Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: JMSC • **Updated:** 19 Nov 2026
https://mockattempt.com/courses/java-microservices-kafka-docker-training

JMSC • JAVA ENTERPRISE BACKEND & MICROSERVICES ENGINEERING | **INTERMEDIATE TO SENIOR BACKEND ARCHITECTURE**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Microservices with Java, Spring Cloud, Kafka, Redis & Docker

Upgrade from monolithic CRUD to high-paying MNC & product-company backend engineering: design, secure, trace, cache, test, and containerize event-driven Java 21 + Spring Boot 3 microservices with Kafka and Redis.

Duration: 8 Weeks • Weekend Live (Sat–Sun • 2h/Day) (92 Hrs) • **Modules:** 8 • **Live Topics:** 139 • **Subtopics:** 1,112+ • **Labs & Tasks:** 278 • **Mocks / AI Vivas:** 139 / 139 • **Projects:** 8 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹4,499 (Net Tuition ₹3,813 + Included 18% GST ₹686) • **Split Option:** ₹2,250 Lock + 1 × ₹2,249 • **Effective ROI:** ~₹49/Guided Hr (~₹32/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 10–12 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (Java 21 LTS, Spring Cloud, Kafka & Redis): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: Java 21 & Spring Boot 3 • Microservices & Bounded Contexts • Spring Cloud Config & Eureka • Spring Cloud Gateway • OpenFeign & Resilience4j • Micrometer Distributed Tracing • Apache Kafka (Outbox & DLQ) • Saga Pattern & Idempotency • Redis 7 Caching & Locks • Docker Compose & GitHub Actions

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Decompose monoliths cleanly into Bounded Context microservices with database-per-service isolation.
2. Implement Eureka discovery, Spring Cloud Gateway routing/JWT auth, OpenFeign clients, and Resilience4j circuit breakers.
3. Build resilient event-driven workflows over Apache Kafka with partition ordering, retries, `.DLT` queues, idempotency, and Transactional Outbox.
4. Accelerate hot reads with Redis 7, verify services with Testcontainers, and run the full mesh via Docker Compose & GitHub Actions CI/CD.
5. Crack LeetCode and HackerRank problems in Java 21 LTS, Spring Cloud, Kafka & Redis with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (Java 21 & Spring Boot 3, Microservices & Bounded Contexts, Spring Cloud Config & Eureka, Spring Cloud Gateway, OpenFeign & Resilience4j) — Automated Java 21 & Spring Boot 3, Microservices & Bounded Contexts, Spring Cloud Config & Eureka, Spring Cloud Gateway, OpenFeign & Resilience4j test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Microservices with Java, Spring Cloud, Kafka, Redis & Docker topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular Java 21 & Spring Boot 3, Microservices & Bounded Contexts, Spring Cloud Config & Eureka, Spring Cloud Gateway, OpenFeign & Resilience4j production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Microservices with Java, Spring Cloud, Kafka, Redis & Docker.

Eligibility & Prerequisites: Working knowledge of Core Java 21 and Spring Boot REST APIs (or completion of SPBT). • No prior Microservices with Java, Spring Cloud, Kafka, Redis & Docker industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (8 MODULES • 139 LIVE TOPICS • 1,112+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Service Mesh, API Gateway, Resilience4j & Apache Kafka (Weeks 1–4) (69 Topics / 552 Subtopics) • Modules 5–8: Month 2 – Redis Caching, Testcontainers, Docker Compose & Distributed Capstone (Weeks 5–8) (70 Topics / 560 Subtopics)

MODULE 1 OF 8 • STAGE 1 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Production REST Contracts & Stateless JWT Security Foundations

Secure Spring Boot 3 REST APIs end-to-end using Spring Security 6 — Authentication vs Authorization, SecurityFilterChain lambda DSL, BCrypt password hashing, UserDetails & UserDetailsService, AuthenticationManager, RBAC, JWT header.payload.signature internals, stateless OncePerRequestFilter validation, CORS, CSRF trade-offs, OAuth2 Resource Server concepts, @PreAuthorize method security, security testing, and Major Project 4.

Objectives: ✓ Trace how requests pass through the Spring Security 6 `SecurityFilterChain` before reaching `DispatcherServlet`. • ✓ Hash passwords irreversibly with `BCryptPasswordEncoder` and load database identities via `UserDetailsService`. • ✓ Implement stateless HMAC-SHA256 / RSA JWT issuance and `OncePerRequestFilter` validation populating `SecurityContextHolder`. • ✓ Enforce Role-Based and Permission-Based Access Control (`ADMIN`, `SELLER`, `CUSTOMER`) at both URL and `@PreAuthorize` method levels. • ✓ Configure strict CORS origins, stateless CSRF policy, custom `401`/`403` JSON entry points, `@WithMockUser` security tests, and deliver Major Project 4 (Secure E-Commerce Backend API).

1.1 Security Fundamentals (Authentication vs Authorization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Principal identity verification (AuthN -> 401 when missing/invalid) vs privilege verification (AuthZ -> 403 when insufficient) • Defense-in-Depth • OWASP API Top 10 (BOLA/IDOR!) • Production Pitfall Guard (Security Fundamentals (Authentication vs Authorization)) • Hands-On Verification (Security Fundamentals (Authentication vs Authorization)) • Security Fundamentals (Authentication vs Authorization) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Security Fundamentals (Authentication vs Authorizati...

Purpose: Preventing Broken Object Level Authorization (BOLA) by checking `order.getUserId().equals(currentUser.id())` in addition to role checks.

What Topic Is Next → Next Topic 1.2: Spring Security 6 Filter Chain Architecture

1.2 Spring Security 6 Filter Chain Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DelegatingFilterProxy` -> `FilterChainProxy` -> ordered `SecurityFilterChain` filters -> `SecurityContextHolder` -> `DispatcherServlet` • Production Pitfall Guard (Spring Security 6 Filter Chain Architecture) • Hands-On Verification (Spring Security 6 Filter Chain Architecture) • Spring Security 6 Filter Chain Architecture — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Security 6 Filter Chain Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding why `@RestControllerAdvice` does NOT catch exceptions thrown inside a Security Filter unless delegated via `AuthenticationEntryPoint` / `AccessDeniedHandler`...

What Topic Is Next → Next Topic 1.3: SecurityFilterChain Configuration (Lambda DSL)

1.3 SecurityFilterChain Configuration (Lambda DSL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern Spring Security 6 `@Bean SecurityFilterChain` lambda DSL (`authorizeHttpRequests` • `requestMatchers` • `sessionManagement` • `exceptionHandling`) • Production Pitfall Guard (SecurityFilterChain Configuration (Lambda DSL)) • Hands-On Verification (SecurityFilterChain Configuration (Lambda DSL)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for SecurityFilterChain Configuration (Lambda DSL)

Purpose: Allowing public `GET /api/v1/products/**` and `POST /api/v1/auth/**` while requiring authentication on everything else (`anyRequest().authenticated()`).

What Topic Is Next → Next Topic 1.4: Password Security (PasswordEncoder and BCrypt)

1.4 Password Security (PasswordEncoder and BCrypt) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why MD5/SHA-256 are broken for passwords • Adaptive salted work-factor hashing with `BCryptPasswordEncoder(12)` • `encode()` on registration vs `matches()` on login • Production Pitfall Guard (Password Security (PasswordEncoder and BCrypt)) • Hands-On Verification (Password Security (PasswordEncoder and BCrypt)) • Password Security (PasswordEncoder and BCrypt) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Password Security (PasswordEncoder and BCrypt)

Purpose: Guaranteeing every stored password hash has a unique per-user cryptographic salt automatically embedded in the `$2a$12$...` string.

What Topic Is Next → Next Topic 1.5: UserDetails Identity Model

1.5 UserDetails Identity Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing/adapting `UserDetails` (`getUsername` • `getPassword` • `getAuthorities` • `isEnabled` • Production Pitfall Guard (UserDetails Identity Model) • Hands-On Verification (UserDetails Identity Model) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for UserDetails Identity Model

Purpose: Creating a clean `AuthenticatedUser` principal record wrapping `userId`, `email`, and `GrantedAuthority` roles without coupling JPA `@Entity` directly to Spring Security.

What Topic Is Next → Next Topic 1.6: UserDetailsService Implementation

1.6 UserDetailsService Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading user credentials & roles from PostgreSQL via `UserRepository.findByEmailIgnoreCase` • Throwing `UsernameNotFoundException` without user-enumeration leaks • Production Pitfall Guard (UserDetailsService Implementation) • Hands-On Verification (UserDetailsService Implementation) • UserDetailsService Implementation — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for UserDetailsService Implementation • Production

Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Bridging PostgreSQL `users` + `roles` tables into Spring Security's authentication provider.

What Topic Is Next → Next Topic 1.7: AuthenticationManager and DaoAuthenticationProvider Workflow

1.7 AuthenticationManager and DaoAuthenticationProvider Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AuthenticationConfiguration.getAuthenticationManager()` • `UsernamePasswordAuthenticationToken` • Credential verification • Disabled/locked account checks • Production Pitfall Guard (AuthenticationManager and DaoAuthenticationProvider Workflow) • Hands-On Verification (AuthenticationManager and DaoAuthenticationProvider Workflow) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for AuthenticationManager

Purpose: Authenticating `/api/v1/auth/login` cleanly via `authenticationManager.authenticate(...)` before minting a JWT.

What Topic Is Next → Next Topic 1.8: Role-Based and Permission-Based Authorization (RBAC)

1.8 Role-Based and Permission-Based Authorization (RBAC) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Coarse roles (`ROLE_ADMIN` • `ROLE_CUSTOMER`) vs fine-grained authorities (`product:write` • `order:refund`) • `@AuthenticationPrincipal` in controllers • Production Pitfall Guard (Role-Based and Permission-Based Authorization (RBAC)) • Hands-On Verification (Role-Based and Permission-Based Authorization (RBAC)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Role-Based

Purpose: Extracting the logged-in user's ID directly from `@AuthenticationPrincipal AppUserPrincipal principal` — NEVER trusting a `userId` sent in client JSON!

What Topic Is Next → Next Topic 1.9: JWT Structure and Claims (Header, Payload, Signature)

1.9 JWT Structure and Claims (Header, Payload, Signature) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base64Url `header.payload.signature` • Standard claims (`sub` • `iss` • `iat` • Production Pitfall Guard (JWT Structure and Claims (Header, Payload, Signature)) • Hands-On Verification (JWT Structure and Claims (Header, Payload, Signature)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for JWT Structure

Purpose: Stateless horizontal scaling where any microservice replica verifies the cryptographic signature without sticky sessions.

What Topic Is Next → Next Topic 1.10: JWT Creation and Cryptographic Validation

1.10 JWT Creation and Cryptographic Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 256-bit+ HMAC-SHA256 (`HS256`) / RSA (`RS256` in microservices) signing with `jjwt` or `spring-boot-starter-oauth2-resource-server` `NimbusJwtEncoder`/`Decoder` • Strict algorithm/expiry/issuer checks • Refresh-token concept • Production Pitfall Guard (JWT Creation and Cryptographic Validation) • Hands-On Verification (JWT Creation and Cryptographic Validation) • JWT Creation and Cryptographic Validation — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for JWT Creation

Purpose: Issuing short-lived (15–30 min) signed access tokens and rejecting expired, tampered, or `alg: none` tokens.

What Topic Is Next → Next Topic 1.11: Custom JWT OncePerRequestFilter

1.11 Custom JWT OncePerRequestFilter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Extending `OncePerRequestFilter` • Extracting `Authorization: Bearer <token>` • Validating token • Setting `UsernamePasswordAuthenticationToken` on `SecurityContextHolder` • Production Pitfall Guard (Custom JWT OncePerRequestFilter) • Hands-On Verification (Custom JWT OncePerRequestFilter) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Custom JWT OncePerRequestFilter

Purpose: Authenticating Bearer requests cleanly once per HTTP dispatch without redundant filter execution on error forwards.

What Topic Is Next → Next Topic 1.12: Stateless Session Configuration

1.12 Stateless Session Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SessionCreationPolicy.STATELESS` • Ensuring zero `JSESSIONID` cookies or server-side `HttpSession` memory growth • Production Pitfall Guard (Stateless Session Configuration) • Hands-On Verification (Stateless Session Configuration) • Stateless Session Configuration — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Stateless Session Configuration • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running 5 stateless Spring Boot containers behind a load balancer where any container can serve any request.

What Topic Is Next → Next Topic 1.13: CORS (Cross-Origin Resource Sharing)

1.13 CORS (Cross-Origin Resource Sharing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Browser Same-Origin Policy • `OPTIONS` preflight requests • Configuring explicit `CorsConfigurationSource` (`allowedOrigins` from env config • `allowedMethods` • Production Pitfall Guard (CORS (Cross-Origin Resource Sharing)) • Hands-On Verification (CORS (Cross-Origin Resource Sharing)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for CORS (Cross-Origin Resource Sharing)

Purpose: Allowing your authorized web frontend domain to call `/api/v1/**` with `Authorization: Bearer` headers cleanly.

