Jai Singh

+91-8700420829 | jaisinghh49@gmail.com | linkedin.com/in/ja1-s1ngh | github.com/jai698 | leetcode.com/Ja1_S1ngh

Summary

Backend engineer with **1.8+ years** of production experience in **Java, Spring Boot, Apache Kafka and PostgreSQL**, building high-throughput, event-driven microservices for TripJack's hotel-supply platform. Track record of owning systems end to end: designed and shipped a Kafka ingestion service that replaced a legacy SQS pipeline, led zero-downtime migrations from a Java 8 monolith to Java 21 microservices, and delivered supplier API integrations across search, booking and post-booking.

Experience

TripJack Feb 2025 – Present
Software Development Engineer 1 *Connaught Place, New Delhi*

- **Built a Kafka-based ARI ingestion service from scratch (Java 21, Spring Boot 3.5, ∼12k LOC)**: Migrated the live availability / rates / inventory (ARI) pipeline for hotel channel managers off a legacy **Java 8 / Spring Boot 1.5 AWS SQS**-polling service onto **Java 21 / Spring Boot 3.5**. Per-supplier **Kafka** consumers (**STAAH, RateGain, HyperGuest, Synxis**) on a shared template/strategy base; supplier transformers normalise JSON and OTA-XML payloads into one unified rate-plan / inventory / room model in **PostgreSQL**. Failed events persisted to **MongoDB** for replay; payloads archived to **S3** on virtual threads; AOP-driven primary/replica datasource routing over **HikariCP** pools.
- **Zero-downtime SQS → Kafka cutover**: Strangler-fig migration: the legacy service forwarded each supplier's pushes to Kafka behind per-supplier feature flags, so consumers could be enabled and rolled back live. Cleared a backlog that had peaked at **10,000+ in-flight messages**, cut consumer lag ∼**70%**, and used per-hotel partition keys to preserve ordering, eliminating the rate-overwrite bugs of the old pull model.
- **Redesigned rules-engine reload — 14× faster (Drools, 13,000 rules)**: The pricing / commission rules engine recompiled all 13,000 business rules on every server every 20 minutes even when nothing had changed, causing latency spikes, stale and inconsistent results across servers, and 15–16 pods provisioned just to absorb the spike. Benchmarked the framework's built-in incremental-update options and showed them **4× slower** than a full rebuild; instead designed a sharded approach — rules split into independent groups, each group content-hashed, and only changed groups recompiled and swapped in atomically. Measured on production-shaped data: reload **3.9 s → 278 ms**, classes loaded **26k → 1.6k**, rules evaluated per request **13k → 812**. Also fixed the non-deterministic rule generation and two reload race conditions uncovered during the redesign.
- **Monolith → microservice migration of extranet management APIs**: Moved the hotel user-management, rates, inventory and rate-plan CRUD API families out of the legacy monolith into a Spring Boot 3 service with controller → validator → orchestrator → persistence layering, wire-format tests written first and documented legacy-parity diffs. Added read-time **fallback between the new and legacy rate / inventory schemas** so hotels moved over incrementally with no big-bang cutover.
- **CRS & channel-manager integrations**: Integrated **Sterling CRS** (search, detail, price-check) and hardened the **Synxis, Jumeirah, HyperGuest, STAAH, RateGain** adapters — occupancy-wise rate de-duplication, cancellation-policy and currency handling, inventory-only push semantics. Built a batched async job that pulls hotel and room static content from four suppliers and reconciles supplier-to-hotel mappings.
- **Search, pricing & post-booking features**: **Elasticsearch** city / hotel search API with a pluggable type-handler registry and geo-location validation; promotion outputs (free nights, early-booker, last-minute) and direct-contract markup rules; **virtual credit card (VCC)** issuance and **OTP** APIs for post-booking; rules-engine client with null-safe fallbacks.
- **Observability & logging cost**: Kibana analytics for search / detail / review, a room-filtering summary logging why each room was excluded, MDC correlation IDs through Kafka headers, JSON structured logs. Replaced EC2-disk logging with centralised **ELK** and moved large payloads to S3, cutting logging spend ∼**40% ($4,000+/month)**.
- **Caching & testing**: **Aerospike** caching for hot supplier data with per-request TTLs, reducing redundant downstream calls ∼60%. Authored **58 JUnit 5 / Mockito test classes** (unit, wire-format and recorded-response tests); contributed to the search service and the **gRPC** supplier-connector service.

Technical Skills

Languages: Java 8 / 21, SQL, JavaScript, C++
Backend: Spring Boot 1.5 / 3.5, Spring Data JPA / Hibernate, Spring Security, Spring Kafka, Spring AOP, REST, gRPC / Protobuf, Resilience4j, Drools, Lombok, CompletableFuture / Virtual Threads
Data & Messaging: PostgreSQL (primary/replica, HikariCP), MongoDB, Aerospike, Elasticsearch, Apache Kafka (AWS MSK), AWS SQS
AWS & Infra: EC2, S3, RDS, IAM, KMS, ECS / EKS, Nginx, Docker, Kubernetes (Devtron), Jenkins, Gradle, Git
Observability & Testing: ELK Stack / Kibana, structured logging, JUnit 5, Mockito, MockMvc, contract tests, JVM benchmarking
Concepts: Event-Driven Architecture, Microservices, Strangler-Fig Migration, Distributed Systems, System Design, DSA

Education

Jaypee University of Information Technology Waknaghat, Solan
B.Tech in Computer Science *2020 – 2024*