Lernzettel

Serviceorientierte Architekturen und Microservices: Boundaries, API-Governance, Contract-First Design, API Versioning

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

Lernzettel: Serviceorientierte Architekturen und Microservices

(1) Ziel und Kontext. Serviceorientierte Architekturen (SOA) und Microservices fokussieren auf lose Kopplung, klare Boundary-Definitionen und orchestrierte bzw. choreografierte Interaktionen von Diensten. Die hier behandelten Themen umfassen Boundaries, API-Governance, Contract-First Design und API Versioning, um verteilte Anwendungen zuverlässig, wartbar und skalierbar zu entwerfen.

(2) Boundaries zwischen Services.

- Boundaries definieren klare Schnittstellen, Eigentums- und Verantwortungsbereiche.
- Typische Merkmale: eigenständige Deployments, eigene Datensätze, API-Schnittstellen als Vertrag.
- Wichtige Konzepte: bounded context (DDD), Datenisolierung, Anti-Korruption Layer beim Zusammenspiel mehrerer Kontexte.

(3) API-Governance.

- Ziele: Konsistenz, Interoperabilität, Sicherheit, Lebenszyklus-Management von APIs.
- Zentrale Elemente: API-Katalog, Richtlinien/Rollen (API-Designer, Reviewer, Produktverantwortliche), Policy-Management.
- Praktiken: Contract-Testing, Observability, Sicherheitstests, Versions- und Deprecation-Strategien.

(4) Contract-First Design.

- Vorgehen: Zuerst einen API-Vertrag entwerfen (OpenAPI/Swagger) und daraufhin Implementierung und Tests ableiten.
- Vorteile: stabile, gut dokumentierte Schnittstellen, Team-Parallelisierung, automatische Generierung von Client- und Server-Stubs.
- Typische Schritte: 1) API-Spezifikation entwerfen; 2) Implementierung an Vertrag anpassen; 3) Vertrags-Tests (Contract Tests); 4) Veröffentlichung des Vertrags.

(5) API Versioning.

- Strategien:
  - Pfadbasierte Versionierung: /v1/..., /v2/...
  - Header-basierte Versionierung: z. B. Accept-Version oder spezielle Request-Header
  - Medien-Typ-Versionierung: z. B. application/vnd.api.v2+json
- Deprecation und Migration: klare Deprecation-Policy, Migrationspfade, Client-Kompatibilität beachten.

- Empfehlungen: Versionierung sollte inkompatible Änderungen sichtbar machen; Koexistenz mehrerer Versionen möglich.

(6) Design- und Architektur-Pattern.

- API-Gateway und Service-Mesh als Eckpfeiler moderner Architekturen.
- API-Orchestrierung vs. API-Choreografie: zentrale Koordination vs. lose Kooperation.
- Zuverlässigkeit: Idempotenz, Timeouts, Retries, Circuit Breaker.
- Vertragssicherheit: Vorab-Validierung der Schemas, Regelmäßige Contract-Tests gegen OpenAPI-Spezifikationen.

(7) Bewertung und Qualitätsmerkmale.

- Kopplung vs. Kohäsion der Services; klare Ownership.
- Autonomie der Deployments; Datenverantwortung pro Service.
- Nicht-funktionale Eigenschaften: Latenz, Fehlertoleranz, Verfügbarkeit, Skalierung.
- Qualität des API-Vertrags: Verständlichkeit, Vollständigkeit,-versionierbare Schnittstellen.

(8) Praxisbeispiel/Übung. Skizziere für einen fiktiven Online-Shop die Boundaries:

- Services: User-Management, Produktkatalog, Bestellung, Zahlungsdienst.
- Definiere klare API-Schnittstellen pro Service (Contract-First-Ansatz) und eine mögliche Versionierungsstrategie (z. B. v1 vs. v2).
- Beschreibe, welche Governance-Richtlinien angewendet werden (Katalog, Contract-Tests, Deprecation-Policy) und welche Boundary-Details zwischen den Services gelten.