What Topic Is Next → Next Topic 1.14: CSRF Protection and Stateless API Trade-offs

1.14 CSRF Protection and Stateless API Trade-offs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Cross-Site Request Forgery exploits ambient browser cookies • Why cookie-based web apps MUST keep CSRF enabled vs why pure `Authorization: Bearer` header APIs disable CSRF • Production Pitfall Guard (CSRF Protection and Stateless API Trade-offs) • Hands-On Verification (CSRF Protection and Stateless API Trade-offs) • CSRF Protection and Stateless API Trade-offs — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for CSRF Protection • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Explaining accurately in security interviews *when* disabling CSRF is safe (pure `Bearer` header auth, zero auth cookies) and *when* it is vulnerable (cookie-stored JWTs!...

What Topic Is Next → Module 1 Capstone (Week 1 Practical Build: Production REST Contracts & Stateless JWT Security Foundations) → Next Topic 2.1: Monolithic Architecture (Strengths, Modularity and Scaling Limits)

Module 1 Hands-On Capstone Project (100 Marks): Week 1 Practical Build: Production REST Contracts & Stateless JWT Security Foundations — Implement and verify a clean, runnable JMSC repository covering Secure Spring Boot 3 REST APIs end-to-end using Spring Security 6 — Authentication vs Authorization, SecurityFilterChain lambd...

MODULE 2 OF 8 • STAGE 1 • 12 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Microservices Architecture Fundamentals

Transition from modular Spring Boot monoliths to distributed Microservices — architectural trade-offs, Domain-Driven Design Bounded Contexts, database-per-service ownership, synchronous HTTP clients (OpenFeign, RestClient/RestTemplate, WebClient), Netflix Eureka service discovery, Spring Cloud Gateway, client-side load balancing, Spring Cloud Config, and distributed correlation IDs.

Objectives: ✓ Evaluate honestly when a modular monolith is better vs when independent Microservices are justified. • ✓ Decompose a commerce domain into clean Bounded Contexts (`user`, `product`, `order`, `inventory`, `payment`, `notification`) with strict database-per-service boundaries. • ✓ Implement synchronous service-to-service calls using declarative `OpenFeign` and modern `RestClient`/`WebClient` with explicit timeouts. • ✓ Register dynamic service instances with Eureka, route traffic through Spring Cloud Gateway, and externalize configs via Spring Cloud Config. • ✓ Propagate `X-Correlation-Id` / trace headers across service hops so distributed requests remain debuggable.

2.1 Monolithic Architecture (Strengths, Modularity and Scaling Limits) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single deployable artifact • Local ACID transactions • Low operational overhead vs deployment contention • Blast radius • Production Pitfall Guard (Monolithic Architecture (Strengths, Modularity and Scaling Limits)) • Hands-On Verification (Monolithic Architecture (Strengths, Modularity and Scaling Limits)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Monolithic Architecture (Strengths, Modularity and S...

Purpose: Understanding why well-engineered startups begin as a clean modular monolith before extracting high-scale microservices.

What Topic Is Next → Next Topic 2.2: Microservices Architecture (Benefits and Distributed Challenges)

2.2 Microservices Architecture (Benefits and Distributed Challenges) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Independent deployability • Fault isolation • Per-service scaling • Database-per-service autonomy vs network fallacies • Production Pitfall Guard (Microservices Architecture (Benefits and Distributed Challenges)) • Hands-On Verification (Microservices Architecture (Benefits and Distributed Challenges)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Microservices Architecture (Benefits and Distributed...

Purpose: Scaling `product-catalog-service` to 10 read replicas during a flash sale while keeping `billing-service` at 2 replicas.

What Topic Is Next → Next Topic 2.3: Service Decomposition Strategies

2.3 Service Decomposition Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decomposing by business capability • Subdomain rather than technical layers • Sizing boundaries around transactional invariants • Production Pitfall Guard (Service Decomposition Strategies) • Hands-On Verification (Service Decomposition Strategies) • Service Decomposition Strategies — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Service Decomposition Strategies

Purpose: Splitting `Catalog`, `Order`, `Inventory`, `Payment`, and `Notification` so each owns a clear business lifecycle.

What Topic Is Next → Next Topic 2.4: Bounded Context and DDD Fundamentals

2.4 Bounded Context and DDD Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ubiquitous Language • How `"Product"` means title/description/SEO in `Catalog` Context vs SKU/warehouse-bin/qty in `Inventory` Context • Anti-Corruption Layer • Production Pitfall Guard (Bounded Context and DDD Fundamentals) • Hands-On Verification (Bounded Context and DDD Fundamentals) • Bounded Context and DDD Fundamentals — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Bounded Context

Purpose: Avoiding one bloated 80-field `GodProduct` class shared across every microservice.

What Topic Is Next → Next Topic 2.5: Service Communication (Synchronous vs Asynchronous)

2.5 Service Communication (Synchronous vs Asynchronous) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Request-response HTTP/REST (immediate query/validation) vs event-driven Kafka messaging (temporal decoupling • Resilience when downstream is offline) • Production Pitfall Guard (Service Communication (Synchronous vs Asynchronous)) • Hands-On Verification (Service Communication (Synchronous vs Asynchronous)) • Service Communication (Synchronous vs Asynchronous) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Service Communication (Synchronous vs Asynchronous) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Validating SKU price synchronously while triggering email/analytics notifications asynchronously via Kafka.

What Topic Is Next → Next Topic 2.6: Service-to-Service REST Communication

2.6 Service-to-Service REST Communication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contract DTOs • Connection & read timeouts (NEVER leave infinite defaults!) • Propagating auth/correlation headers • Handling downstream `4xx` vs `5xx` • Production Pitfall Guard (Service-to-Service REST Communication) • Hands-On Verification (Service-to-Service REST Communication) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production

Runtime Execution & Memory Lifecycle for Service-to-Service REST Communication

Purpose: `order-service` verifying product pricing & active status from `product-service` with a 2-second read timeout.

What Topic Is Next → Next Topic 2.7: Spring Cloud OpenFeign (Declarative REST Clients)

2.7 Spring Cloud OpenFeign (Declarative REST Clients) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableFeignClients` • `@FeignClient(name = "product-service")` • Declarative Spring MVC annotations • `ErrorDecoder` • Production Pitfall Guard (Spring Cloud OpenFeign (Declarative REST Clients)) • Hands-On Verification (Spring Cloud OpenFeign (Declarative REST Clients)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud OpenFeign (Declarative REST Clients)

Purpose: Calling `productClient.getProductBySku(sku)` as a clean, type-safe Java interface integrated with Eureka load balancing.

What Topic Is Next → Next Topic 2.8: RestTemplate Legacy Awareness & Modern Spring RestClient

2.8 RestTemplate Legacy Awareness & Modern Spring RestClient [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Maintaining `RestTemplate` in existing codebases vs migrating to Spring 6.1+ fluent synchronous `RestClient` • Production Pitfall Guard (RestTemplate Legacy Awareness & Modern Spring RestClient) • Hands-On Verification (RestTemplate Legacy Awareness & Modern Spring RestClient) • RestTemplate Legacy Awareness & Modern Spring RestClient — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for RestTemplate Legacy Awareness • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Reading brownfield `RestTemplate` code confidently while writing clean fluent `RestClient` calls in new Spring Boot 3.2+ apps.

What Topic Is Next → Next Topic 2.9: WebClient (Non-Blocking & Reactive HTTP Client)

2.9 WebClient (Non-Blocking & Reactive HTTP Client) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WebClient.builder()` • `Mono`/`Flux` basics • Parallel scatter-gather fan-out calls • Timeout & error status handlers (`onStatus`) • Production Pitfall Guard (WebClient (Non-Blocking & Reactive HTTP Client)) • Hands-On Verification (WebClient (Non-Blocking & Reactive HTTP Client)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for WebClient (Non-Blocking & Reactive HTTP Client)

Purpose: Firing concurrent non-blocking HTTP requests to multiple downstream services and combining responses cleanly.

What Topic Is Next → Next Topic 2.10: Service Discovery Concepts

2.10 Service Discovery Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why hardcoded `localhost:8082` breaks in autoscaled container environments • Client-side vs server-side service discovery • Heartbeats & eviction • Production Pitfall Guard (Service Discovery Concepts) • Hands-On Verification (Service Discovery Concepts) • Service Discovery Concepts — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Service Discovery Concepts

Purpose: Resolving logical service names (`lb://inventory-service`) dynamically across scaling instances.

What Topic Is Next → Next Topic 2.11: Netflix Eureka Service Registry and Discovery

2.11 Netflix Eureka Service Registry and Discovery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Eureka Server registry dashboard (`8761`) • Instance registration • 30s heartbeats • Self-preservation mode • Production Pitfall Guard (Netflix Eureka Service Registry and Discovery) • Hands-On Verification (Netflix Eureka Service Registry and Discovery) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Netflix Eureka Service Registry

Purpose: Running a local & staging Eureka discovery registry where all microservices register and discover peers automatically.

What Topic Is Next → Next Topic 2.12: API Gateway Responsibilities

2.12 API Gateway Responsibilities [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single public edge entry point • Path routing • Centralized JWT verification • Rate limiting • Production Pitfall Guard (API Gateway Responsibilities) • Hands-On Verification (API Gateway Responsibilities) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for API Gateway Responsibilities

Purpose: Exposing only `api-gateway:8080` to external clients while keeping internal microservices on private Docker networks.

What Topic Is Next → Next Topic 2.13: Spring Cloud Gateway Implementation

2.13 Spring Cloud Gateway Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-cloud-starter-gateway` • Route Predicates (`Path`) • GatewayFilters (`AddRequestHeader` • `StripPrefix` • Production Pitfall Guard (Spring Cloud Gateway Implementation) • Hands-On Verification (Spring Cloud Gateway Implementation) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud Gateway Implementation

Purpose: Configuring declarative YAML routes forwarding `/api/v1/orders/**` to `lb://order-service`.

What Topic Is Next → Next Topic 2.14: Client-Side Load Balancing

2.14 Client-Side Load Balancing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Cloud LoadBalancer (`@LoadBalanced`) • Round-Robin instance selection across multiple running ports (`--server.port=0`) • Production Pitfall Guard (Client-Side Load Balancing) • Hands-On Verification (Client-Side Load Balancing) • Client-Side Load Balancing — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Client-Side Load Balancing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Distributing `order-service` traffic evenly across two running `product-service` instances and skipping unhealthy ones.

What Topic Is Next → Next Topic 2.15: Centralized Configuration Management Concepts

2.15 Centralized Configuration Management Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why managing 12 separate `application.yml` files across 6 services x 3 environments drifts without centralized config • Production Pitfall Guard (Centralized Configuration Management Concepts) • Hands-On Verification (Centralized Configuration Management Concepts) • Centralized Configuration Management Concepts — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Centralized Configuration Management Concepts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Storing version-controlled service configuration in a central Git-backed config repository.

What Topic Is Next → Next Topic 2.16: Spring Cloud Config Server & Client

2.16 Spring Cloud Config Server & Client [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableConfigServer` • `spring.config.import=optional:configserver:http://localhost:8888` • Actuator `/refresh` awareness • Production Pitfall Guard (Spring Cloud Config Server & Client) • Hands-On Verification (Spring Cloud Config Server & Client) • Spring Cloud Config Server & Client — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud Config Server

Purpose: Serving environment-specific configuration from `config-server` to all microservices at startup.

What Topic Is Next → Next Topic 2.17: Distributed Tracing Concepts

2.17 Distributed Tracing Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Trace ID (spans entire multi-service request) vs Span ID (single hop/operation) • W3C `traceparent` header propagation • Production Pitfall Guard (Distributed Tracing Concepts) • Hands-On Verification (Distributed Tracing Concepts) • Distributed Tracing Concepts — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Distributed Tracing Concepts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Reconstructing exact millisecond timelines across `Gateway -> Order -> Inventory -> PostgreSQL` when debugging latency.

What Topic Is Next → Next Topic 2.18: Correlation ID Request Tracking

2.18 Correlation ID Request Tracking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Gateway GlobalFilter generating `X-Correlation-Id` if absent • Propagating via Feign `RequestInterceptor` • SLF4J MDC into every log line & error response • Production Pitfall Guard (Correlation ID Request Tracking) • Hands-On Verification (Correlation ID Request Tracking) • Correlation ID Request Tracking — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Correlation ID Request Tracking

Purpose: Giving support engineers a `correlationId` in API error responses that maps directly to every microservice's logs.

What Topic Is Next → Module 2 Capstone (Week 17 Lab Build: Eureka + Spring Cloud Gateway + OpenFeign Service Mesh) → Next Topic 3.1: Spring Cloud Ecosystem and Release Train Compatibility

Module 2 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Eureka + Spring Cloud Gateway + OpenFeign Service Mesh — Extract `product-service` and `order-service` behind `api-gateway` and `discovery-server` (Eureka) with declarative `OpenFeign` calls, Spring Cloud Config externalization, and `X-Correlation...

MODULE 3 OF 8 • STAGE 1 • 12 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Spring Cloud and Distributed Systems

Engineer fault-tolerant distributed Java backends — Eureka Server/Client hardening, Gateway JWT edge security, Spring Cloud LoadBalancer, Resilience4j Circuit Breakers, Retries with exponential backoff, Rate Limiters, Timeouts & Fallbacks, Distributed Transactions & Saga Pattern (Choreography vs Orchestration), Eventual Consistency, Idempotency Keys, Micrometer Tracing, and Observability.

