

# Advanced Java: Microservices, Streaming and MCP Servers

## An intensive two-day design and implementation lab for experienced engineers

FA-0548 · 2 days

Software Development · DevOps, Cloud & Infrastructure · Cybersecurity · Agentic & Generative AI

Implement and review a focused Java service and MCP integration while examining streaming, security and operational trade-offs.

---

## Introduction

This two-day advanced lab is for engineers already comfortable with Java, Linux, microservices and messaging. It connects JVM concurrency and API design with streaming semantics and a focused Java MCP server integration for AI-tool clients.

Participants implement selected service and tool-server paths, inspect failures and review contracts, security boundaries and observable behaviour. Broader production engineering topics are design clinics and follow-on practice, not a promise to build or certify a complete production-grade platform in two days.

Deliverables are reviewed for repeatable builds, explicit configuration, validation, an API contract, basic threat-model notes and an observability baseline. The course is a technical workshop, not a graded professional certification.

## Learning outcomes

The course teaches engineers to:

- Apply Java 21 service-design and concurrency concepts and recognise preview-feature constraints.
- Design API boundaries with validation, error handling, idempotency and bounded work.
- Reason about streaming ordering, replay, delivery semantics and consumer resilience.
- Build a focused Java MCP server exposing constrained tools or resources through an interoperable SDK/protocol revision.

- Apply appropriate transport, token, authorisation and least-privilege controls to the sample service.
- Inspect logs, metrics and failure behaviour, then identify additional testing and operational work needed for production.

## Prerequisites

- Write and debug non-trivial Java services without assistance (JVM internals familiarity strongly recommended).
- Use Linux fluently: networking tools, systemd basics, process/FD inspection, cgroups/container fundamentals, shell proficiency.
- Be comfortable with: HTTP, TLS, JSON, concurrency, basic distributed systems concepts, Git, and CI workflows.
- Have prior exposure to microservices and messaging (Kafka/Pulsar/RabbitMQ equivalent concepts).
- Bring a working Java 21 Linux environment with: build tooling (Maven/Gradle), containers (Docker/Podman), and an IDE/editor.

Use a Java MCP SDK release compatible with the chosen JDK and protocol revision; verify client/server interoperability before the lab. Preview Java features are discussed separately and are not required as production defaults.

## Training outline

### 1. Day 1 — JVM Services, Contracts and Streaming

#### Java 21 Runtime and Concurrency

Language and runtime features that change service design

- Records as contract carriers and boundary objects
- Sealed types for constrained domain modeling
- Pattern matching usage for safer dispatch paths

Concurrency in modern Java services

- Virtual threads: where they help, where they hurt
- Structured concurrency in Java 21: conceptual model, preview status and adoption constraints; not a stable production default.
- Thread-per-request vs reactive pipelines: decision framework
- Synchronization and contention hotspots in real services

JVM performance and reliability essentials

- GC selection and service-level implications

- Warmup, JIT behavior, and latency cliffs
- Allocation discipline for high-throughput endpoints
- Safe timeouts, cancellation, and resource cleanup patterns

## Microservice Architecture

Service boundary definition under real constraints

- domain boundaries and coupling controls
- shared libraries vs shared schemas vs shared services
- dependency direction rules and “no cycles” enforcement

Communication patterns and trade-offs

- synchronous (HTTP/gRPC) vs asynchronous (events/streams)
- fan-out strategies and failure amplification control
- data consistency and correctness strategy selection

Data ownership and schema evolution

- contract-first thinking
- backward/forward compatibility policies
- migration patterns that avoid flag days

## Endpoint Engineering

REST endpoint design (production-grade)

- resource modeling and consistent naming
- pagination, filtering, sorting, and query complexity limits
- partial responses and field projections
- versioning strategies and deprecation mechanics

Request/response correctness

- validation layers and error taxonomy
- idempotency keys and replay handling
- correlation IDs and request provenance
- rate limiting, quotas, and abuse resistance

Input/output safety

- canonicalization rules
- payload size limits
- content-type enforcement
- safe serialization practices

High-performance endpoint implementation patterns

- concurrency model selection per endpoint type
- timeout budgets and deadline propagation
- backpressure approaches for overloaded systems

## Streaming Correctness

Streaming fundamentals that matter in production

- ordering guarantees and partitioning strategy
- consumer group design and scaling dynamics
- offset management and replay strategy

Delivery semantics and correctness

- at-most-once / at-least-once / effectively-once goals
- idempotent processing and deduplication strategies
- transactional outbox and inbox patterns

Resilience in stream processing

- retry policies and poison message handling
- DLQ design and operational workflow
- backpressure and load shedding

Schema management and evolution

- schema registry approaches
- compatibility rules and enforcement
- event versioning and translator patterns

Observability for streams

- lag, throughput, error rates, DLQ volume
- per-key hotspots and partition skew diagnosis

## 2. Day 2 — Secure MCP Integration and Operational Review

### Security Baseline

Threat modeling as an engineering input

- attack surface inventory per service
- trust boundaries and data classification
- abuse cases for endpoints and streaming consumers

Authentication and authorization

- OAuth2/OIDC concepts applied to microservices
- JWT validation correctness (audience, issuer, clock skew, rotation)
- scopes/roles/claims mapping to authorization decisions
- service-to-service auth patterns

#### Transport and service identity

- TLS/mTLS basics for production deployments
- cert rotation strategies
- zero-trust posture for internal networks

#### Secrets management

- environment vs file mounts vs secret stores
- rotation and blast-radius reduction
- least privilege configuration

#### API security controls

- input validation hardening
- rate limiting and bot resistance
- safe error reporting (no data leaks)

#### Supply chain and runtime hardening

- dependency controls and signing posture
- container image hardening and minimal base images
- runtime permissions, seccomp/apparmor concepts

## Java MCP Server Lab

MCP model for backend engineers: select an interoperable protocol revision and consult the corresponding SDK contracts.

