Theory On Microservices

1. Microservices Fundamentals

Core Concepts

- Service Decomposition: Breaking down applications into small, independent services
- Bounded Context: Each service owns its data and domain logic (DDD principle)
- Autonomous Services: Independently deployable and scalable
- Polyglot Persistence: Different services can use different databases
- Resilience: Services should handle failures gracefully

Benefits

- Improved scalability
- Faster development cycles
- Technology flexibility
- Better fault isolation
- Easier maintenance

Challenges

- Distributed system complexity
- Data consistency
- Network latency
- Testing difficulties
- Operational overhead

2. Service Communication Patterns

Synchronous Communication

- REST API: HTTP/HTTPS with JSON/XML
- gRPC: High-performance RPC framework
- GraphQL: Flexible query language for APIs

Asynchronous Communication

- Message Brokers: RabbitMQ, Kafka
- Event Sourcing: Capture all changes as events
- CQRS: Separate read and write models

Service Discovery

- Client-side: Eureka, Consul
- Server-side: Kubernetes services, Load balancers

3. API Gateway Pattern

Key Responsibilities

- Routing: Direct requests to appropriate services
- Aggregation: Combine results from multiple services
- Offloading: Handle cross-cutting concerns:
    - Authentication/Authorization
    - Rate limiting
    - Caching
    - Request/Response transformation
    - Circuit breaking

Spring Cloud Gateway Features

- Route predicates and filters
- Integration with service discovery
- Reactive programming model
- Built-in resilience patterns

4. Resilience Patterns

Circuit Breaker

- Closed State: Normal operation
- Open State: Short-circuit requests when failures exceed threshold
- Half-Open State: Trial requests to check if service recovered

Other Patterns

- Retries: Transient fault handling

- Bulkheads: Resource isolation
- Rate Limiting: Prevent overload
- Fallbacks: Graceful degradation

Resilience4J vs Hystrix

- Resilience4J is the modern replacement
- Functional programming style
- Lightweight and modular
- Better metrics and monitoring

5. Distributed Data Management

Database per Service

- Each service owns its data schema
- Prevents tight coupling
- Enables technology diversity

Transaction Management

- Saga Pattern: Sequence of local transactions
- Eventual Consistency: Accept temporary inconsistency
- Two-Phase Commit: Complex but strong consistency

6. Observability

Essential Components

- Logging: Centralized log aggregation
- Metrics: Time-series monitoring
- Tracing: Distributed request tracking
- Health Checks: Service status monitoring

Spring Boot 3.0 Support

- Micrometer integration
- Actuator endpoints
- OpenTelemetry compatibility
- Prometheus and Grafana support

7. Security Considerations

Key Aspects

- Authentication: OAuth2, JWT
- Authorization: Role-based access
- Secure Communication: TLS/HTTPS
- Secrets Management: Vault, Config Server encryption

Spring Security 6.0 Features

- OAuth2 Resource Server
- Reactive security support
- Improved CSRF protection
- Modern password storage

8. Deployment Strategies

Common Approaches

- Containerization: Docker images
- Orchestration: Kubernetes, Docker Swarm
- Serverless: AWS Lambda, Azure Functions
- Blue-Green Deployment: Zero-downtime updates
- Canary Releases: Gradual rollout

CI/CD Integration

- Automated testing
- Pipeline as code
- Infrastructure as code
- GitOps principles