Objectives: ✓ Secure the edge `api-gateway` with JWT authentication and forward verified user identity headers to internal services. • ✓ Prevent cascading microservice outages using `Resilience4j` Circuit Breakers, TimeLimiters, bounded Retries, RateLimiters, and graceful Fallbacks. • ✓ Explain why 2-Phase Commit (2PC) fails to scale across microservices and design Saga Choreography/Orchestration workflows with compensating actions. • ✓ Design idempotent `POST` / payment / consumer operations using `Idempotency-Key` tables so retries never double-charge. • ✓ Instrument logs, metrics (`Actuator` + `Micrometer`), and distributed traces (`Micrometer Tracing` + Zipkin) for production observability.

3.1 Spring Cloud Ecosystem and Release Train Compatibility [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Cloud 2023/2024 release trains compatible with Spring Boot 3.x • BOM alignment (`spring-cloud-dependencies`) • Key cloud-native starters • Production Pitfall Guard (Spring Cloud Ecosystem and Release Train Compatibility) • Hands-On Verification (Spring Cloud Ecosystem and Release Train Compatibility) • Spring Cloud Ecosystem and Release Train Compatibility — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud Ecosystem

Purpose: Importing `spring-cloud-dependencies` BOM so Gateway, Eureka, OpenFeign, Config, and CircuitBreaker versions always match Boot 3.x.

What Topic Is Next → Next Topic 3.2: Eureka Server Registry Setup and Health Tuning

3.2 Eureka Server Registry Setup and Health Tuning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standalone & peer-aware Eureka Server config • Actuator health endpoints • Lease expiration & renewal intervals in dev vs prod • Production Pitfall Guard (Eureka Server Registry Setup and Health Tuning) • Hands-On Verification (Eureka Server Registry Setup and Health Tuning) • Eureka Server Registry Setup and Health Tuning — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Eureka Server Registry Setup

Purpose: Keeping a clean service registry health view across all running Major Project 5 services.

What Topic Is Next → Next Topic 3.3: Eureka Client Registration and Metadata

3.3 Eureka Client Registration and Metadata [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring.application.name` (critical service ID!) • `prefer-ip-address: true` in Docker containers • Graceful deregistration on shutdown • Production Pitfall Guard (Eureka Client Registration and Metadata) • Hands-On Verification (Eureka Client Registration and Metadata) • Eureka Client Registration and Metadata — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Eureka Client Registration

Purpose: Setting `prefer-ip-address: true` so Docker containers advertise reachable container IPs rather than unresolvable random container hostnames.

What Topic Is Next → Next Topic 3.4: Spring Cloud Gateway Routing and Custom Filters

3.4 Spring Cloud Gateway Routing and Custom Filters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Path/Method/Header predicates • `RewritePath` • Custom `GlobalFilter` & `GatewayFilterFactory` for request logging • Header sanitization • Production Pitfall Guard (Spring Cloud Gateway Routing and Custom Filters) • Hands-On Verification (Spring Cloud Gateway Routing and Custom Filters) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud Gateway Routing

Purpose: Stripping spoofed client-supplied `X-User-Id` headers at the gateway before injecting verified JWT claims!

What Topic Is Next → Next Topic 3.5: API Gateway Security (Edge JWT Authentication)

3.5 API Gateway Security (Edge JWT Authentication) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating Bearer JWTs at Spring Cloud Gateway (`SecurityWebFilterChain`) • Propagating verified principal claims/tokens downstream • Production Pitfall Guard (API Gateway Security (Edge JWT Authentication)) • Hands-On Verification (API Gateway Security (Edge JWT Authentication)) • API Gateway Security (Edge JWT Authentication) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for API Gateway Security (Edge JWT Authentication) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Rejecting unauthenticated or expired tokens right at the perimeter before internal services spend CPU or DB connections.

What Topic Is Next → Next Topic 3.6: Spring Cloud LoadBalancer Customization

3.6 Spring Cloud LoadBalancer Customization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Round-Robin vs Random load balancing • Health-check-aware instance filtering • Caching discovered instance lists • Production Pitfall Guard (Spring Cloud LoadBalancer Customization) • Hands-On Verification (Spring Cloud LoadBalancer Customization) • Spring Cloud LoadBalancer Customization — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cloud LoadBalancer Customization

Purpose: Routing inter-service calls only to `UP` instances without hitting crashed containers.

What Topic Is Next → Next Topic 3.7: Circuit Breaker Pattern (Preventing Cascading Failures)

3.7 Circuit Breaker Pattern (Preventing Cascading Failures) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why a slow downstream service takes down the entire microservice fleet via thread-pool saturation • `CLOSED` -> `OPEN` -> `HALF_OPEN` state machine • Production Pitfall Guard (Circuit Breaker Pattern (Preventing Cascading Failures)) • Hands-On Verification (Circuit Breaker Pattern (Preventing Cascading Failures)) • Circuit Breaker Pattern (Preventing Cascading Failures) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Circuit Breaker Pattern (Preventing Cascading Failur... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Failing fast in < 1ms when `payment-gateway` is down (`OPEN` state) instead of tying up 200 Tomcat threads waiting on timeouts.

What Topic Is Next → Next Topic 3.8: Resilience4j CircuitBreaker Implementation

3.8 Resilience4j CircuitBreaker Implementation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-cloud-starter-circuitbreaker-resilience4j` • `@CircuitBreaker(name = "inventory" • FallbackMethod = "...")` • `slidingWindowSize` • Production Pitfall Guard (Resilience4j CircuitBreaker Implementation) • Hands-On Verification (Resilience4j CircuitBreaker Implementation) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Resilience4j CircuitBreaker Implementation

Purpose: Wrapping `inventoryClient` calls in `order-service` with Resilience4j and monitoring live circuit state via Actuator.

What Topic Is Next → Next Topic 3.9: Retry Policies and Exponential Backoff with Jitter

3.9 Retry Policies and Exponential Backoff with Jitter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Retry` configuration • MaxAttempts • Exponential backoff + random jitter (preventing retry storms/thundering herd!) • Retrying ONLY transient idempotent operations • Production Pitfall Guard (Retry Policies and Exponential Backoff with Jitter) • Hands-On Verification (Retry Policies and Exponential Backoff with Jitter) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Retry Policies

Purpose: Retrying a transient network `SocketTimeoutException` 3 times (`200ms -> 400ms -> 800ms` + jitter) safely.

What Topic Is Next → Next Topic 3.10: Rate Limiting and Request Throttling

3.10 Rate Limiting and Request Throttling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Protecting login/OTP/checkout APIs from abuse • Traffic spikes using Resilience4j `@RateLimiter` (`limitForPeriod` • `limitRefreshPeriod`) -> HTTP `429 Too Many Requests` • Production Pitfall Guard (Rate Limiting and Request Throttling) • Hands-On Verification (Rate Limiting and Request Throttling) • Rate Limiting and Request Throttling — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Rate Limiting

Purpose: Throttling `/api/v1/auth/login` and `/api/v1/orders` bursts cleanly with `429` + `Retry-After` header.

What Topic Is Next → Next Topic 3.11: Timeouts and Bulkhead Isolation

3.11 Timeouts and Bulkhead Isolation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Socket read timeouts • Resilience4j `@TimeLimiter` / `@Bulkhead` (isolating thread/semaphore pools per downstream dependency so one slow partner cannot starve healthy endpoints) • Production Pitfall Guard (Timeouts and Bulkhead Isolation) • Hands-On Verification (Timeouts and Bulkhead Isolation) •

Timeouts and Bulkhead Isolation — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Timeouts • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Capping concurrent outbound calls to a slow shipping provider via `@Bulkhead` while catalog browsing stays 100% responsive.

What Topic Is Next → Next Topic 3.12: Graceful Fallback Strategies

3.12 Graceful Fallback Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cached stale read fallback • Default degraded response • Queuing order as `PENDING_VERIFICATION` vs explicit `503 Service Unavailable` when safety requires • Production Pitfall Guard (Graceful Fallback Strategies) • Hands-On Verification (Graceful Fallback Strategies) • Graceful Fallback Strategies — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Graceful Fallback Strategies

Purpose: Serving cached product recommendations when the recommendation service is down, while refusing unsafe blind financial debits!

What Topic Is Next → Next Topic 3.13: Distributed Transactions Problem Space

3.13 Distributed Transactions Problem Space [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why classic XA / Two-Phase Commit (2PC) locks resources across services • Kills availability (CAP theorem trade-offs) in modern cloud microservices • Production Pitfall Guard (Distributed Transactions Problem Space) • Hands-On Verification (Distributed Transactions Problem Space) • Distributed Transactions Problem Space — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Distributed Transactions Problem Space • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Coordinating `Order Creation` -> `Inventory Reservation` -> `Payment Capture` across 3 independent PostgreSQL databases.

What Topic Is Next → Next Topic 3.14: Saga Pattern (Choreography vs Orchestration)

3.14 Saga Pattern (Choreography vs Orchestration) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Local ACID transactions per service + compensating transactions (`ReleaseStock` if `PaymentFailed` • `CancelOrder`) • Event Choreography (peer events) vs Saga Orchestrator • Production Pitfall Guard (Saga Pattern (Choreography vs Orchestration)) • Hands-On Verification (Saga Pattern (Choreography vs Orchestration)) • Saga Pattern (Choreography vs Orchestration) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Saga Pattern (Choreography vs Orchestration)

Purpose: Executing `PENDING -> STOCK_RESERVED -> PAID -> CONFIRMED` (or compensating `STOCK_RELEASED -> CANCELLED` on payment failure!).

What Topic Is Next → Next Topic 3.15: Eventual Consistency in Distributed Systems

3.15 Eventual Consistency in Distributed Systems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strong consistency within an aggregate vs eventual consistency across Bounded Contexts • Intermediate lifecycle states (`PENDING_PAYMENT`) • UI/API polling & webhooks • Production Pitfall Guard (Eventual Consistency in Distributed Systems) • Hands-On Verification (Eventual Consistency in Distributed Systems) • Eventual Consistency in Distributed Systems — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Eventual Consistency in Distributed Systems

Purpose: Returning `202 Accepted` / `status: PENDING` immediately on order submission while async Saga steps converge within milliseconds.

What Topic Is Next → Next Topic 3.16: Idempotency in Distributed APIs

3.16 Idempotency in Distributed APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Idempotency-Key` UUID header • `idempotency_keys` PostgreSQL unique constraint table storing `(key • User_id • Request_hash • Production Pitfall Guard (Idempotency in Distributed APIs) • Hands-On Verification (Idempotency in Distributed APIs) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Idempotency in Distributed APIs

Purpose: Guaranteeing that if a mobile app retries `POST /api/v1/orders` 3 times on flaky 4G, only ONE order and ONE payment are created!

What Topic Is Next → Next Topic 3.17: Centralized Config Server Operations

3.17 Centralized Config Server Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Environment profiles in Config Server • Health checks • Fail-fast vs optional config startup • Encrypting sensitive config properties • Production Pitfall Guard (Centralized Config Server Operations) • Hands-On Verification (Centralized Config Server Operations) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Centralized Config Server Operations

Purpose: Managing shared Kafka broker addresses, Eureka URLs, and Resilience4j thresholds cleanly across 6 services.

What Topic Is Next → Next Topic 3.18: Distributed Tracing with Micrometer Tracing

3.18 Distributed Tracing with Micrometer Tracing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spring Boot 3 `micrometer-tracing-bridge-brave` + `zipkin-reporter-brave` (replaces deprecated Spring Cloud Sleuth!) • Automatic `[service • TraceId • SpanId]` log pattern • Production Pitfall Guard (Distributed Tracing with Micrometer Tracing) • Hands-On Verification (Distributed Tracing with Micrometer Tracing) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Distributed Tracing with Micrometer Tracing

Purpose: Visualizing end-to-end request waterfall spans across Gateway, Order, Product, and Inventory in Zipkin.

What Topic Is Next → Next Topic 3.19: Three Pillars of Observability (Logs, Metrics and Traces)

3.19 Three Pillars of Observability (Logs, Metrics and Traces) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Structured JSON logs (what happened) • Actuator + Micrometer Prometheus metrics (`/actuator/health` • `/actuator/metrics` • JVM/HikariCP/HTTP p95 latency) • Production Pitfall Guard (Three Pillars of Observability (Logs, Metrics and Traces)) • Hands-On Verification (Three Pillars of Observability (Logs, Metrics and Traces)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Three Pillars of Observability (Logs, Metrics and Tr...

Purpose: Monitoring `http.server.requests` p95 latency, HikariCP active connections, and custom `orders.placed` counters via Micrometer `MeterRegistry`.

What Topic Is Next → Module 3 Capstone (Week 18 Lab Build: Fault-Tolerant Resilience4j & Idempotent Saga Checkout) → Next Topic 4.1: Messaging Fundamentals and Event-Driven Architecture

Module 3 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: Fault-Tolerant Resilience4j & Idempotent Saga Checkout — Harden `order-service` -> `inventory-service` -> `payment-service` interactions using Resilience4j CircuitBreakers, exponential-backoff Retries, RateLimiters, `Idempotency-Key` deduplication...

