Comprehensive Notes on Microservices

Introduction to Microservices

Microservices are a software architectural style that structures an application as a collection of loosely coupled services. Each service is fine-grained and the protocols are lightweight.

Benefits:

- Scalability
- Flexibility in technology stack
- Easier deployment and maintenance

Comparison with Monolithic Architecture:

- Monolithic: Single codebase, tightly coupled
- Microservices: Modular, independently deployable components

Principles of Microservices

- Single Responsibility Principle: Each microservice should focus on a single business capability.
- Decentralized Data Management: Services manage their own data.
- Continuous Delivery: Enables frequent and safe releases.
- Scalability and Flexibility: Services can be scaled independently.

Challenges of Microservices

- Complexity Management: Harder to manage multiple services.
- Inter-Service Communication: Needs reliable, efficient communication mechanisms.
- Data Consistency: Handling consistency across distributed systems is challenging.
- Monitoring and Logging: Requires centralized and structured logging mechanisms.

Microservices Architecture Overview

Components:

- Service Discovery: Automatically detect services.
- API Gateway: Entry point for all clients.
- Load Balancer: Distributes traffic efficiently.

Comprehensive Notes on Microservices

Design Patterns

- Service Registry & Discovery: Manages dynamic service locations (e.g., Eureka).
- Circuit Breaker: Prevents cascading failures.
- API Composition: Aggregates results from multiple services.
- Saga Pattern: Manages distributed transactions with compensating actions.

Inter-Service Communication

- Synchronous: REST, gRPC ? real-time response.
- Asynchronous: Message Queues (RabbitMQ, Kafka), Event Streaming.
- Failures & Retries: Circuit breaker and retry strategies enhance resilience.

Data Management

- Database per Service: Each service owns its data.
- Event Sourcing: Store state as events.
- CQRS: Separates read and write operations for scalability.

Security

- Authentication & Authorization: Verifying identity and access levels.
- Secure Communication: HTTPS, OAuth2 for secure data transfer.
- Best Practices: Use JWT, API Gateway filtering, service-level authentication.

Spring Cloud for Microservices

Spring Cloud simplifies microservices development by integrating features like configuration management, service discovery, circuit breakers, etc.

- Components: Spring Cloud Config, Eureka, Zuul, Ribbon, Hystrix.
- Eureka: For service registration and discovery.

Spring Security for Microservices

Spring Security is a powerful framework to handle authentication and authorization.

Comprehensive Notes on Microservices

- Configure security for REST APIs.
- Supports OAuth2, JWT, role-based access.
- Best practices include securing endpoints and validating tokens.

Centralized Authentication and Authorization

- OAuth2/OIDC: Protocols for delegation and identity verification.
- Authorization Servers: Manage access tokens and user credentials.
- JWT: Compact, URL-safe tokens.
- SSO: One login for multiple systems, essential in microservices.