

Inhaltsverzeichnis

	Einleitung	15
	Über den Autor	20
I	Microservices	21
I.1	Was sind Microservices?	22
I.1.1	Klein und darauf spezialisiert, eine bestimmte Aufgabe richtig gut zu erledigen.	22
I.1.2	Eigenständigkeit	23
I.2	Die wichtigsten Vorteile	24
I.2.1	Verschiedenartige Technologien	24
I.2.2	Belastbarkeit	26
I.2.3	Skalierung	26
I.2.4	Komfortables Deployment	27
I.2.5	Betriebliche Abstimmung	28
I.2.6	Modularer Aufbau	28
I.2.7	Austauschbarkeit	29
I.3	Was ist mit serviceorientierten Architekturen?	29
I.4	Weitere Verfahren zur Aufspaltung	30
I.4.1	Programmbibliotheken	31
I.4.2	Module	31
I.5	Kein Patentrezept	33
I.6	Fazit	33
2	Der fortentwickelte Systemarchitekt	35
2.1	Unangebrachte Vergleiche	35
2.2	Das Zukunftsbild eines Systemarchitekten	37
2.3	Zoneneinteilung	39
2.4	Ein grundsätzlicher Ansatz	40
2.4.1	Strategische Ziele	41
2.4.2	Prinzipien	41
2.4.3	Praktiken	42

2.4.4	Prinzipien und Praktiken vereinigen	42
2.4.5	Ein Praxisbeispiel	43
2.5	Mindestvorgaben	44
2.5.1	Monitoring	44
2.5.2	Schnittstellen	45
2.5.3	Architektonische Sicherheit	45
2.6	Lenkung durch Code	46
2.6.1	Musterbeispiele	46
2.6.2	Maßgeschneiderte Servicevorlagen	46
2.7	Technische Schulden	48
2.8	Ausnahmebehandlung	49
2.9	Governance und Steuerung aus der Mitte	50
2.10	Aufbau eines Entwicklerteams	52
2.11	Fazit	52
3	Gestaltung von Services	55
3.1	Kurz vorgestellt: MusicCorp	55
3.2	Wodurch zeichnet sich ein guter Service aus?	56
3.2.1	Lose Kopplung	56
3.2.2	Hochgradige Geschlossenheit	56
3.3	Begrenzter Kontext	57
3.3.1	Geteilte und verborgene Modelle	58
3.3.2	Module und Services	59
3.3.3	Verfrühte Aufteilung	60
3.4	Funktionalitäten des Kontexts	61
3.5	Schildkröten bis ganz unten	61
3.6	Kommunikation unter geschäftlichen Aspekten	63
3.7	Der technische Rahmen	63
3.8	Fazit	65
4	Integration	67
4.1	Die Suche nach der optimalen Integrationsmethode	67
4.1.1	Zu Ausfällen führende Änderungen vermeiden	67
4.1.2	Technologieunabhängige APIs verwenden	67
4.1.3	Services für den Nutzer vereinfachen	68
4.1.4	Implementierungsdetails verbergen	68
4.2	Kundendatensätze	68
4.3	Gemeinsame Nutzung der Datenbank	69
4.4	Synchrone kontra asynchrone Kommunikation	70