MODULE 4 OF 8 • STAGE 1 • 12 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Apache Kafka and Asynchronous Communication

Build high-throughput, fault-tolerant event-driven Java backends with Apache Kafka and Spring for Apache Kafka — brokers, topics, partitions, keys, producers (`KafkaTemplate`), consumers (`@KafkaListener`), consumer groups, offsets & commits, replication/acks, JSON/Avro serialization, retry topics, Dead Letter Queues (DLQ), ordering guarantees, idempotent consumers, and the Transactional Outbox Pattern.

Objectives: ✓ Contrast traditional point-to-point queues with Apache Kafka's partitioned, append-only distributed commit log. • ✓ Publish keyed domain events reliably using Spring `KafkaTemplate` with `acks=all` and idempotent producer settings. • ✓ Scale parallel consumption using Consumer Groups while understanding partition-to-consumer cardinality and offset commits. • ✓ Handle poison-pill messages cleanly with `DefaultErrorHandler` / `@RetryableTopic` backoff retries and Dead Letter Topics (`.DLT`). • ✓ Guarantee per-entity ordering via partition keys, prevent duplicate side effects via idempotent `processed_events` tables, and avoid dual-write bugs via the Transactional Outbox Pattern.

4.1 Messaging Fundamentals and Event-Driven Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Producers • Brokers • Consumers • Point-to-point queues vs pub/sub event streams • Production Pitfall Guard (Messaging Fundamentals and Event-Driven Architecture) • Hands-On Verification (Messaging Fundamentals and Event-Driven Architecture) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Messaging Fundamentals

Purpose: Allowing `order-service` to accept orders in 15ms even while `notification-service` (SMTP email) experiences temporary slowness.

What Topic Is Next → Next Topic 4.2: Synchronous REST vs Asynchronous Kafka Trade-offs

4.2 Synchronous REST vs Asynchronous Kafka Trade-offs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Latency decoupling • Spike buffering/backpressure • Multi-subscriber fan-out vs debugging complexity • Eventual consistency • Production Pitfall Guard (Synchronous REST vs Asynchronous Kafka Trade-offs) • Hands-On Verification (Synchronous REST vs Asynchronous Kafka Trade-offs) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Synchronous REST vs Asynchronous Kafka Trade-offs

Purpose: Having `inventory-service`, `notification-service`, and `analytics-service` all react independently to one `order.placed` topic without changing `order-service`!

What Topic Is Next → Next Topic 4.3: Kafka Architecture (Brokers, Topics, Partitions and KRaft)

4.3 Kafka Architecture (Brokers, Topics, Partitions and KRaft) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable append-only partition logs • Partition keys (`hash(key) % numPartitions`) • Retention by time/size (messages are NOT deleted on read!) • Modern KRaft metadata mode • Production Pitfall Guard (Kafka Architecture (Brokers, Topics, Partitions and KRaft)) • Hands-On Verification (Kafka Architecture (Brokers, Topics, Partitions and KRaft)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Kafka Architecture (Brokers, Topics, Partitions and...

Purpose: Replaying historical events for a brand-new analytics consumer because Kafka retains events for 7 days regardless of prior reads.

What Topic Is Next → Next Topic 4.4: Kafka Producers (Partitioning, Batching and Acks)

4.4 Kafka Producers (Partitioning, Batching and Acks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Key-based murmur2 partitioning • `acks=0` vs `acks=1` vs `acks=all` • `enable.idempotence=true` (default in modern Kafka!) • `linger.ms` & `batch.size` • Production Pitfall Guard (Kafka Producers (Partitioning, Batching and Acks)) • Hands-On Verification (Kafka Producers (Partitioning, Batching and Acks)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Kafka Producers (Partitioning, Batching and Acks)

Purpose: Passing `String.valueOf(orderId)` as the message key so all events for Order #42 land in the same partition in exact chronological order!

What Topic Is Next → Next Topic 4.5: Kafka Consumers (Poll Loop and Deserialization)

4.5 Kafka Consumers (Poll Loop and Deserialization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pull-based `poll()` model • `auto.offset.reset` (`earliest` vs `latest`) • Heartbeats vs `max.poll.interval.ms` • Production Pitfall Guard (Kafka Consumers (Poll Loop and Deserialization)) • Hands-On Verification (Kafka Consumers (Poll Loop and Deserialization)) • Kafka Consumers (Poll Loop and Deserialization) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Kafka Consumers (Poll Loop and Deserialization)

Purpose: Setting `auto-offset-reset: earliest` in microservices so a newly deployed consumer group never misses events published right before it booted.

What Topic Is Next → Next Topic 4.6: Consumer Groups and Parallelism

4.6 Consumer Groups and Parallelism [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Load-sharing partitions within one `group.id` vs independent broadcast across different `group.id`s • Why `activeConsumers <= partitionCount` • Production Pitfall Guard (Consumer Groups and Parallelism) • Hands-On Verification (Consumer Groups and Parallelism) • Consumer Groups and Parallelism — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Consumer Groups • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running 3 `inventory-service` instances (`group.id=inventory`) sharing 3 partitions in parallel while `notification-service` (`group.id=notifications`) receives its own f...

What Topic Is Next → Next Topic 4.7: Offsets and Commit Strategies

4.7 Offsets and Commit Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Current offset vs committed offset (`__consumer_offsets`) • Auto-commit risks (data loss if crash before processing finishes!) vs Spring Kafka `RECORD`/`BATCH` commit-after-process • Production Pitfall Guard (Offsets and Commit Strategies) • Hands-On Verification (Offsets and Commit Strategies) • Offsets and Commit Strategies — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Offsets • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring offsets commit ONLY *after* database persistence succeeds (`enable-auto-commit: false` + Spring `AckMode.RECORD`), guaranteeing At-Least-Once delivery.

What Topic Is Next → Next Topic 4.8: Partition Replication, ISR and Fault Tolerance

4.8 Partition Replication, ISR and Fault Tolerance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replication factor (`RF=3` in prod) • Partition Leader vs Followers • In-Sync Replicas (`ISR`) • `min.insync.replicas=2` + `acks=all` durability guarantee • Production Pitfall Guard (Partition Replication, ISR and Fault Tolerance) • Hands-On Verification (Partition Replication, ISR and Fault Tolerance) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Partition Replication

Purpose: Guaranteeing zero committed event loss even if a Kafka broker disk dies abruptly.

What Topic Is Next → Next Topic 4.9: Message Serialization (JSON and Apache Avro Fundamentals)

4.9 Message Serialization (JSON and Apache Avro Fundamentals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `JsonSerializer` / `ErrorHandlingDeserializer` wrapping `JsonDeserializer` (critical for poison pills!) • Backward/forward schema evolution & Avro Schema Registry overview • Production Pitfall Guard (Message Serialization (JSON and Apache Avro Fundamentals)) • Hands-On Verification (Message Serialization (JSON and Apache Avro Fundamentals)) • Message Serialization (JSON and Apache Avro Fundamentals) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Message Serialization (JSON and Apache Avro Fundamen... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Wrapping `JsonDeserializer` inside `ErrorHandlingDeserializer` so a single malformed JSON message routes to DLQ instead of bricking the consumer in an infinite deserializ...

What Topic Is Next → Next Topic 4.10: Spring for Apache Kafka Integration

4.10 Spring for Apache Kafka Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `spring-kafka` starter • `KafkaTemplate<String • Object>` • `NewTopic` `@Bean` auto-creation • Production Pitfall Guard (Spring for Apache Kafka Integration) • Hands-On Verification (Spring for Apache Kafka Integration) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring for Apache Kafka Integration

Purpose: Auto-provisioning `order-events`, `inventory-events`, and `payment-events` topics cleanly via `TopicBuilder`.

What Topic Is Next → Next Topic 4.11: Kafka Listeners (@KafkaListener)

4.11 Kafka Listeners (@KafkaListener) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@KafkaListener(topics = ... • GroupId = ...)` • Extracting `@Payload` • `@Header(KafkaHeaders.RECEIVED_KEY)` • Production Pitfall Guard (Kafka Listeners (@KafkaListener)) • Hands-On Verification (Kafka Listeners (@KafkaListener)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Kafka Listeners (@KafkaListener)

Purpose: Logging topic, partition, offset, and `eventId` cleanly in every consumer handler.

What Topic Is Next → Next Topic 4.12: Consumer Error Handling

4.12 Consumer Error Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Transient errors (DB/network blip -> retry!) vs non-retryable errors (`IllegalArgumentException` / invalid data -> skip straight to DLT!) • `DefaultErrorHandler` + `FixedBackOff` / `ExponentialBackOff` • Production Pitfall Guard (Consumer Error Handling) • Hands-On Verification (Consumer Error Handling) • Consumer Error Handling — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Consumer Error Handling • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing a single bad event from blocking 10,000 valid subsequent events behind it in the partition.

What Topic Is Next → Next Topic 4.13: Retry Topics (@RetryableTopic)

4.13 Retry Topics (@RetryableTopic) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Non-blocking delayed retries via Spring Kafka `@RetryableTopic(attempts = "4" • Backoff = @Backoff(delay = 1000 • Multiplier = 2.0))` • Main partition stays unblocked! • Production Pitfall Guard (Retry Topics (@RetryableTopic)) • Hands-On Verification (Retry Topics (@RetryableTopic)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Retry Topics (@RetryableTopic)

Purpose: Retrying temporary SMTP/webhook failures across `-retry-0`, `-retry-1` topics without stalling real-time order events on the main topic.

What Topic Is Next → Next Topic 4.14: Dead Letter Queue (DLQ / DLT)

4.14 Dead Letter Queue (DLQ / DLT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `DeadLetterPublishingRecoverer` • `@DltHandler` • Storing diagnostic failure headers (`kafka_dlt-exception-message`) • Alerting & replay runbooks • Production Pitfall Guard (Dead Letter Queue (DLQ / DLT)) • Hands-On Verification (Dead Letter Queue (DLQ / DLT)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory

Lifecycle for Dead Letter Queue (DLQ / DLT)

Purpose: Capturing exhausted failed messages into `order-events.DLT` + persisting an admin alert record so zero customer events vanish silently.

What Topic Is Next → Next Topic 4.15: Event-Driven Design (Domain Events vs Integration Events)

4.15 Event-Driven Design (Domain Events vs Integration Events) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Internal Domain Events vs public versioned cross-service Integration Events • Thin notification events vs self-contained Event-Carried State Transfer • Schema backward compatibility (additive optional fields!) • Production Pitfall Guard (Event-Driven Design (Domain Events vs Integration Events)) • Hands-On Verification (Event-Driven Design (Domain Events vs Integration Events)) • Event-Driven Design (Domain Events vs Integration Events) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Event-Driven Design (Domain Events vs Integration Ev...

Purpose: Including `eventId`, `eventType`, `schemaVersion`, `occurredAt`, and `correlationId` metadata on every integration event.

What Topic Is Next → Next Topic 4.16: Message Ordering Guarantees

4.16 Message Ordering Guarantees [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Per-partition FIFO guarantee • Choosing the right partition key (`orderId` for order lifecycle • `accountId` for ledger) • `max.in.flight.requests.per.connection <= 5` with idempotent producer • Production Pitfall Guard (Message Ordering Guarantees) • Hands-On Verification (Message Ordering Guarantees) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Message Ordering Guarantees

Purpose: Ensuring `OrderPlaced -> OrderPaid -> OrderShipped` for `orderId=9001` are always processed in exact sequence.

What Topic Is Next → Next Topic 4.17: Idempotent Consumers (Deduplicating At-Least-Once Delivery)

4.17 Idempotent Consumers (Deduplicating At-Least-Once Delivery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why network/rebalance hiccups cause duplicate deliveries in Kafka ("At-Least-Once") • Inserting `event_id UUID PRIMARY KEY` into `processed_events` inside the SAME local `@Transactional` DB commit as the business write! • Production Pitfall Guard (Idempotent Consumers (Deduplicating At-Least-Once Delivery)) • Hands-On Verification (Idempotent Consumers (Deduplicating At-Least-Once Delivery)) • Idempotent Consumers (Deduplicating At-Least-Once Delivery) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Idempotent Consumers (Deduplicating At-Least-Once De... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring that if `PaymentConfirmedEvent(eventId=abc)` is delivered twice after a rebalance, inventory is deducted and email sent ONLY ONCE!

What Topic Is Next → Next Topic 4.18: Transactional Outbox Pattern

4.18 Transactional Outbox Pattern [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: The Dual-Write bug (`orderRepo.save()` commits in PostgreSQL • Then JVM crashes 1ms before `kafkaTemplate.send()` -> lost event!) • Solving by writing `Order` + `OutboxEvent` row in ONE PostgreSQL transaction • Relaying via `@Scheduled` poller / CDC • Production Pitfall Guard (Transactional Outbox Pattern) • Hands-On Verification (Transactional Outbox Pattern) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Transactional Outbox Pattern

Purpose: Achieving 100% reliable event publishing in financial and e-commerce order services without distributed XA transactions.

What Topic Is Next → Module 4 Capstone (Week 19 Lab Build: Event-Driven Kafka Order & Notification Pipeline) → Next Topic 5.1: NoSQL vs Relational Databases (Polyglot Persistence)

Module 4 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Event-Driven Kafka Order & Notification Pipeline — Implement asynchronous `OrderPlaced`, `InventoryUpdated`, `PaymentConfirmed`, and `EmailNotification` workflows using Spring Kafka, partition keys, `@RetryableTopic`, `.DLT` Dead Letter Queu...