- Servers, clients, request lifecycles and transport/version compatibility; earlier session-based and current request-based revisions differ.
- tools vs resources vs prompts (capability boundaries)
- capability discovery and contracts

#### MCP server architecture and service layering

- protocol layer vs domain layer separation
- tool execution model and cancellation/timeout handling
- deterministic outputs and safe error models

#### Transport strategies (Linux-first operational view)

- stdio transport for local/agent integrations
- Streamable HTTP for networked deployments; distinguish it from legacy HTTP/SSE implementations when supporting older clients.

- Transport-specific lifecycle, cancellation and shutdown handling; avoid assuming one session model across protocol revisions.

#### Authorization and access control for MCP

- HTTP authorisation and resource-server token boundaries according to the selected MCP specification; local stdio uses a different credential boundary.
- scoping access to tools/resources
- per-tool authorization rules and auditing

#### Security risks specific to MCP-style tool servers

- tool chaining risk containment
- prompt injection-aware design constraints
- filesystem/network access minimization
- safe argument handling and validation discipline

#### Observability and governance for MCP servers

- tool invocation logging with redaction
- per-tool metrics (latency, failures, volume)
- trace correlation with upstream requests
- audit trails and retention considerations

#### Packaging and distribution

- versioning and compatibility commitments
- configuration profiles per environment
- running as a Linux service (systemd/container)

#### Reliability engineering for MCP servers

- isolation of tool execution
- concurrency caps and queueing strategy
- circuit breakers for downstream dependencies
- safe degradation and “deny-by-default” behavior

## Composing Services, Streams and Tools

#### Exposing microservice capabilities as MCP tools safely

- tool granularity and permission design
- rate limiting and quota enforcement for tools
- preventing “LLM as a traffic amplifier” failure modes

#### Event-driven tool workflows

- tools that publish events vs tools that query state
- ensuring idempotency and auditability

Data access strategy

- MCP resources as read-only surfaces where possible
- write operations with explicit confirmations and constraints
- segregation of duties between read and write capabilities

## Observability and Review

Logging that supports incident response

- structured logging discipline
- correlation IDs and trace propagation
- redaction rules and sensitive fields handling

Metrics that answer operational questions

- golden signals (latency, traffic, errors, saturation)
- endpoint-level and consumer-level SLIs/SLOs
- business and domain metrics separation

Distributed tracing

- span modeling and cardinality discipline
- sampling strategy and tail-based considerations

Alerting and runbooks

- alert design to avoid noise
- escalation paths and triage steps

## Linux Operation: Design Clinic and Follow-On Practice

Process, memory, and file descriptor discipline

- ulimit strategy and failure modes
- ephemeral port exhaustion patterns
- diagnosing blocked threads and stalled IO

Systemd as a real deployment target (even when using containers)

- unit design for services
- health signaling
- logging integration
- restart policy correctness

Container runtime realities

- cgroups CPU/memory constraints and JVM tuning implications

- immutable images, minimal attack surface
- configuration injection patterns

# Resilience, Testing and Delivery: Follow-On Practice

## Resilience and Partial Failure

### Timeouts and retries

- retry budgets and exponential backoff discipline
- jitter and thundering herd avoidance
- distinguishing retryable vs non-retryable failures

### Circuit breakers, bulkheads, and load shedding

- protecting dependencies and self-protection
- per-endpoint concurrency caps

### Data correctness under partial failure

- compensating actions
- saga patterns (or alternatives) with clear invariants

### Chaos and fault injection mindset

- failure mode inventory
- Document and test the intended behaviour during dependency outages rather than promise untested failure guarantees.

## Test Strategy

### Contract tests and compatibility enforcement

- consumer-driven contracts for endpoints
- event schema compatibility gates

### Integration testing with real dependencies

- ephemeral environments
- deterministic test data and cleanup strategy

### Performance testing essentials

- latency percentiles and tail behavior
- soak testing and resource leak detection

### Security testing essentials

- authz bypass checks

- input fuzzing mindset
- dependency vulnerability response workflow

## Delivery and Operational Readiness

### Build and release discipline

- reproducible builds and artifact provenance
- versioning and changelog expectations

### Configuration management

- environment-specific config without code forks
- safe defaults and explicit overrides

### Deployment strategies

- rolling, blue/green, canary basics
- rollback safety and DB migration discipline

### Operational readiness review checklist for further production work.

- dashboards, alerts, runbooks
- on-call handoff quality bar
- post-incident learning loop