4.5	Orchestrierung kontra Choreografie	72
4.6	Aufruf entfernter Prozeduren (RPC)	75
4.6.1	Kopplung von Technologien	76
4.6.2	Lokale Aufrufe sind keine entfernten Aufrufe.	76
4.6.3	Fragilität	77
4.6.4	Ist RPC ein Übel?	78
4.7	REST	79
4.7.1	REST und HTTP	80
4.7.2	HATEOAS	81
4.7.3	JSON, XML oder etwas anderes?	83
4.7.4	Vorsicht vor zu viel Komfort	84
4.7.5	Nachteile von REST über HTTP	85
4.8	Implementierung asynchroner ereignisgesteuerter Kollaboration	86
4.8.1	Verfügbare Technologien	86
4.8.2	Die Kompliziertheit asynchroner Architekturen	88
4.9	Services als Zustandsautomaten	90
4.10	Reactive Extensions	90
4.11	DRY und die Gefahren der Wiederverwendung von Code im Microservices-Umfeld	91
4.11.1	Client-Bibliotheken	92
4.12	Zugriff über Referenzen	93
4.13	Versionierung	95
4.13.1	Solange wie möglich hinauszögern	95
4.13.2	Zu Ausfällen führende Änderungen rechtzeitig erkennen	96
4.13.3	Verwendung semantischer Versionierung	97
4.13.4	Mehrere Endpunkte gleichzeitig betreiben	98
4.13.5	Mehrere Serviceversionen gleichzeitig betreiben	99
4.14	Benutzerschnittstellen	101
4.14.1	Zunehmend digital	101
4.14.2	Voraussetzungen	102
4.14.3	Aufbau der API	102
4.14.4	Bausteine der Benutzeroberfläche	104
4.14.5	Back-Ends für Front-Ends	106
4.14.6	Ein Hybridansatz	108
4.15	Integration der Software von Drittherstellern	108
4.15.1	Fehlende Entscheidungsmöglichkeiten	109

4.15.2	Anpassungen	109
4.15.3	Integrationswirrwarr	110
4.15.4	Auf sich selbst gestellt	110
4.15.5	Das Strangler-Pattern	113
4.16	Fazit	114
5	Die Aufspaltung des Monolithen	115
5.1	Seams	115
5.2	Aufspaltung von MusicCorp	116
5.3	Gründe zur Aufspaltung des Monolithen	117
5.3.1	Tempo der Änderungen	117
5.3.2	Teamstruktur	118
5.3.3	Sicherheitsaspekte	118
5.3.4	Technologie	118
5.4	Verwickelte Abhängigkeiten	118
5.5	Die Datenbank	119
5.6	Dem Problem zu Leibe rücken	119
5.7	Beispiel: Auflösen von Fremdschlüssel-Relationen	120
5.8	Beispiel: Statische Daten gemeinsam nutzen	122
5.9	Beispiel: Veränderliche Daten gemeinsam nutzen	123
5.10	Beispiel: Tabellen gemeinsam nutzen	125
5.11	Refactoring von Datenbanken	126
5.11.1	Die Aufspaltung umsetzen	126
5.12	Abgrenzung von Transaktionen	127
5.12.1	Versuchen Sie es später noch mal	129
5.12.2	Abbruch des gesamten Vorgangs	129
5.12.3	Verteilte Transaktionen	130
5.12.4	Was also tun?	131
5.13	Berichte	131
5.14	Datenbanken zur Berichterstellung	132
5.15	Datenabruf über Serviceaufrufe	134
5.16	Datenpumpen	135
5.16.1	Alternative Ziele	137
5.17	Ereignis-Datenpumpen	137
5.18	Backup-Datenpumpe	139
5.19	Benachrichtigung in Echtzeit	139
5.20	Änderungen verursachen Aufwand	140
5.21	Erkennen der eigentlichen Ursachen	141
5.22	Fazit	141

6	Deployment	143
6.1	Continuous Integration für Einsteiger	143
6.1.1	Machen Sie es auch richtig?	144
6.2	Continuous Integration und Microservices	145
6.3	Build Pipelines und Continuous Delivery	148
6.3.1	Die unvermeidlichen Ausnahmen	149
6.4	Plattformspezifische Artefakte	150
6.5	Betriebssystemspezifische Artefakte	151
6.6	Selbsterstellte Images	152
6.6.1	Images als Artefakte	154
6.6.2	Unveränderliche Server	155
6.7	Umgebungen	155
6.7.1	Servicekonfiguration	157
6.7.2	Zuordnung der Services zu den Hosts	158
6.7.3	Mehrere Services pro Host	158
6.7.4	Anwendungscontainer	161
6.7.5	Ein Service pro Host	162
6.7.6	Platform-as-a-Service (PaaS)	163
6.8	Automatisierung	164
6.8.1	Zwei Fallstudien zur Leistungsfähigkeit der Automatisierung	165
6.9	Physisch wird virtuell	166
6.9.1	Herkömmliche Virtualisierung	166
6.9.2	Vagrant	168
6.9.3	Linux-Container	168
6.9.4	Docker	170
6.10	Schnittstelle für das Deployment	171
6.10.1	Definition der Umgebung	173
6.11	Fazit	174
7	Testen	177
7.1	Testtypen	177
7.2	Testumfang	178
7.2.1	Unit-Tests	180
7.2.2	Servicetests	181
7.2.3	End-to-End-Tests	182
7.2.4	Nachteile	182
7.2.5	Wie viele Tests?	183

