ENGINEER-TO-ARCHITECT AND DURABLE THINKING SKILLS

Designing Microservices Architectures

Level: Practitioner • 2 days (expandable to 3) • Virtual, In-person

Overview

Microservices promise independent deployment, team autonomy, and scale, and they can deliver all three. They also turn every function call into a network call, every transaction into a distributed problem, and every deployment into an operational discipline. Most microservices failures are not technology failures; they are boundary failures, made by teams that split a system before understanding it.

This is a hands-on, practitioner course. It starts with the honest question most courses skip: whether you need microservices at all, and what a well-structured monolith gives up by comparison. From there it follows the gradient of decisions a designer actually faces: where to draw service boundaries, how services should communicate, how to manage data without shared databases, and how to keep a distributed system observable and resilient. We deliberately go deep on boundaries, communication, and data rather than surveying every tool in the ecosystem. Every module ends with a design lab, and each module builds on the one before, carrying one system through the whole course.

Who Should Attend

- Developers and technical leads designing or evolving service-based systems
- Architects deciding whether and how to decompose a monolith
- Engineers joining a microservices effort who want to understand the design decisions behind it

Learners who want a broader grounding in architecture first should take *Fundamentals of Software Architecture*.

Prerequisites

- Solid experience building server-side applications
- Familiarity with REST APIs and basic distributed concepts
- Comfort reading and discussing system diagrams

What You Will Learn

- Judge when microservices are worth their cost, and argue the monolith case honestly
- Identify service boundaries using domain analysis and coupling criteria
- Design inter-service communication, choosing between synchronous and event-driven styles
- Design data ownership per service and handle transactions and queries that span services
- Apply resilience patterns so one failing service does not take down the system
- Plan an incremental migration from a monolith without a risky rewrite

Course Outline

Day one: should you, and where to cut

- The Microservices Tradeoff
  - What microservices actually buy you: deployability, team autonomy, targeted scale

  - What they cost: distribution, operational load, and eventual consistency
  - The modular monolith as a serious alternative, not a consolation prize
  - Lab: given a case-study system and its constraints, argue for or against microservices and defend it
- Finding Service Boundaries
  - Decomposing by business capability and by subdomain
  - Bounded contexts as boundary candidates, and where DDD helps
  - Granularity: the forces that push services apart and the forces that pull them together
  - Lab: draw service boundaries for the case-study system and justify each cut
- Communication Between Services
  - Synchronous calls: REST and gRPC, and the coupling they create
  - Asynchronous messaging and events: decoupling at the price of complexity
  - Contracts, versioning, and not breaking your consumers
  - Lab: design the communication style for each interaction in your boundary map, with stated reasons

Day two: data, resilience, and getting there

- Data in a Distributed World
  - Database per service: why shared databases quietly recreate the monolith
  - Sagas and eventual consistency in place of distributed transactions
  - Queries that cross services: composition, CQRS, and read models
  - Lab: design the data ownership map and a saga for one multi-service transaction
- Resilience and Observability
  - Designing for partial failure: timeouts, retries, circuit breakers, and bulkheads
  - Observability: correlated logs, metrics, and traces across service hops
  - Deployment and platform concerns at a design level: containers, orchestration, and gateways
  - Lab: run a failure walkthrough of your design, find the cascading failure, and fix it
- Migration and Evolution
  - The strangler fig: carving services out of a monolith incrementally
  - Sequencing: which service to extract first and why
  - Knowing when to merge services back together
  - Lab: write a phased migration plan for the case-study monolith and present the first two steps

Extended Version

The three-day version keeps the same gradient and adds depth and a capstone:

- Deeper event-driven design: event storming the case study and designing around domain events
- Service mesh, API gateway, and platform tradeoffs in more depth
- Testing strategies for distributed systems: contract tests and testing in production safely
- A capstone: teams design a complete microservices architecture from a fresh brief and defend it in review