment microservice must route to different clearing channels based on SLA windows?

- **Situation**: NEFT (10 AM–5 PM), UPI (24x7), IMPS (fallback).
- **Walkthrough**:
  - Use **RoutingStrategy bean** based on `LocalTime.now()` or CRON.
  - Maintain **clearing matrix config** in DB or config server.
  - Apply **CircuitBreaker** on each route — fallback to next available.
  - Expose `/clearingWindowStatus` API for visibility.

---

47. What if customer KYC documents must be stored securely but allow facial verification from cloud AI APIs?

- **Situation**: Files stored on-prem, but cloud AI models needed for facial analysis.
- **Walkthrough**:
  - Extract only cropped face image + blur background before upload.
  - Use **pseudonymized ID token** in filename.
  - Signed URL with **TTL < 1 minute** to call AWS Rekognition or similar.
  - Delete face image post-processing or store tokenized variant.

---

48. What if you're migrating to AWS, but RBI requires TLS v1.3 and FIPS-140-2 enforcement?

- **Situation**: App must meet strict crypto requirements in prod.
- **Walkthrough**:
  - Use **AWS ALB/NLB with TLSv1.3 listener policies**.
  - Enable **FIPS endpoints** on services (e.g., S3, KMS).
  - Spring Boot must use **BouncyCastle FIPS provider** if needed.
  - Pen test all exposed APIs + audit via Inspector/GuardDuty.

---

49. What if multiple TCoOp banks require isolated SLAs and custom reporting but share the same codebase?

- **Situation**: Multi-tenant SaaS for CoOps, each with different audit needs.
- **Walkthrough**:
  - Design tenant-aware services using `@RequestScope TenantContext`.
  - Configure **per-tenant report generators** using strategy pattern.
  - Metrics tagged by `tenantId` (Micrometer) + separate dashboards (Grafana).
  - Per-tenant S3 bucket or prefix for report isolation.

---

50. What if cloud-native fintech modules have to integrate with a private leased line to an RTGS switch?

- **Situation**: Secure link to core payment infra cannot be exposed over internet.
- **Walkthrough**:
  - Use **AWS Direct Connect** or **VPN over leased line** from data center to VPC.
  - Host integration pod on **dedicated subnet** with egress to leased line only.
  - Use sidecar or reverse proxy to standardize RTGS payloads.
  - Monitor link with heartbeat APIs and failover alerts.

FinTech Compliance-Only Drill Set (20 Scenarios)

---

51. What if your mobile app logs show PAN and Aadhaar values in crash reports?

- **Violation**: Exposes **PII** in logs – violates RBI + **DPDP** and **PCI-DSS 3.2.1**.
- **Solution**:
  - Mask all PAN/Aadhaar except last 4 digits in logs.
  - Enable **log scrubbing filters** on mobile + backend.
  - Set `spring.log.masked-fields=pan,aadhaar,password` via config.
  - Retain logs for audit, but restrict log access via IAM and KMS.

---

52. What if RBI asks for complete transaction traceability of a failed NEFT transfer 60 days old?

- **Requirement**: **7 years retention** + complete trace.
- **Solution**:
  - Store all transaction metadata in **immutable format** (e.g., S3 + SHA256 + QLDB).
  - Include **trace ID, customer ID, timestamp, failure reason**.
  - Use OpenTelemetry spans and store `traceId` in database alongside business ID.
  - Expose audit endpoint for compliance team with RBAC + KMS audit trail.

---

53. What if a customer's data is stored in Singapore S3 bucket but residency requires India-only?

- **Violation**: **RBI Master Circular on data localization**.
- **Solution**:
  - Enforce **bucket policy with `aws:RequestedRegion != ap-south-1` deny**.
  - Use SCP (Service Control Policy) to block services in non-compliant regions.
  - Implement **CI pipeline rule** to fail deploys with non-IN region targets.
  - Migrate non-compliant S3 objects using AWS CLI + KMS-IN encryption key.

---

54. What if SOC2 audit finds passwords stored as SHA-1 hashes in legacy user table?

- **Violation**: **Weak hashing violates SOC2 / PCI-DSS encryption requirements**.
- **Solution**:
  - Migrate to `bcrypt` or `PBKDF2` with per-user salt.