7.3	Implementierung von Servicetests	183
7.3.1	Mock-Objekte kontra Platzhalter	184
7.3.2	Ein intelligenterer Platzhalterservice	185
7.4	Knifflige End-to-End-Tests	185
7.5	Nachteile von End-to-End-Tests	187
7.5.1	Unzuverlässige und fragile Tests	187
7.5.2	Wer programmiert die Tests?	188
7.5.3	Testdauer	189
7.5.4	Das große Auftürmen	190
7.5.5	Die Metaversion	191
7.6	Abläufe testen, nicht Funktionalitäten	191
7.7	Abhilfe durch Consumer-Driven Tests	192
7.7.1	Pact	194
7.7.2	Konversationen	195
7.8	End-to-End-Tests: Pro und Kontra	196
7.9	Testen nach der Veröffentlichung	196
7.9.1	Deployment und Veröffentlichung trennen	197
7.9.2	Canary-Veröffentlichung	198
7.9.3	MTTR kontra MTBR	200
7.10	Funktionsübergreifende Tests	201
7.10.1	Geschwindigkeitstests	202
7.11	Fazit	203
8	Monitoring	205
8.1	Ein Service, ein Server	206
8.2	Ein Service, mehrere Server	207
8.3	Mehrere Services, mehrere Server	208
8.4	Protokolle, Protokolle und noch mehr Protokolle	208
8.5	Kennzahlen mehrerer Services	209
8.6	Servicekennzahlen	211
8.7	Monitoring von Pseudo-Ereignissen	212
8.7.1	Implementierung des semantischen Monitorings	213
8.8	Korrelations-IDs	213
8.9	Die Aufrufkette	216
8.10	Standardisierung	216
8.11	Zielgruppen	217
8.12	Wie geht es weiter?	218
8.13	Fazit	219

9	Sicherheit	221
9.1	Authentifizierung und Autorisierung	221
9.1.1	Gängige Single-Sign-On-Implementierungen	222
9.1.2	Single-Sign-On-Gateway	223
9.1.3	Fein unterteilte Authentifizierung	225
9.2	Authentifizierung und Autorisierung von Services	226
9.2.1	Im internen Netzwerk ist alles erlaubt	226
9.2.2	Authentifizierung über HTTP(S)	226
9.2.3	Verwendung von SAML oder OpenID Connect	227
9.2.4	Client-Zertifikate	228
9.2.5	HMAC über HTTP	229
9.2.6	API-Schlüssel	230
9.2.7	Das Stellvertreterproblem	231
9.3	Schutz ruhender Daten	233
9.3.1	Wohlbekannte Verfahren einsetzen	234
9.3.2	Die Bedeutung der Schlüssel	235
9.3.3	Was soll verschlüsselt werden?	235
9.3.4	Entschlüsselung bei Bedarf	236
9.3.5	Backups verschlüsseln	236
9.4	Gestaffelte Sicherheitsstrategie	236
9.4.1	Firewalls	236
9.4.2	Protokollierung	236
9.4.3	Intrusion-Detection-Systeme	237
9.4.4	Unterteilung des Netzwerks	237
9.4.5	Betriebssystem	238
9.5	Ein ausgearbeitetes Beispiel	239
9.6	Datensparsamkeit	241
9.7	Der Faktor Mensch	242
9.8	Eine Goldene Regel	242
9.9	Integrierte Sicherheit	243
9.10	Externe Prüfung	243
9.11	Fazit	244
10	Conways Gesetz und Systemdesign	245
10.1	Beweise	245
10.1.1	Lose und eng gekoppelte Organisationen	246
10.1.2	Windows Vista	246
10.2	Netflix und Amazon	246
10.3	Was kann man damit anfangen?	247