MODULE 5 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Redis, NoSQL and Backend Performance

Accelerate distributed backends using Polyglot Persistence (MongoDB document fundamentals) and Redis in-memory caching & rate limiting — MongoDB documents/indexes/aggregations, Spring Data MongoDB, Redis Strings/Hashes/Lists/Sets, Cache-Aside pattern, Spring Cache (@Cacheable/@CachePut/@CacheEvict), TTL & stampede prevention, HikariCP & N+1 tuning, and Redis rate limiting — culminating in Major Project 5.

Objectives: ✓ Choose objectively between PostgreSQL (relational ACID), MongoDB (flexible document schemas), and Redis (sub-millisecond in-memory state). • ✓ Model documents, create compound indexes, run aggregation pipelines, and build `MongoRepository` services in Spring Data MongoDB. • ✓ Use Redis Strings, Hashes, Lists, and Sets and implement Cache-Aside caching cleanly with `@Cacheable`, `@CachePut`, and `@CacheEvict`. • ✓ Configure randomized TTLs, JSON Redis serialization, cache invalidation on writes, and Redis-backed API rate limiting. • ✓ Deliver Month 5 Deliverable: Major Project 5 — Distributed E-Commerce Microservices Application (`Gateway`, `Eureka`, `User`, `Product`, `Order`, `Inventory`, `Payment`, `Notification`, `Kafka`, `Redis`, `PostgreSQL`).

5.1 NoSQL vs Relational Databases (Polyglot Persistence) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Document • Key-Value • Wide-Column • Graph stores vs RDBMS • Production Pitfall Guard (NoSQL vs Relational Databases (Polyglot Persistence)) • Hands-On Verification (NoSQL vs Relational Databases (Polyglot Persistence)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for NoSQL vs Relational Databases (Polyglot Persistence)

Purpose: Matching each microservice's storage engine to its exact access pattern instead of forcing one tool everywhere.

What Topic Is Next → Next Topic 5.2: MongoDB Fundamentals

5.2 MongoDB Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: BSON binary JSON documents • Collections • Dynamic/schema-validated document structure • `ObjectId` 12-byte structure • Production Pitfall Guard (MongoDB Fundamentals) • Hands-On Verification (MongoDB Fundamentals) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts &

non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for MongoDB Fundamentals

Purpose: Storing polymorphic category attributes (laptops have `ramGb`/`cpu`; shirts have `fabric`/`chestIn`) without 40 nullable SQL columns.

What Topic Is Next → Next Topic 5.3: MongoDB Data Modeling (Embedding vs Referencing)

5.3 MongoDB Data Modeling (Embedding vs Referencing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Embedding bounded 1:few sub-documents (`addresses` on `user`) vs Referencing unbounded 1:million collections (`reviews` referencing `productId`) • 16MB BSON document limit! • Production Pitfall Guard (MongoDB Data Modeling (Embedding vs Referencing)) • Hands-On Verification (MongoDB Data Modeling (Embedding vs Referencing)) • MongoDB Data Modeling (Embedding vs Referencing) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for MongoDB Data Modeling (Embedding vs Referencing) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Embedding 3–8 product specification pairs inside `ProductDoc` while keeping unbounded customer `Reviews` as a separate collection!

What Topic Is Next → Next Topic 5.4: MongoDB CRUD Operations

5.4 MongoDB CRUD Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `insertOne`/`insertMany` • `find` projections • `$gt`/`$in` filters • Atomic `$set` / `$inc` / `$push` updates (never overwrite whole doc blindly!) • Production Pitfall Guard (MongoDB CRUD Operations) • Hands-On Verification (MongoDB CRUD Operations) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for MongoDB CRUD Operations

Purpose: Atomically incrementing view counters (`$inc: { views: 1 }`) without read-modify-write lost updates.

What Topic Is Next → Next Topic 5.5: MongoDB Indexing

5.5 MongoDB Indexing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default `_id` index • Compound indexes & ESR rule (`Equality -> Sort -> Range`) • Text/TTL indexes (`expireAfterSeconds`) • `.explain("executionStats")` (`IXSCAN` vs `COLLSCAN`) • Production Pitfall Guard (MongoDB Indexing) • Hands-On Verification (MongoDB Indexing) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for MongoDB Indexing

Purpose: Auto-expiring audit/OTP documents via a MongoDB TTL index and verifying `IXSCAN` via `explain`.

What Topic Is Next → Next Topic 5.6: MongoDB Aggregation Pipeline Fundamentals

5.6 MongoDB Aggregation Pipeline Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage `$match` (push early to use indexes!) -> `$group` (`$avg` • `$sum`) -> `$sort` -> `$limit` -> `$project` • Production Pitfall Guard (MongoDB Aggregation Pipeline Fundamentals) • Hands-On Verification (MongoDB Aggregation Pipeline Fundamentals) • MongoDB Aggregation Pipeline Fundamentals — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for MongoDB Aggregation Pipeline Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Computing average star rating and rating-count breakdown per `productId` directly inside MongoDB.

What Topic Is Next → Next Topic 5.7: Spring Data MongoDB (MongoRepository and MongoTemplate)

5.7 Spring Data MongoDB (MongoRepository and MongoTemplate) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Document(collection = "reviews")` • `MongoRepository<ReviewDoc • String>` • Derived queries • Production Pitfall Guard (Spring Data MongoDB (MongoRepository and MongoTemplate)) • Hands-On Verification (Spring Data MongoDB (MongoRepository and MongoTemplate)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Data MongoDB (MongoRepository and MongoTempla...

Purpose: Integrating a clean `ReviewRepository` microservice module with Spring Boot 3.

What Topic Is Next → Next Topic 5.8: Redis In-Memory Data Store Architecture

5.8 Redis In-Memory Data Store Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-threaded event-loop command execution (atomic commands • Sub-millisecond latency!) • RDB snapshots & AOF persistence modes • Lettuce non-blocking Spring Boot client • Production Pitfall Guard (Redis In-Memory Data Store Architecture) • Hands-On Verification (Redis In-Memory Data Store Architecture) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Redis In-Memory Data Store Architecture

Purpose: Serving hot product details, active promos, token blacklists, and rate-limit counters in < 1ms.

What Topic Is Next → Next Topic 5.9: Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)

5.9 Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strings (`GET/SET EX/INCR`) • Hashes (`HSET/HGETALL` for user carts!) • Lists (`LPUSH/RPOP`) • Sets (`SADD/SISMEMBER`) • Production Pitfall Guard (Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)) • Hands-On Verification (Redis Core Data Structures (Strings, Hashes, Lists, Sets, Sorted Sets)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Redis Core Data Structures (Strings, Hashes, Lists,...

Purpose: Storing active shopping carts as Redis Hashes (`cart:{userId}` -> `field=productId, value=qty`) with automatic 7-day TTL!

What Topic Is Next → Next Topic 5.10: Caching Patterns (Cache-Aside, Write-Through, Read-Through)

5.10 Caching Patterns (Cache-Aside, Write-Through, Read-Through) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cache-Aside (Read: check Redis -> on miss read PostgreSQL -> populate Redis with TTL • Write: update PostgreSQL -> evict Redis key!) • Cache hit ratio formulas • Production Pitfall Guard (Caching Patterns (Cache-Aside, Write-Through, Read-Through)) • Hands-On Verification (Caching Patterns (Cache-Aside, Write-Through, Read-Through)) • Caching Patterns (Cache-Aside, Write-Through, Read-Through) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory

Lifecycle for Caching Patterns (Cache-Aside, Write-Through, Read-T...

Purpose: Offloading 90%+ of repetitive `GET /api/v1/products/{id}` queries from PostgreSQL during high-traffic sales.

What Topic Is Next → Next Topic 5.11: Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)

5.11 Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EnableCaching` • Configuring `RedisCacheManager` with `GenericJackson2JsonRedisSerializer` (not unreadable JDK binary serialization!) • SpEL keys `key = "#id"` • `unless = "#result == null"` • Production Pitfall Guard (Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)) • Hands-On Verification (Spring Cache Abstraction (@Cacheable, @CachePut, @CacheEvict)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Spring Cache Abstraction (@Cacheable, @CachePut, @Ca...

Purpose: Declaratively caching `getProductById` with `@Cacheable(cacheNames = "products", key = "#id")` and evicting on `updateProduct`/`deleteProduct`!

What Topic Is Next → Next Topic 5.12: Cache Invalidation, Penetration and Stampede Protection

5.12 Cache Invalidation, Penetration and Stampede Protection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why cache invalidation is hard • Cache Penetration (caching short-TTL empty sentinel or Bloom filter for non-existent IDs) • Cache Avalanche (adding random TTL jitter!) • Cache Stampede (`sync = true` lock) • Production Pitfall Guard (Cache Invalidation, Penetration and Stampede Protection) • Hands-On Verification (Cache Invalidation, Penetration and Stampede Protection) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Cache Invalidation

Purpose: Using `@Cacheable(..., sync = true)` on hot homepage/banner keys so 500 concurrent threads don't stampede PostgreSQL simultaneously when TTL expires!

What Topic Is Next → Next Topic 5.13: Time-To-Live (TTL) and Eviction Policies

5.13 Time-To-Live (TTL) and Eviction Policies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mandatory TTL on every cache key (`EXPIRE`) • `maxmemory` policies (`allkeys-lru` • `volatile-lru` • `noeviction`) • Production Pitfall Guard (Time-To-Live (TTL) and Eviction Policies) • Hands-On Verification (Time-To-Live (TTL) and Eviction Policies) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Time-To-Live (TTL)

Purpose: Setting 15-minute TTL on catalog pages, 5-minute TTL on OTPs, and exact remaining-JWT-lifetime TTL on revoked token `jti` blacklists!

What Topic Is Next → Next Topic 5.14: Backend Database and API Performance Profiling

5.14 Backend Database and API Performance Profiling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Measuring p50/p95/p99 latency (not misleading averages!) • SQL query count assertions • Payload compression • Avoiding chatty N-call loops • Production Pitfall Guard (Backend Database and API Performance Profiling) • Hands-On Verification (Backend Database and API Performance Profiling) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Backend Database

Purpose: Running a 50-concurrent-user JMeter test plan to measure p95 latency before and after Redis caching + SQL indexing.

What Topic Is Next → Next Topic 5.15: Connection Pooling Tuning in Microservices

5.15 Connection Pooling Tuning in Microservices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sizing HikariCP per microservice (`total_connections = replicas * maxPoolSize < PostgreSQL max_connections`) • `leak-detection-threshold` • Fast `connection-timeout` • Production Pitfall Guard (Connection Pooling Tuning in Microservices) • Hands-On Verification (Connection Pooling Tuning in Microservices) • Connection Pooling Tuning in Microservices — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Connection Pooling Tuning in Microservices

Purpose: Configuring `maximum-pool-size: 10` across 6 microservices (60 total connections < PostgreSQL default 100 limit).

What Topic Is Next → Next Topic 5.16: Detecting and Eliminating N+1 Queries at Scale

5.16 Detecting and Eliminating N+1 Queries at Scale [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@EntityGraph(attributePaths = {"category"})` • Constructor DTO projections • Two-query ID-page + `IN (:ids)` collection fetch • Hibernate statistics check • Production Pitfall Guard (Detecting and Eliminating N+1 Queries at Scale) • Hands-On Verification (Detecting and Eliminating N+1 Queries at Scale) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Detecting

Purpose: Fetching 20 paginated orders + all their `OrderItem` lines in exactly 2 SQL queries instead of 21 queries (`1 + 20`)!

What Topic Is Next → Next Topic 5.17: High-Volume API Pagination Strategies

5.17 High-Volume API Pagination Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Slice<T>` (avoids expensive `SELECT COUNT(*)` on millions of rows!) • Cursor-based `?afterId=1050&limit=20` pagination for feeds/events • Production Pitfall Guard (High-Volume API Pagination Strategies) • Hands-On Verification (High-Volume API Pagination Strategies) • High-Volume API Pagination Strategies — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for High-Volume API Pagination Strategies • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Powering sub-5ms order history and job-listing feeds without full-table `COUNT(*)` bottlenecks.

What Topic Is Next → Next Topic 5.18: Redis-Based Distributed Rate Limiting

5.18 Redis-Based Distributed Rate Limiting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why JVM-local rate limiters leak `N * limit` across `N` replicas • Atomic Redis fixed-window (`INCR` + `EXPIRE` Lua script) & Spring Cloud Gateway `RequestRateLimiter` token bucket • Production Pitfall Guard (Redis-Based Distributed Rate Limiting) • Hands-On Verification (Redis-Based Distributed Rate Limiting) • Redis-Based Distributed Rate Limiting — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking

patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Redis-Based Distributed Rate Limiting • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Enforcing a cluster-wide limit of `10 req/sec` per IP/API-key at `api-gateway` backed by shared Redis!

What Topic Is Next → Module 5 Capstone (Major Project 5: Distributed E-Commerce Microservices Application) → Next Topic 6.1: Software Testing Fundamentals (Unit, Integration and System Testing)

Module 5 Hands-On Capstone Project (100 Marks): Major Project 5: Distributed E-Commerce Microservices Application — Evolve the commerce domain into a resilient, event-driven distributed Microservices system consisting of independently deployable Spring Boot 3 services (`API Gateway`, `Eureka Server`, `Use...

