Lernzettel

Architekturstile verteilter Anwendungssysteme: Client-Server, 2-/3-Tier, Microservices, SOA, Event-Driven Architecture

Universität: Technische Universität Berlin
Kurs/Modul: Architektur von Anwendungssystemen
Erstellungsdatum: September 19, 2025

Lernzettel: Architekturstile verteilter Anwendungssysteme

(1) Überblick und Ziele. Verteilte Architekturstile beschreiben, wie Anwendungen über mehrere Rechner oder Prozesse hinweg verteilt laufen. Zentrale Merkmale sind lose Kopplung, Skalierbarkeit, Fehlertoleranz und die Trennung von Verantwortung. Typische Technologien umfassen Middleware, Remote Procedure Calls, Messaging Systeme, API Gateways und Web-APIs. Die Wahl des Stils beeinflusst Qualitätseigenschaften wie Leistung, Wartbarkeit, Sicherheit und Transaktionskontinuität.

(2) Client-Server Architektur. In der Client-Server-Architektur kommunizieren Clients direkt mit einem Server. Oft lässt sich der Server weiter in Presentation, Business Logic und Datenhaltung unterteilen, ist aber meist stärker zentriert.

Komponenten. Client (UI, Eingabevalidierung) – Server (Logik, Datenzugriff). **Kommunikation.** Request/Response über Netzwerkprotokolle wie HTTP. **Vorteile.** Zentralisierte Ressourcen, einfache Entwicklung von Desktop- oder Web-Anwendungen. **Nachteile.** Skalierung wird durch Serverleistung begrenzt, Engpässe beim Datenzugriff, Update Koordination notwendig.

(3) 2-Tier vs 3-Tier Architekturen. 2-Tier: Client-Application verbindet sich direkt mit der Datenbank auf dem Server. Business Logic liegt häufig im Client oder auf dem Server, wird aber nicht separat deployed. **3-Tier:** Eine zusätzliche Logikschicht (Application Server) trennt Präsentation, Logik und Datenhaltung. Vorteile: bessere Skalierung, Wartbarkeit; Nachteile: höhere Komplexität.

(4) Microservices. Eigenschaften: kleine, eigenständige Dienste mit eigener Datenhaltung, unabhängiges Deployment, klare Servicegrenzen. **Kommunikation.** REST, GraphQL, gRPC oder Messaging. **Vorteile.** Skalierbarkeit, isolierte Fehlertoleranz, unabhängige Teams. **Nachteile.** Verteilte Transaktionen, verteiltes Debugging, zusätzliche Governance nötig. **Architekturprinzipien.** Bindung über API-Verträge, lose Kopplung, schmale Services, Dezentralisierung der Daten. **Technologien.** API-Gateway, Service Registry, CI/CD, Observability Tools (Tracing, Logging, Metrics).

(5) Service Oriented Architecture (SOA). SOA fokussiert auf wiederverwendbare Services, die lose gekoppelt kommunizieren und durch Standards wie SOAP/REST, WSDL, Enterprise Service Bus (ESB) verbunden sind. **Vorteile.** Wiederverwendbarkeit, klare Schnittstellen, zentrale Governance. **Nachteile.** Oft schwergewichtig, ESB kann Flaschenhals werden, Overhead. **Beziehung zu Microservices:** SOA ist breiter, stärker governance gesteuert; Microservices ist eine agile Form mit kleineren Services.

(6) Event Driven Architecture (EDA). In EDA kommunizieren Komponenten asynchron über Events. Publisher erzeugen Events, Subscriber reagieren darauf. **Aspekte.** Messaging Middleware, Event Brokers (z.B. Kafka, RabbitMQ). **Vorteile.** Entkopplung, Skalierbarkeit, Resilienz. **Herausforderungen.** Konsistenz, Ordering, idempotente Verarbeitung, Fehlerbehandlung. **Pattern-Erweiterungen.** Event Sourcing, CQRS (Command Query Responsibility Segregation).

(7) Vergleich und Bewertung.

- Kriterien: Kopplung, Skalierbarkeit, Fehlertoleranz, Transaktionsmanagement, Wartbarkeit,

Sicherheit.

- Client-Server: geeignet bei einfachen, zentral kontrollierten Systemen.
- 2-Tier vs 3-Tier: 3-Tier ermöglicht bessere Skalierung und Wartbarkeit.
- Microservices: hohe Skalierbarkeit und Deploy-Flexibilität, aber verteilte Komplexität.
- SOA: gute Wiederverwendbarkeit bei größerer Governance, ESB-Architektur kann Flaschenhälse erzeugen.
- EDA: loses Koppeln, asynchrone Kommunikation; Herausforderungen bei Konsistenz, Ordering und Debugging.