Microservices und Enterprise-Software

Wie passt das zusammen?

Holger Knoche

CAU Kiel, Arbeitsgruppe Software Engineering

9. März 2020 @ AK MSDO

C A U
Christian-Albrechts-Universität zu Kiel

Agenda

1. Microservices vs. Enterprise-Software?
2. Häufige Reibungspunkte
3. Weg zu einer Referenzarchitektur?

Eigenschaften µS-Beispiele

- Einfache, kurze Prozesse für den Endkunden
- Dynamische, offene Workloads
- Web- oder App-basiert
- Offenheit gegenüber Veränderungen
- Häufig „junge“ Anwendungen

Prioritäten

- Elastizität / Skalierbarkeit
- Verfügbarkeit
- Time to Market

Enterprise-Software

- Komplexe, teils lange Prozesse (z.B. Kreditantrag, Lebensversicherung)
- Stabile und bekannte Anzahl professioneller Anwender
- Stabilität der Arbeitsabläufe bevorzugt
- Oft „alte“ Anwendungen oder –bestandteile

Prioritäten

- Wartbarkeit / Evolvierbarkeit
- Verfügbarkeit

Totale Gegensätze?

Nein, denn:

- Agilität und TTM werden im Enterprise-Umfeld wichtiger
  - Unterschiedliche Evolutionsgeschwindigkeit
  - Abschied von klassischen Releaseprozessen
- Trend zum Customer Self-Service
- Microservices versprechen weitere Vorteile:
  - Gesteigerte Entwicklerproduktivität
  - Bessere Testbarkeit
  - Höhere Wartbarkeit

Totale Gegensätze?

Aber:

- Unabhängige Deploybarkeit ist absolute Voraussetzung
- Inklusive Spezifikation der notwendigen Umgebung

Häufige Reibungspunkte

- Wiederverwendung
- Integration
- Transaktionen
- Prozessteuerung / Workflow
- Wo schneide ich wie?

Wiederverwendung

Enterprise

Stark dafür

Microservices

Eher dagegen

Wiederverw.: Argumente

Enterprise:

- Zentralisierung von Änderungen, insb. Fehlerfixes
- Effizienzgewinn, keine Redundanz

Microservices:

- Schafft starke Bindungen und zementiert den Provider
- Führt zum unbemerkten Einsickern evtl. unerwünschter Änderungen
- Erschwert unabhängige Evolution

Wiederverw.: Meinung

Anhaltspunkt: Das Single Responsibility Principle (SRP): *„A module should be responsible to one, and only one, actor.“*

Konsequenz: Wiederverwendung nur…

- …für denselben Aktor (Stakeholder) oder
- …falls alle Aktoren desbezüglich indifferent sind und
- …die unabhängige Evolution nicht behindert wird

Integration

Enterprise

Integration möglichst früh

Microservices

Integration möglichst spät

Integration: Argumente

Enterprise

- Früh „höherwertige“ Services schaffen
- DB-Integration ist performant

Microservices

- DB-Integration widerspricht Microservices
- Gefahr von „chatty services“
- Gute Parallelisierbarkeit erst im UI

Integration: Meinung

Grundstrategie: Möglichst späte Integration mit flankierenden Performancemustern

- Replikation in lokale DB (Achtung: Ggf. Nutzungsbedingungen des Providers)
- CQRS, falls sinnvoll
- Batch-APIs (z.B. mit GraphQL)

Transaktionen

Enterprise

Eigentlich immer

Microservices

Nur lokal

Transaktionen: Argumente

Enterprise

- Bietet Sicherheit im Abbruchfall
- War schon immer so

Microservices

- Verteilte Transaktionen behindern Skalierung und Verfügbarkeit
- TX werden nicht von allen DBs unterstützt
- Verteilte TX machen immer Probleme
- Die Geschäftsprozesse bieten oft Alternativen

Transaktionen: Meinung

Grundstrategie: Vermeidung verteilter Transaktionen

- Prüfung der Geschäftsprozesse auf inhärente Sicherheitsmechanismen
- Vermeiden der „großen Abschlusstransaktion“
- Geeigneter Prozess- und Komponentenschnitt
- Ggf. Nutzung von Sagas

Workflow

Enterprise

Sehr beliebt

Microservices

Bloß keine Orchestrierung

Workflow: Argumente

Enterprise

- Prozessteuerung ist notwendig, insb. lange Prozesse
- Erlauben Monitoring des Geschäftsbetriebs

Microservices

- Orchestrierung führt zu stärkerer Kopplung
- Choreographie bevorzugt, da geringere Kopplung
- Workflow-Engines erfordern ggf. Co-Deployment

Workflow: Meinung

Grundstrategie: Workflow ist okay, wo sinnvoll, aber mit Bedacht:

- Innerhalb eines Teams ist Co-Deployment einigermaßen unschädlich
- Trotzdem Versionierung o.ä. erforderlich
- Teamübergreifend stabile Prozesse anstreben, die Subprozesse per Event anstoßen

Schnittmuster?

Microservices, SCS, Verticals,… – Was denn nun?

- Abgrenzung der Begriffe zueinander schwierig
- Technisch geprägt (Deployment)
- Organisatorisch nur schwach verankert
- Bezug zu DDD teils schwierig
- Kaum Anhalt für Verortung benötigter Konzepte (Workflow, Saga)

Relevante Dimensionen

Fachlich-organisatorisch

- Team
- Aktor / Stakeholder
- (Sub-)domäne
- Domäne / Organisation
- „Öffentlichkeit“

DDD

- Bounded Context
- Aggregate Root

Relevante Dimensionen (contd)

Technisch

- Deployment
- Deployable
- Service
- SCS

Zusammenspiel

- Keine teamübergreifenden technischen Konstrukte
- Services behandeln Aggregate Roots oder „kleine“ Bounded Contexts, SCS nur noch BCs
- Evolvierbare Kontrakte mit zunehmender Lebensdauer auf höheren Ebenen
- Domain Events spätestens ab SCS
- Für Härtefälle: Batch APIs
- Jedes Deployable realisiert in sich abgeschlossene Schritte (nur hier TX zulässig)

Zusammenspiel (contd)

- Jedes technische Artefakt ist einem Aktor zugeordnet oder stabil indifferent
- Orchestrierung innerhalb von Teams (Workflow, Saga), darüber nach Möglichkeit Explikation im Prozess
- Übergeordnete Prozesse möglichst stabil und über Events koordiniert bzw. per UI integriert

Fazit

Grundsätzlich gehen Microservice-Konzepte und Enterprise-Software zusammen.

Allerdings:

- Unabhängiges Deployment ist notwendig
- Kompromisse können und müssen ggf. anders ausgelegt werden