MODULE 6 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 19 TOPICS (152 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Testing and Quality Engineering

Build automated quality confidence across every backend layer — the Testing Pyramid, JUnit 5 lifecycle & parameterized tests, Mockito mocks/stubs/spies/captors, Spring Boot Test slices (@WebMvcTest + MockMvc, @DataJpaTest), full-stack integration testing with real Docker PostgreSQL via Testcontainers, Postman/OpenAPI contract checks, JaCoCo coverage, and regression discipline.

Objectives: ✓ Balance fast isolated unit tests, focused Spring slice tests (`@WebMvcTest`, `@DataJpaTest`), and real-database integration tests (`@SpringBootTest` + `Testcontainers`). • ✓ Write expressive JUnit 5 tests using `@BeforeEach`, `@Nested`, `@ParameterizedTest`, and clear `Arrange -> Act -> Assert` structure. • ✓ Isolate `@Service` business logic in < 50ms using `@ExtendWith(MockitoExtension.class)`, `@Mock`, `@InjectMocks`, `when(...).thenReturn(...)`, and `verify(...)`. • ✓ Test HTTP status codes, `@Valid` validation errors, and JWT/RBAC rules via `@WebMvcTest` + `MockMvc`, and verify PostgreSQL queries against real containerized Postgres via `Testcontainers`. • ✓ Measure meaningful JaCoCo branch/line coverage and prevent production regressions.

6.1 Software Testing Fundamentals (Unit, Integration and System Testing)

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deterministic fast unit tests vs component/DB integration tests vs end-to-end API contract tests • Arrange-Act-Assert (Given-When-Then) discipline • Production Pitfall Guard (Software Testing Fundamentals (Unit, Integration and System Testing)) • Hands-On Verification (Software Testing Fundamentals (Unit, Integration and System Testing)) • Software Testing Fundamentals (Unit, Integration and System Testing) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Software Testing Fundamentals (Unit, Integration and... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Catching broken business calculations, SQL bugs, and security regressions automatically before any PR merges.

What Topic Is Next → Next Topic 6.2: The Testing Pyramid in Spring Boot Backends

6.2 The Testing Pyramid in Spring Boot Backends

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ~70% fast Mockito/domain unit tests • ~20% `@WebMvcTest`/`@DataJpaTest`/`Testcontainers` slice & integration tests • ~10% E2E smoke workflows • Production Pitfall Guard (The Testing Pyramid in Spring Boot Backends) • Hands-On Verification (The Testing Pyramid in Spring Boot Backends) • The Testing Pyramid in Spring Boot Backends — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for The Testing Pyramid in Spring Boot Backends

Purpose: Keeping `./mvnw test` under 30 seconds so developers actually run tests before every commit.

What Topic Is Next → Next Topic 6.3: JUnit 5 (Jupiter) Architecture and Lifecycle

6.3 JUnit 5 (Jupiter) Architecture and Lifecycle

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@Test` • `@DisplayName` • `@BeforeEach` • `@AfterEach` • Production Pitfall Guard (JUnit 5 (Jupiter) Architecture and Lifecycle) • Hands-On Verification (JUnit 5 (Jupiter) Architecture and Lifecycle) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for JUnit 5 (Jupiter) Architecture

Purpose: Organizing `OrderServiceTest` into readable `@Nested class WhenPlacingOrder` and `@Nested class WhenCancellingOrder` suites.

What Topic Is Next → Next Topic 6.4: JUnit 5 Assertions and Exception Verification

6.4 JUnit 5 Assertions and Exception Verification

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `assertEquals` • `assertTrue` • `assertNotNull` • `assertAll` (reports ALL failures at once!) • Production Pitfall Guard (JUnit 5 Assertions and Exception Verification) • Hands-On Verification (JUnit 5 Assertions and Exception Verification) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for JUnit 5 Assertions

Purpose: Verifying both happy-path return DTO fields (`assertAll`) and exact domain exceptions (`assertThrows`) when business rules reject bad inputs.

What Topic Is Next → Next Topic 6.5: Parameterized Tests (@ParameterizedTest)

6.5 Parameterized Tests (@ParameterizedTest)

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@ValueSource` • `@CsvSource` • `@EnumSource` • `@MethodSource` for testing 10 boundary/invalid inputs with zero copy-pasted test methods • Production Pitfall Guard (Parameterized Tests (@ParameterizedTest)) • Hands-On Verification (Parameterized Tests (@ParameterizedTest)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Parameterized Tests (@ParameterizedTest)

Purpose: Testing tax slabs, coupon rules, SKU regexes, and password strength across 12 edge cases in one concise method.

What Topic Is Next → Next Topic 6.6: Mockito Fundamentals (@ExtendWith(MockitoExtension.class))

6.6 Mockito Fundamentals (@ExtendWith(MockitoExtension.class))

[50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pure Java + Mockito unit tests (zero Spring context startup!) • `@Mock` • `@InjectMocks` • Stubbing `when(repo.findById(1L)).thenReturn(Optional.of(entity))` • Production Pitfall Guard (Mockito Fundamentals (@ExtendWith(MockitoExtension.class))) • Hands-On Verification (Mockito Fundamentals (@ExtendWith(MockitoExtension.class))) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Mockito Fundamentals (@ExtendWith(MockitoExtension.c...

Purpose: Testing 15 complex `OrderService` branches (out of stock, expired coupon, duplicate SKU, inactive user) in < 100ms total!

What Topic Is Next → Next Topic 6.7: Test Doubles (Mock vs Stub vs Spy vs Fake)

6.7 Test Doubles (Mock vs Stub vs Spy vs Fake) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stubs (canned answers) • Mocks (interaction verification) • `@Spy` (partial real object — use sparingly!) • In-memory Fakes (`FakeClock` • Production Pitfall Guard (Test Doubles (Mock vs Stub vs Spy vs Fake)) • Hands-On Verification (Test Doubles (Mock vs Stub vs Spy vs Fake)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Test Doubles (Mock vs Stub vs Spy vs Fake)

Purpose: Passing `Clock.fixed(Instant.parse("2026-11-17T00:00:00Z"), ZoneOffset.UTC)` instead of mocking static `Instant.now()`!

What Topic Is Next → Next Topic 6.8: Interaction Verification (verify, ArgumentCaptor and Matchers)

6.8 Interaction Verification (verify, ArgumentCaptor and Matchers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `verify(kafkaTemplate).send(...)` • `verify(repo • Never()).save(any())` on validation failure • `ArgumentCaptor<Order>` to inspect exact saved fields • Production Pitfall Guard (Interaction Verification (verify, ArgumentCaptor and Matchers)) • Hands-On Verification (Interaction Verification (verify, ArgumentCaptor and Matchers)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Interaction Verification (verify, ArgumentCaptor and...

Purpose: Proving `orderRepo.save(...)` was NEVER called when stock check failed and capturing the exact `Order` persisted on success.

What Topic Is Next → Next Topic 6.9: Full Context Integration Testing (@SpringBootTest)

6.9 Full Context Integration Testing (@SpringBootTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WebEnvironment.RANDOM_PORT` vs `MOCK` • `TestRestTemplate` / `RestTestClient` • `@ActiveProfiles("test")` • Minimizing context reloads across test classes via shared base config • Production Pitfall Guard (Full Context Integration Testing (@SpringBootTest)) • Hands-On Verification (Full Context Integration Testing (@SpringBootTest)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Full Context Integration Testing (@SpringBootTest)

Purpose: Verifying full end-to-end wiring from HTTP FilterChain -> Controller -> Transactional Service -> JPA -> Database.

What Topic Is Next → Next Topic 6.10: Web Layer Testing with MockMvc

6.10 Web Layer Testing with MockMvc [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `mockMvc.perform(post(...).contentType(APPLICATION_JSON).content(...))` • `status()` • `header()` • `jsonPath("$.id")` • Production Pitfall Guard (Web Layer Testing with MockMvc) • Hands-On Verification (Web Layer Testing with MockMvc) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Web Layer Testing with MockMvc

Purpose: Asserting exact HTTP status codes (`201`, `400`, `401`, `403`, `404`, `409`), `Location` headers, and JSON field contracts without opening a TCP port.

What Topic Is Next → Next Topic 6.11: Repository Slice Testing (@DataJpaTest)

6.11 Repository Slice Testing (@DataJpaTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Focused JPA/Hibernate slice • Automatic per-test transaction rollback • `TestEntityManager` (`persistAndFlush` + `clear` to test real DB reads rather than L1 cache!) • Production Pitfall Guard (Repository Slice Testing (@DataJpaTest)) • Hands-On Verification (Repository Slice Testing (@DataJpaTest)) • Repository Slice Testing (@DataJpaTest) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Repository Slice Testing (@DataJpaTest)

Purpose: Verifying custom `@Query` JPQL, unique constraints, Specifications, and pagination queries fast and cleanly.

What Topic Is Next → Next Topic 6.12: Controller Slice Testing (@WebMvcTest)

6.12 Controller Slice Testing (@WebMvcTest) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading ONLY web layer (`@RestController` • `@RestControllerAdvice` • Jackson • Bean Validation • Production Pitfall Guard (Controller Slice Testing (@WebMvcTest)) • Hands-On Verification (Controller Slice Testing (@WebMvcTest)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Controller Slice Testing (@WebMvcTest)

Purpose: Testing all 25 HTTP endpoints, request validations, and role guards in seconds with zero database startup.

What Topic Is Next → Next Topic 6.13: Database and API Integration Testing

6.13 Database and API Integration Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end HTTP -> Service -> JPA -> PostgreSQL flow verification • Test data isolation (`@Sql` / clean setup) • Testing real transaction rollbacks & optimistic locks • Production Pitfall Guard (Database and API Integration Testing) • Hands-On Verification (Database and API Integration Testing) • Database and API Integration Testing — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Database

Purpose: Verifying that a failed checkout truly leaves `orders` empty and `products.stock` untouched in PostgreSQL.

What Topic Is Next → Next Topic 6.14: Testcontainers (Real PostgreSQL & Redis in Docker Tests)

6.14 Testcontainers (Real PostgreSQL & Redis in Docker Tests) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `org.testcontainers:postgresql` • `@ServiceConnection` (Spring Boot 3.1+ zero-boilerplate wiring!) • Catching PostgreSQL-specific SQL/JSONB/constraint bugs that H2 misses • Production Pitfall Guard (Testcontainers (Real PostgreSQL & Redis in Docker Tests)) • Hands-On Verification (Testcontainers (Real PostgreSQL & Redis in Docker Tests)) • Testcontainers (Real PostgreSQL & Redis in Docker Tests) — Interview Readiness • Idiomatic JMSC

Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Testcontainers (Real PostgreSQL & Redis in Docker Te...

Purpose: Running CI integration tests against a real ephemeral `postgres:16-alpine` container via `@ServiceConnection`!

What Topic Is Next → Next Topic 6.15: Automated API Contract Testing with Postman / Newman

6.15 Automated API Contract Testing with Postman / Newman [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chaining Postman requests (Register -> Login capture `token` -> Create Product capture `id` -> Place Order -> Assert Stock) • CLI execution via Newman in CI • Production Pitfall Guard (Automated API Contract Testing with Postman / Newman) • Hands-On Verification (Automated API Contract Testing with Postman / Newman) • Automated API Contract Testing with Postman / Newman — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Automated API Contract Testing with Postman / Newman • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Shipping a self-verifying `postman_collection.json` alongside your Capstone repository.

What Topic Is Next → Next Topic 6.16: OpenAPI & Swagger Contract Verification

6.16 OpenAPI & Swagger Contract Verification [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying `/v3/api-docs` generates valid OpenAPI 3.0/3.1 JSON without schema conflicts • Documenting `400`/`401`/`403`/`404` responses accurately • Production Pitfall Guard (OpenAPI & Swagger Contract Verification) • Hands-On Verification (OpenAPI & Swagger Contract Verification) • OpenAPI & Swagger Contract Verification — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for OpenAPI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Adding a fast `MockMvc` test asserting `GET /v3/api-docs` returns `200 OK` so broken Swagger annotations never break runtime docs.

What Topic Is Next → Next Topic 6.17: Clean Code, SOLID Principles and Maintainability

6.17 Clean Code, SOLID Principles and Maintainability [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single Responsibility • Dependency Inversion • Small pure functions • Guard clauses • Production Pitfall Guard (Clean Code, SOLID Principles and Maintainability) • Hands-On Verification (Clean Code, SOLID Principles and Maintainability) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Clean Code

Purpose: Keeping backend codebases easy to extend, review, and explain clearly during live code walkthrough interviews.

What Topic Is Next → Next Topic 6.18: Code Coverage with JaCoCo

6.18 Code Coverage with JaCoCo [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `jacoco-maven-plugin` • Line vs branch coverage (`if/else` paths!) • Excluding auto-generated DTOs/Application main • Why meaningful assertions beat vanity 100% getter tests • Production Pitfall Guard (Code Coverage with JaCoCo) • Hands-On Verification (Code Coverage with JaCoCo) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Code Coverage with JaCoCo

Purpose: Generating `target/site/jacoco/index.html` in Maven to spot untested error/security branches in `@Service` classes.