- Log and audit all password resets or hash transformations.
- Use Spring Security's `DelegatingPasswordEncoder`.
- Maintain dual hash fields (`sha1_pass`, `bcrypt_pass`) temporarily for backward compatibility.

---

55. What if DPDP Act compliance requires traceable consent for every loan approval but app backend has no audit log?

- **Violation**: Consent cannot be proven retroactively.
- **Solution**:
  - Use **Consent Ledger** (e.g., QLDB, PostgreSQL immutable schema).
  - Store `userId`, `consentType`, `timestamp`, `IP`, `device`, `signedDocId`.
  - Record this during onboarding + show consent preview screen before approval.
  - Auto-reject transaction if consent record not found.

---

56. What if RBI mandates user OTP-based eKYC but your system skips OTP in fallback flow?

- **Violation**: Fails **UIDAI eKYC** norms.
- **Solution**:
  - Enforce **OTP match as mandatory precondition** for eKYC API.
  - Capture failed/fraudulent bypass attempts with audit ID + session metadata.
  - Lock user session if OTP attempt limit exceeded.
  - Use Spring AOP to validate every controller flow invokes OTP layer.

---

57. What if a third-party credit scoring API violates RBI digital lending norms on data sharing?

- **Violation**: Lending partners cannot share behavioral data to unauthorized parties.
- **Solution**:
  - Maintain **partner registry** with roles, purposes, and API access logs.
  - Classify data types: Financial, Personal, Behavioral.
  - Restrict access via **purpose-bound tokens or signed JWT scopes**.
  - Auto-expire shared tokens and enforce one-time use.

---

58. What if your cloud S3 bucket holding account statements has no encryption enabled?

- **Violation**: Violates **PCI-DSS Section 3.4** and **RBI Data Security guidelines**.
- **Solution**:
  - Enforce **S3 default encryption with KMS** (AES-256).
  - Use SCP to deny `PutObject` without `x-amz-server-side-encryption`.
  - Trigger AWS Config rule to auto-detect and notify if encryption disabled.

---

59. What if audit team flags lack of user-role privilege logs for all CRUD operations?

- **Violation**: Fails **SOC2**, **RBI**, and **ISO 27001** traceability.
- **Solution**:
  - Add Spring Security context (username, role) to MDC and persist per request.
  - Store in `audit_log` table: `user`, `role`, `resource`, `action`, `old_value`, `new_value`.
  - Encrypt audit logs + backup to WORM S3 bucket.

---

60. What if RBI demands “replay protection” and “tamper-evidence” for payment APIs?

- **Compliance**: Ensure **replay attack protection** and **signed payloads**.
- **Solution**:
  - Use `X-Request-Id` + timestamp in headers.
  - Sign payloads with HMAC using server-shared secret.
  - Validate timestamp difference within ±5 mins.
  - Reject duplicate requests via Redis/cache nonce tracking.

---

61. What if production secrets were rotated without proof of audit controls?

- **Risk**: Fails **SOC2** Section CC6.1 (Access Controls).
- **Solution**:
  - Store all secrets in AWS Secrets Manager with versioning.
  - Log every access with IAM role, source IP, time.
  - Auto-rotate every 30 days and maintain audit reports (CloudTrail).

---

62. What if PAN data was shared over webhook to CRM partner without consent?

- **Breach**: Violates **DPDP Act**, **RBI privacy mandates**.
- **Solution**:
  - Apply **consent scope matching** before every outbound webhook.
  - Use masking at message formatter layer.
  - Maintain log of all webhook payloads with `audit_token`.

---

63. What if customers file RTI to get details of algorithm used for auto-loan rejection?

- **Requirement**: **Explainability under RBI's AI Fairness guidelines**.
- **Solution**:
  - Store ML decision metadata: features used, weight, model version.
  - Provide simplified explanation via pre-defined decision trees or logic.
  - Add `/loan-decision/{id}/explanation` API.