10.4	Anpassung an Kommunikationswege	247
10.5	Verantwortlichkeit für Services	249
10.6	Gemeinschaftliche Verantwortlichkeit für Services	249
10.6.1	Schwierige Aufspaltung	249
10.6.2	Feature-Teams	250
10.6.3	Engpässe bei der Auslieferung	250
10.7	Interner Open-Source-Code	251
10.7.1	Aufgaben der Koordinatoren	252
10.7.2	Ausgereifte Services	253
10.7.3	Werkzeugsammlungen	253
10.8	Begrenzte Kontexte und Teamstrukturen	253
10.9	Verwaiste Services?	254
10.10	Fallstudie: RealEstate.com.au	254
10.11	Conways Gesetz auf den Kopf gestellt	256
10.12	Menschen	257
10.13	Fazit	258
11	Microservices skalieren	259
11.1	Ausfälle gibt es immer.	259
11.2	Wie viel ist zu viel?	260
11.3	Schrittweiser Abbau der Funktionalität	261
11.4	Architektonische Sicherheitsmaßnahmen.	262
11.5	Die antifragile Organisation	265
11.5.1	Timeouts	266
11.5.2	Circuit Breaker	266
11.5.3	Das Bulkhead-Pattern	269
11.5.4	Isolierung	270
11.6	Idempotenz	270
11.7	Skalierung	272
11.7.1	Mehr Leistung	272
11.7.2	Arbeitslast aufteilen	273
11.7.3	Risikoverteilung	273
11.7.4	Lastverteilung	274
11.7.5	Worker-Systeme	276
11.7.6	Neuanfang	277
11.8	Datenbanken skalieren	278
11.8.1	Verfügbarkeit des Services kontra Lebensdauer der Daten	278
11.8.2	Skalierung bei Lesevorgängen	279

II.8.3	Skalierung bei Schreibvorgängen	280
II.8.4	Gemeinsam genutzte Datenbankinfrastruktur	281
II.8.5	CQRS.	281
II.9	Caching.	282
II.9.1	Clientseitiges Caching, Proxy und serverseitiges Caching	283
II.9.2	Caching und HTTP	284
II.9.3	Caching bei Schreibvorgängen	285
II.9.4	Caching zur Erhöhung der Belastbarkeit	286
II.9.5	Den Ursprung verbergen.	286
II.9.6	Möglichst einfach	287
II.9.7	Cache Poisoning: Ein warnendes Beispiel	288
II.10	Automatische Skalierung	289
II.11	Das CAP-Theorem.	290
II.11.1	Aufgabe der Konsistenz	292
II.11.2	Aufgabe der Verfügbarkeit.	292
II.11.3	Aufgabe der Partitionstoleranz?	294
II.11.4	AP oder CP?	294
II.11.5	Keine Frage eines Entweder-Oders	294
II.11.6	Abbildung der Wirklichkeit	295
II.12	Serviceerkennung	296
II.12.1	DNS	296
II.13	Dynamische Registrierung von Services	298
II.13.1	Zookeeper	298
II.13.2	Consul	300
II.13.3	Eureka	301
II.13.4	Eigene Serviceregistrierung.	301
II.13.5	Menschliches Interesse	302
II.14	Services dokumentieren	302
II.14.1	Swagger	302
II.14.2	HAL und der HAL-Browser	303
II.15	Ein sich selbst beschreibendes System	304
II.16	Fazit	305
12	Auf den Punkt gebracht	307
12.1	Prinzipien.	307
12.1.1	Geschäftsvorgänge modellieren	308
12.1.2	Automatisierung kultivieren	308
12.1.3	Implementierungsdetails verbergen.	309

12.1.4	Dezentralisierung	309
12.1.5	Unabhängiges Deployment	310
12.1.6	Ausfälle eingrenzen	310
12.1.7	Umfassendes Monitoring	311
12.2	Wann sollte man auf Microservices verzichten?	311
12.3	Schlusswort	312
	Stichwortverzeichnis	313