What Topic Is Next → Next Topic 6.19: Regression Testing Discipline (Bug -> Failing Test -> Fix)

6.19 Regression Testing Discipline (Bug -> Failing Test -> Fix) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reproducing every reported bug as a failing automated test FIRST • Fixing the code until green • Keeping that regression guard in CI permanently • Production Pitfall Guard (Regression Testing Discipline (Bug -> Failing Test -> Fix)) • Hands-On Verification (Regression Testing Discipline (Bug -> Failing Test -> Fix)) • Regression Testing Discipline (Bug -> Failing Test -> Fix) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Regression Testing Discipline (Bug -> Failing Test -...

Purpose: Ensuring fixed production edge cases (e.g., rounding error or expired-coupon bypass) can NEVER silently return months later.

What Topic Is Next → Module 6 Capstone (Week 21 Lab Build: Full-Pyramid JUnit 5, Mockito & Testcontainers Suite) → Next Topic 7.1: Linux Architecture and Server Distributions

Module 6 Hands-On Capstone Project (100 Marks): Week 21 Lab Build: Full-Pyramid JUnit 5, Mockito & Testcontainers Suite — Verify a Spring Boot backend with >= 75% service coverage using JUnit 5 `@ParameterizedTest`, Mockito unit tests, `@WebMvcTest` + `MockMvc` security/validation slice tests, `@DataJpaTest`, a...

MODULE 7 OF 8 • STAGE 2 • 11 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Docker, Linux and CI/CD Fundamentals

Package, run, network, and automate Java 21 + Spring Boot 3 backends at a practical production-ready level — Linux CLI/permissions/processes/networking, multi-stage Java 21 Dockerfiles, container volumes & networks, multi-service Docker Compose stacks (Spring Boot + PostgreSQL + Redis + Kafka), Maven build lifecycles, GitHub Actions CI/CD pipelines, cloud deployment fundamentals, and secret hygiene.

Objectives: ✓ Navigate Linux servers confidently using essential filesystem, permission (`chmod`/`chown`), process (`ps`/`top`/`kill`), networking (`ss`/`curl`), and environment commands. • ✓ Author optimized, non-root, multi-stage Java 21 `Dockerfile`s (`maven:3.9-eclipse-temurin-21` build stage -> slim `eclipse-temurin:21-jre-alpine` runtime stage). • ✓ Manage container logs, volumes, bridge networks, and health-checked `docker-compose.yml` stacks running Spring Boot + PostgreSQL + Redis + Kafka. • ✓ Automate `mvn -B verify` + Docker image builds on every push/PR using GitHub Actions workflows. • ✓ Deploy a containerized Spring Boot backend cleanly with `prod` profiles, externalized environment variables, and strict secret management.

7.1 Linux Architecture and Server Distributions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Kernel vs shell/userland • Ubuntu/Debian vs RHEL/Rocky vs minimal Alpine containers • Filesystem hierarchy (`/etc` • `/var/log` • Production Pitfall Guard (Linux Architecture and Server Distributions) • Hands-On Verification (Linux Architecture and Server Distributions) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Linux Architecture

Purpose: Understanding the Linux runtime environment inside 99% of production cloud VMs and Docker containers.

What Topic Is Next → Next Topic 7.2: Essential Linux Terminal Commands

7.2 Essential Linux Terminal Commands [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pwd` • `ls -la` • `cd` • `mkdir -p` • Production Pitfall Guard (Essential Linux Terminal Commands) • Hands-On Verification (Essential Linux Terminal Commands) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Essential Linux Terminal Commands

Purpose: Tailing live Spring Boot logs (`tail -f app.log | grep ERROR`) and searching config files over SSH.

What Topic Is Next → Next Topic 7.3: Linux File Permissions (chmod and chown)

7.3 Linux File Permissions (chmod and chown) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Owner / Group / Others (`rwx` = `4 • 1`) • `chmod 755` (`mvnw` executable bit!) • `chmod 600` (SSH keys & secret env files) • Production Pitfall Guard (Linux File Permissions (chmod and chown)) • Hands-On Verification (Linux File Permissions (chmod and chown)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Linux File Permissions (chmod and chown)

Purpose: Fixing `Permission denied: ./mvnw` in Linux CI via `chmod +x mvnw` (`git update-index --chmod=+x mvnw`) and running containers as non-root `appuser`.

What Topic Is Next → Next Topic 7.4: Linux Process Management (ps, top, jps, kill)

7.4 Linux Process Management (ps, top, jps, kill) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ps aux | grep java` • `top`/`htop` CPU & RSS memory • JDK `jps`/`jstack` • Graceful `kill -15` (`SIGTERM`) vs last-resort `kill -9` (`SIGKILL`) • Production Pitfall Guard (Linux Process Management (ps, top, jps, kill)) • Hands-On Verification (Linux Process Management (ps, top, jps, kill)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Linux Process Management (ps, top, jps, kill)

Purpose: Sending `SIGTERM` (`kill -15`) so Spring Boot's graceful shutdown finishes active requests and closes HikariCP/Kafka cleanly.

What Topic Is Next → Next Topic 7.5: Linux Networking Basics (IP, Ports, DNS, HTTP, curl)

7.5 Linux Networking Basics (IP, Ports, DNS, HTTP, curl) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `127.0.0.1` loopback vs `0.0.0.0` all interfaces • Checking listening ports (`ss -tulpn` / `lsof -i :8080`) • DNS (`nslookup`/`dig`) • `curl -i -X POST` health & API debugging • Production Pitfall Guard (Linux Networking Basics (IP, Ports, DNS, HTTP, curl)) • Hands-On Verification (Linux Networking Basics (IP, Ports, DNS, HTTP, curl)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Linux Networking Basics (IP, Ports, DNS, HTTP, curl)

Purpose: Diagnosing `Address already in use: 8080` or `Connection refused` (binding `127.0.0.1` instead of `0.0.0.0` inside Docker!) in 10 seconds.

What Topic Is Next → Next Topic 7.6: Environment Variables in Linux & Containers

7.6 Environment Variables in Linux & Containers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `export KEY=VAL` • `env`/`printenv` • Spring Boot relaxed binding (`SPRING_DATASOURCE_URL` -> `spring.datasource.url`) • Production Pitfall Guard (Environment Variables in Linux & Containers) • Hands-On Verification (Environment Variables in Linux & Containers) • Environment Variables in Linux & Containers — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Environment Variables in Linux

Purpose: Overriding any `application.yml` key in Docker/Linux simply by setting uppercase underscore environment variables!

What Topic Is Next → Next Topic 7.7: Docker and Containerization Fundamentals

7.7 Docker and Containerization Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Containers (shared Linux kernel + cgroups/namespaces isolation • MBs • Sub-second start) vs traditional VMs (full guest OS • GBs) • Production Pitfall Guard (Docker and Containerization Fundamentals) • Hands-On Verification (Docker and Containerization Fundamentals) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker

Purpose: Packaging Java 21 + exact OS libraries + JAR into an immutable image that runs identically locally, in CI, and in cloud prod.

What Topic Is Next → Next Topic 7.8: Docker Images and Layer Caching

7.8 Docker Images and Layer Caching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-only union filesystem layers • Tags (`eclipse-temurin:21-jre-alpine` vs mutable `:latest`) • Layer cache invalidation order • `.dockerignore`! • Production Pitfall Guard (Docker Images and Layer Caching) • Hands-On Verification (Docker Images and Layer Caching) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker Images

Purpose: Copying `pom.xml` and downloading Maven dependencies in a cached layer BEFORE `COPY src ./src` so code-only rebuilds finish in seconds!

What Topic Is Next → Next Topic 7.9: Running and Managing Containers

7.9 Running and Managing Containers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Port publishing `-p 8080:8080` (`host:container`) • Detached `-d` • `--name` • `-e` / `--env-file` • Production Pitfall Guard (Running and Managing Containers) • Hands-On Verification (Running and Managing Containers) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Running

Purpose: Mapping host port `8081` to container port `8080` (`-p 8081:8080`) cleanly without rebuilding the JAR.

What Topic Is Next → Next Topic 7.10: Authoring Production Java 21 Multi-Stage Dockerfiles

7.10 Authoring Production Java 21 Multi-Stage Dockerfiles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stage 1 (`maven:3.9-eclipse-temurin-21` compiles JAR) -> Stage 2 (`eclipse-temurin:21-jre-alpine` copies only JAR • Runs as non-root `USER` • `ENTRYPOINT ["java" • "-XX:MaxRAMPercentage=75.0" • Production Pitfall Guard (Authoring Production Java 21 Multi-Stage Dockerfiles) • Hands-On Verification

(Authoring Production Java 21 Multi-Stage Dockerfiles) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Authoring Production Java 21 Multi-Stage Dockerfiles

Purpose: Shrinking image size from ~750MB (JDK + Maven cache) down to ~190MB (slim JRE + JAR) with smaller attack surface and proper `SIGTERM` handling!

What Topic Is Next → Next Topic 7.11: Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)

7.11 Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `docker build -t` • `docker ps -a` • `docker logs -f --tail 100` • `docker exec -it <container> sh` • Production Pitfall Guard (Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)) • Hands-On Verification (Essential Docker CLI Commands (build, run, ps, logs, exec, stats, stop)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Essential Docker CLI Commands (build, run, ps, logs,...

Purpose: Inspecting startup failure logs (`docker logs`) and container memory usage (`docker stats`) rapidly.

What Topic Is Next → Next Topic 7.12: Docker Volumes and Persistent Database Storage

7.12 Docker Volumes and Persistent Database Storage [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why container writable layers vanish on `docker rm` • Named volumes (`pg_data:/var/lib/postgresql/data`) vs bind mounts • Production Pitfall Guard (Docker Volumes and Persistent Database Storage) • Hands-On Verification (Docker Volumes and Persistent Database Storage) • Docker Volumes and Persistent Database Storage — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker Volumes • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Persisting PostgreSQL tables and Kafka logs across container restarts and image upgrades via named Docker volumes.

What Topic Is Next → Next Topic 7.13: Docker Container Networking

7.13 Docker Container Networking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: User-defined bridge networks • Built-in Docker DNS resolution by service/container name (`jdbc:postgresql://postgres:5432/jobportal` — NOT `localhost`!) • Production Pitfall Guard (Docker Container Networking) • Hands-On Verification (Docker Container Networking) • Docker Container Networking — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker Container Networking • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Connecting `order-service` to `postgres:5432`, `redis:6379`, and `kafka:9092` via Docker DNS service names.

What Topic Is Next → Next Topic 7.14: Docker Compose for Multi-Container Applications

7.14 Docker Compose for Multi-Container Applications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `compose.yaml` (`services` • `networks` • `volumes` • `environment` • Production Pitfall Guard (Docker Compose for Multi-Container Applications) • Hands-On Verification (Docker Compose for Multi-Container Applications) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker Compose for Multi-Container Applications

Purpose: Using `depends_on` with `condition: service_healthy` (`pg_isready`) so Spring Boot waits until PostgreSQL is genuinely accepting SQL connections before booting!

What Topic Is Next → Next Topic 7.15: Containerized PostgreSQL Setup and Init Scripts

7.15 Containerized PostgreSQL Setup and Init Scripts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `postgres:16-alpine` • `POSTGRES_DB`/`USER`/`PASSWORD` • `pg_isready` healthcheck • `/docker-entrypoint-initdb.d/` multi-database bootstrap SQL • Production Pitfall Guard (Containerized PostgreSQL Setup and Init Scripts) • Hands-On Verification (Containerized PostgreSQL Setup and Init Scripts) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Containerized PostgreSQL Setup

Purpose: Initializing isolated databases (`user_db`, `job_db`, `application_db`) cleanly inside one local dev PostgreSQL container.

What Topic Is Next → Next Topic 7.16: Multi-Service Spring Boot + Kafka + Redis Compose Stack

7.16 Multi-Service Spring Boot + Kafka + Redis Compose Stack [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Orchestrating Eureka • Gateway • Microservices • PostgreSQL • Production Pitfall Guard (Multi-Service Spring Boot + Kafka + Redis Compose Stack) • Hands-On Verification (Multi-Service Spring Boot + Kafka + Redis Compose Stack) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Multi-Service Spring Boot + Kafka + Redis Compose St...

Purpose: Booting the complete Final Capstone architecture locally with a single `docker compose up --build` command.

What Topic Is Next → Next Topic 7.17: CI/CD Fundamentals (Continuous Integration & Continuous Delivery)

7.17 CI/CD Fundamentals (Continuous Integration & Continuous Delivery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automated build + test + lint + container build on every PR (CI) vs automated staging/production release deployment (CD) • Fast feedback loops • Production Pitfall Guard (CI/CD Fundamentals (Continuous Integration & Continuous Delivery)) • Hands-On Verification (CI/CD Fundamentals (Continuous Integration & Continuous Delivery)) • CI/CD Fundamentals (Continuous Integration & Continuous Delivery) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for CI/CD Fundamentals (Continuous Integration & Continu... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Blocking broken code or failing JUnit tests from ever merging into `main`.

What Topic Is Next → Next Topic 7.18: GitHub Actions Workflow Fundamentals

7.18 GitHub Actions Workflow Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.github/workflows/ci.yml` • Triggers (`on: [push • Pull_request]`) • Ubuntu runners • Production Pitfall Guard (GitHub Actions Workflow Fundamentals) • Hands-On Verification (GitHub Actions Workflow Fundamentals) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for GitHub Actions Workflow Fundamentals

Purpose: Getting free, automatic green/red PR status checks on every student and team GitHub repository.