---

64. What if customer requests account deletion under DPDP but is still under AML watchlist?

- **Conflict**: **Data erasure vs legal retention**.
- **Solution**:
  - Retain AML-flagged user data under **legal hold policy**.
  - Provide customer with reason + appeal path.
  - Mark data as inactive + locked, but not deleted.
  - Automate flag-based data lifecycle exceptions.

---

65. What if customer profile is used to cross-sell insurance without explicit opt-in?

- **Violation**: Digital Lending + Fair Use guidelines.
- **Solution**:
  - Use **Consent Category: Cross-sell**, separated from KYC/usage.
  - Do not bundle product upsell inside essential workflow.

- Create `opt-in-token` table for every upsell action.

---

66. What if an AML alert is generated but not reviewed in 24 hrs due to backlog?

- **SLA Breach**: Regulatory non-compliance.
- **Solution**:
  - Auto-prioritize alerts by risk level and assign to reviewer queue.
  - Maintain `alert_acknowledged_at` and `assigned_to` fields.
  - Send escalations via email/SMS to compliance officers.

---

67. What if a PCI-DSS audit flags `autocomplete=on` in login page forms?

- **UI violation**.
- **Solution**:
  - Add `autocomplete=off` in all sensitive input fields.
  - Enforce CSP headers and iframe busting.
  - Add OWASP ZAP/DAST scanner in CI pipeline.

---

68. What if you store scanned cheque images in a shared S3 folder with public URL?

- **Violation**: Exposes sensitive data under RBI norms.
- **Solution**:
  - Restrict S3 access to signed URLs only.
  - Add bucket policy denying `s3:GetObject` from `aws:PrincipalType != IAM`.
  - Store images with KMS encryption and hash-based filename.

---

69. What if a transaction record is missing traceId or referenceId in audit logs?

- **Traceability failure**.
- **Solution**:
  - Enforce middleware to generate `traceId` per request (OpenTelemetry or custom UUID).
  - Validate log completeness in pre-shutdown hook.

- Block commit if audit trace is not generated.

---

70. What if customer data is backed up on physical tape drive for DR but not encrypted?

- **Risk**: Violates RBI's IT Framework for NBFCs.
- **Solution**:
  - Mandate AES-256 or HSM encryption for any media backups.
  - Log checksum + physical access trail.
  - Auto-expire backup after retention TTL + log destruction.

Scenario	Compliance Area	Explanation / Solution
PAN/Aadhaar in Logs	Mask PII	Mask all but last 4 digits; use log scrubbing filters
Transaction Traceability	Audit Logging	Store transaction metadata with traceId in immutable storage
Data Residency (S3)	Region Policy	Block data outside India via SCP and bucket policy
Weak Hashing (SHA-1)	Password Policy	Upgrade to bcrypt/PBKDF2; dual field migration
Consent Audit	DPDP Compliance	Consent ledger with timestamp, device ID, signed doc
eKYC OTP Bypass	UIDAI Rule	OTP validation mandatory before eKYC execution
Third-party API Access	Data Scope Restriction	Restrict API scopes, partner registry enforcement
S3 Encryption	PCI-DSS 3.4	Enforce S3 KMS encryption and detect violations via AWS Config
CRUD Logging	SOC2 CC6.1	Store user-role-action logs in audit DB + encryption
Replay Protection	RBI API Guidelines	Use signed payloads + nonce cache to prevent replays
Secret Rotation	SOC2 Traceability	Use Secrets Manager + CloudTrail + auto-rotation
PII Webhooks	DPDP Violation	Mask PII in outbound payloads; log all webhook events
Loan Decision Explainability	AI Fairness	Store model inputs + version + rule-based explanation
Account Deletion vs AML	Legal Conflict	Soft-delete flagged accounts; retain under legal hold
Cross-sell without Consent	Fair Usage	Separate consent category for upselling use
AML Backlog	RBI SLA Breach	Auto-alert & track acknowledgment timestamps
Autocomplete in Login	PCI-DSS UI	Add `autocomplete=off` and secure headers
Cheque Image Exposure	Sensitive Data Leak	Signed URLs + encrypted storage + hash naming
Missing Trace ID	Audit Trail Gaps	Middleware-generated UUID per request, pre-commit check
Unencrypted Tape Backup	DR Policy	Encrypt all physical media + log retention + destruction