Spring Cloud Training: Microservices Architecture with Spring Boot

Introduction to the Spring Cloud training course

Presentation of the course objectives.

Understanding the role of Spring Cloud in the Spring ecosystem.

Identifying the limitations of a monolithic application.

Understanding when a microservices architecture is relevant.

Introduction to microservices

What is a microservice?

Monolith, modular monolith, and microservices.

Advantages and limitations of a microservices architecture.

Scalability, team autonomy, and independent deployment.

Network complexity, observability, security, and error management.

Splitting an application into microservices

Identifying business domains.

Understanding bounded contexts.

Defining the responsibilities of each service.

Avoiding shared databases.

Defining API contracts between services.

Hands-on workshop:

Analyzing an existing application.

Identifying potential services.

Defining service responsibilities and interactions.

Creating microservices with Spring Boot

Structure of a microservices project.

Creating multiple Spring Boot applications.

Exposing REST APIs.

Managing environment profiles.

Service-specific configuration.

Hands-on workshop:

Creating two Spring Boot microservices.

Setting up the first REST endpoints.

Configuring the services.

Communication between microservices

Synchronous communication with REST.

Using RestClient or WebClient.

Managing network errors.

Timeouts and error responses.

Propagating a correlation ID.

Hands-on workshop:

Making two microservices communicate.

Managing an error case between services.

Adding simple diagnostic logic.

Service discovery with Spring Cloud

Why use a service registry.

Introduction to Eureka.

Automatic service registration.

Service discovery.

Calling services by logical name.

Hands-on workshop:

Creating a Eureka Server.

Registering microservices.

Validating service discovery.

API Gateway with Spring Cloud Gateway

Role of an API Gateway.

Single entry point.

Routing to microservices.

Request and response filters.

Centralized management of cross-cutting behaviors.

Hands-on workshop:

Creating an API Gateway.

Configuring routes.

Exposing services through the Gateway.

Centralized configuration with Spring Cloud Config

Challenges of distributed configuration.

Principle of the Config Server.

Configuration by environment.

Managing shared and service-specific properties.

Hands-on workshop:

Creating a Config Server.

Centralizing the configuration of several services.

Changing configuration without modifying application code.

Microservices resilience

Understanding failures in distributed systems.

Timeouts, retries, and circuit breakers.

Introduction to Resilience4J.

Controlled service degradation.

Hands-on workshop:

Simulating a service failure.

Adding a resilience strategy.

Testing application behavior.

Asynchronous communication and events

Difference between synchronous and asynchronous communication.

Introduction to Kafka or an event-driven approach.

Producers, consumers, and business events.

Use cases in a microservices architecture.

Hands-on workshop:

Introduction or simple setup of an event-based exchange.

Publishing and consuming an event.

Microservices observability

Why observability is essential.

Application logs.

Correlation ID.

Health checks.

Metrics.

Introduction to Micrometer, Prometheus, Grafana, or equivalent tools.

Hands-on workshop:

Enabling health endpoints.

Reading logs from multiple services.

Following a request across several components.

Testing in a microservices architecture

Unit tests.

Integration tests.

API tests.

Contract validation between services.

Testing strategies adapted to distributed architectures.

Hands-on workshop:

Testing a REST endpoint.

Testing communication between services.

Validating an end-to-end scenario.

Local deployment with Docker Compose

Containerizing services.

Organizing a local environment.

Application services, databases, Gateway, and registry.

Hands-on workshop:

Running the microservices architecture with Docker Compose.

Validating the complete startup of the environment.

Testing an end-to-end scenario.