What Topic Is Next → Next Topic 7.19: Automated Build, Test and Package Pipeline

7.19 Automated Build, Test and Package Pipeline [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `./mvnw -B verify` in GitHub Actions (Docker is pre-installed on GitHub Ubuntu runners so Testcontainers works out of the box!) • Uploading JaCoCo/Surefire artifacts • Building Docker image • Production Pitfall Guard (Automated Build, Test and Package Pipeline) • Hands-On Verification (Automated Build, Test and Package Pipeline) • Automated Build, Test and Package Pipeline — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Automated Build

Purpose: Executing unit tests,PostgreSQL Testcontainers integration tests, JAR packaging, and `docker build` validation on every Pull Request.

What Topic Is Next → Next Topic 7.20: Maven Build Lifecycle Mastery (clean, compile, test, package, verify)

7.20 Maven Build Lifecycle Mastery (clean, compile, test, package, verify) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Standard Maven phases (`validate -> compile -> test -> package -> integration-test -> verify -> install`) • Surefire (`*Test`) vs Failsafe (`*IT`) • Multi-module parent `pom.xml` basics • Production Pitfall Guard (Maven Build Lifecycle Mastery (clean, compile, test, package, verify)) • Hands-On Verification (Maven Build Lifecycle Mastery (clean, compile, test, package, verify)) • Maven Build Lifecycle Mastery (clean, compile, test, package, verify) — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Maven Build Lifecycle Mastery (clean, compile, test,...

Purpose: Sharing common Java 21, Spring Boot 3, and Spring Cloud versions cleanly across microservices via a root parent `pom.xml`.

What Topic Is Next → Next Topic 7.21: Deploying a Spring Boot Backend Application

7.21 Deploying a Spring Boot Backend Application [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deploying containerized Spring Boot + managed PostgreSQL/Redis to a cloud VM / container platform (Render/Railway/VPS) • Reverse proxy TLS (Nginx/Caddy) • `/actuator/health` readiness probes • Production Pitfall Guard (Deploying a Spring Boot Backend Application) • Hands-On Verification (Deploying a Spring Boot Backend Application) • Deploying a Spring Boot Backend Application — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Deploying a Spring Boot Backend Application

Purpose: Publishing a live HTTPS URL + Swagger UI for your Capstone so interviewers can test your live API during technical rounds.

What Topic Is Next → Next Topic 7.22: Production Profiles and Environment Configuration

7.22 Production Profiles and Environment Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SPRING_PROFILES_ACTIVE=prod` • Disabling `show-sql` & verbose debug logs • Setting `ddl-auto: validate` • Configuring CORS allowed frontend origins • Production Pitfall Guard (Production Profiles and Environment Configuration) • Hands-On Verification (Production Profiles and Environment Configuration) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Production Profiles

Purpose: Ensuring production boot uses validated schemas, structured logs, and bounded heap (`-XX:MaxRAMPercentage=75.0`).

What Topic Is Next → Next Topic 7.23: Secrets Management and Secure Configuration Practices

7.23 Secrets Management and Secure Configuration Practices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: GitHub Actions Encrypted Secrets (`${{ secrets.DB_PASSWORD }}`) • Docker Compose `.env` + `.env.example` • Rotating leaked keys immediately • Never baking secrets into `Dockerfile` `ENV`/`ARG` layers! • Production Pitfall Guard (Secrets Management and Secure Configuration Practices) • Hands-On Verification (Secrets Management and Secure Configuration Practices) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Secrets Management

Purpose: Injecting `JWT_SECRET`, `DB_PASSWORD`, and registry tokens at runtime only so `docker inspect` / image layers never leak credentials.

What Topic Is Next → Module 7 Capstone (Week 22 Lab Build: Multi-Stage Docker Compose & GitHub Actions CI Pipeline) → Next Topic 8.1: Job Portal Domain Modeling, Bounded Contexts & ER Schema Design

Module 7 Hands-On Capstone Project (100 Marks): Week 22 Lab Build: Multi-Stage Docker Compose & GitHub Actions CI Pipeline — Containerize a Spring Boot 3 + PostgreSQL + Redis + Kafka stack using a non-root multi-stage Java 21 `Dockerfile`, health-checked `docker-compose.yml`, and automated `.github/workflows/ci.ym...

MODULE 8 OF 8 • STAGE 2 • 12 LIVE + LAB HOURS • 10 TOPICS (80 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Final Industry Capstone Project and Placement Readiness

Design, implement, test, containerize, document, deploy, and defend the Final Industry Capstone — a distributed Job Portal Backend System built with Java 21, Spring Boot 3, Spring Security/JWT, Spring Data JPA, PostgreSQL, Spring Cloud Gateway, Eureka, Apache Kafka, Redis, JUnit 5/Mockito, Swagger/OpenAPI, and Docker Compose.

Objectives: ✓ Architect and build all core business modules of the Job Portal Backend System: Auth/User, Candidate, Employer, Job, Application, and Notification services. • ✓ Implement JWT authentication, RBAC (`CANDIDATE`, `EMPLOYER`, `ADMIN`), ownership guards, and dynamic paginated/cached Job Search APIs. • ✓ Connect asynchronous application lifecycle notifications (`ApplicationSubmitted`, `ApplicationShortlisted`, `InterviewScheduled`) over Apache Kafka with idempotency and DLQ. • ✓ Package the complete system with JUnit 5/Mockito/Testcontainers tests, OpenAPI/Swagger docs, Postman collection, ER & Architecture diagrams, and `docker-compose.yml`. • ✓ Deploy and deliver a live end-to-end Capstone demonstration and technical viva defense for placement readiness.

8.1 Job Portal Domain Modeling, Bounded Contexts & ER Schema Design [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing schemas • Constraints • Composite indexes for Users • Candidates (skills/resume metadata) • Production Pitfall Guard (Job Portal Domain Modeling, Bounded Contexts & ER Schema Design) • Hands-On Verification (Job Portal Domain Modeling, Bounded Contexts & ER Schema Design) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Job Portal Domain Modeling

Purpose: Preventing duplicate job applications at the database level via `UNIQUE (job_id, candidate_id)` and indexing `(status, location, min_salary)`.

What Topic Is Next → Next Topic 8.2: User & Authentication Service (Registration, Login, JWT & RBAC)

8.2 User & Authentication Service (Registration, Login, JWT & RBAC) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: BCrypt registration for `CANDIDATE` • `EMPLOYER` accounts • JWT access/refresh token issuance • Account status checks • Production Pitfall Guard (User & Authentication Service (Registration, Login, JWT & RBAC)) • Hands-On Verification (User & Authentication Service (Registration, Login, JWT & RBAC)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for User

Purpose: Enforcing strict role separation so only `EMPLOYER`s post jobs and only `CANDIDATE`s submit applications.

What Topic Is Next → Next Topic 8.3: Candidate & Employer Profile Services

8.3 Candidate & Employer Profile Services [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Candidate profile • Education • Experience years • Skill tags & resume metadata • Production Pitfall Guard (Candidate & Employer Profile Services) • Hands-On Verification (Candidate & Employer Profile Services) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Candidate

Purpose: Ensuring Employer A can update ONLY Employer A's company profile and job postings.

What Topic Is Next → Next Topic 8.4: Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)

8.4 Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Job posting • Status lifecycle (`DRAFT` • `OPEN` • `CLOSED`) • Production Pitfall Guard (Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)) • Hands-On Verification (Job Service & Dynamic Search Engine (Filtering, Sorting & Pagination)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Job Service

Purpose: Supporting `GET /api/v1/jobs?q=java&location=Bengaluru&workMode=HYBRID&minSalary=600000&page=0&size=20&sort=newest`.

What Topic Is Next → Next Topic 8.5: Application Service & Status Workflow

8.5 Application Service & Status Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Applying to an `OPEN` job (`Idempotency-Key` + `UNIQUE(job_id • Candidate_id)`) • Tracking states (`APPLIED -> UNDER_REVIEW -> SHORTLISTED -> INTERVIEW -> OFFERED / REJECTED`) • Employer vs candidate views • Production Pitfall Guard (Application Service & Status Workflow) • Hands-On Verification (Application Service & Status Workflow) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Application Service

Purpose: Validating via `job-service` (OpenFeign + Resilience4j) that a job is `OPEN` before accepting a candidate's application.

What Topic Is Next → Next Topic 8.6: Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)

8.6 Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Publishing `ApplicationSubmittedEvent` & `ApplicationStatusChangedEvent` (keyed by `applicationId`) • `notification-service` `@KafkaListener` with idempotency table • `@RetryableTopic` • `.DLT` • Production Pitfall Guard (Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)) • Hands-On Verification (Kafka Notification Pipeline (Asynchronous Events, Retry & DLQ)) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Kafka Notification Pipeline (Asynchronous Events, Re...

Purpose: Notifying employers on new applications and candidates on shortlist/interview updates asynchronously without slowing down API responses.

What Topic Is Next → Next Topic 8.7: API Gateway, Eureka Discovery & Redis Caching Integration

8.7 API Gateway, Eureka Discovery & Redis Caching Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Routing all external requests through `api-gateway` (`8080`) + Eureka (`8761`) • Redis `@Cacheable` on hot `GET /api/v1/jobs/{id}` with `@CacheEvict` on job edits • Gateway Redis rate limiting • Production Pitfall Guard (API Gateway, Eureka Discovery & Redis Caching Integration) • Hands-On Verification (API Gateway, Eureka Discovery & Redis Caching Integration) • API Gateway, Eureka Discovery & Redis Caching Integration — Interview Readiness • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for API Gateway

Purpose: Protecting login/apply endpoints with Redis rate limits and serving hot job detail views in < 2ms.

What Topic Is Next → Next Topic 8.8: Comprehensive Test Suite & OpenAPI Documentation

8.8 Comprehensive Test Suite & OpenAPI Documentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: JUnit 5 + Mockito service unit tests • `@WebMvcTest` RBAC/validation tests • PostgreSQL `Testcontainers` integration tests • `springdoc-openapi` Swagger UI across services • Production Pitfall Guard (Comprehensive Test Suite & OpenAPI Documentation) • Hands-On Verification (Comprehensive Test Suite & OpenAPI Documentation) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Comprehensive Test Suite

Purpose: Proving to technical interviewers with one `./mvnw verify` run that your Capstone is genuinely engineered, not a fragile demo.

What Topic Is Next → Next Topic 8.9: Docker Compose Packaging, GitHub Actions CI & Cloud Deployment

8.9 Docker Compose Packaging, GitHub Actions CI & Cloud Deployment [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage Java 21 `Dockerfile`s • Root `docker-compose.yml` with healthchecks (`Postgres` • `Redis` • `Kafka` • Production Pitfall Guard (Docker Compose Packaging, GitHub Actions CI & Cloud Deployment) • Hands-On Verification (Docker Compose Packaging, GitHub Actions CI & Cloud Deployment) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Docker Compose Packaging

Purpose: Enabling any reviewer or recruiter to clone the repo, copy `.env.example` -> `.env`, and boot the entire system with `docker compose up -d`.

What Topic Is Next → Next Topic 8.10: Capstone Technical README, Postman Suite & Placement Viva Defense

8.10 Capstone Technical README, Postman Suite & Placement Viva Defense [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Polishing portfolio `README.md` (architecture diagram • ER diagram • Tech choices • Curl/Postman quickstart • Production Pitfall Guard (Capstone Technical README, Postman Suite & Placement Viva Defense) • Hands-On Verification (Capstone Technical README, Postman Suite & Placement Viva Defense) • Idiomatic JMSC Implementation: clean modular architecture, strict contracts & non-blocking patterns • Microservices with Java, Spring Cloud, Kafka, Redis & Docker Production Runtime Execution & Memory Lifecycle for Capstone Technical README

Purpose: Converting your 6-month engineering effort into concrete, verifiable resume bullets and confident technical interview performance.

What Topic Is Next → Module 8 Capstone → End-of-Course Live Client Project: Microservices with Java, Spring Cloud, Kafka, Redis & Docker — Enterprise Client Production Platform

Module 8 Hands-On Capstone Project (100 Marks): Final Industry Capstone: Job Portal Backend System (Distributed Java 21, Spring Boot 3, Kafka, Redis & Docker) — Design, engineer, test, containerize, deploy, and defend a portfolio-ready distributed Job Portal Backend System comprising `User Service`, `Candidate Service`, `Employer Service`, `Job Serv...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Microservices with Java, Spring Cloud, Kafka, Redis & Docker — Enterprise Client Production Platform • **Confirmed Role:** Microservices with Java, Spring Cloud, Kafka, Redis & Docker Engineering Project Intern

Domain & Production Infra: Live Enterprise Java Enterprise Backend & Microservices Engineering Client Engagement (Java 21 & Spring Boot 3, Microservices & Bounded Contexts, Spring Cloud Config & Eureka, Spring Cloud Gateway, OpenFeign & Resilience4j • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular Java 21 & Spring Boot 3, Microservices & Bounded Contexts, Spring Cloud Config & Eureka, Spring Cloud Gateway, OpenFeign & Resilience4j production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Microservices with Java, Spring Cloud, Kafka, Redis & Docker Developer, Associate Java Enterprise Backend & Microservices Engineering Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.5 LPA **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 LPA – ₹19.0 LPA **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 LPA – ₹38.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/java-microservices-kafka-docker-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.