

HANSER



Leseprobe

zu

Cloud-native Computing

von Nane Kratzke

Print-ISBN: 978-3-446-47914-2

E-Book-ISBN: 978-3-446-47925-8

epub-ISBN: 978-3-446-48029-2

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446479142>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	XIII
----------------------	-------------

1 Einleitung	1
1.1 An wen sich dieses Buch richtet	2
1.2 Was dieses Buch behandelt	3
1.3 Sprachliche Konventionen	5
1.4 Notationskonventionen	6
1.5 Ergänzende Materialien	7

Teil I: Grundlagen	9
---------------------------------	----------

2 Cloud Computing	11
2.1 Service-Modelle	12
2.1.1 Infrastructure as a Service (IaaS)	15
2.1.2 Platform as a Service (PaaS)	15
2.1.3 Software as a Service (SaaS)	16
2.2 Cloud-Ökonomie	19
2.2.1 Eignung von unterschiedlichen Arten von Workloads	19
2.2.2 Effekt von Zuteilungsdauer und Ressourcengröße	22
2.3 Entwicklung der letzten Jahre	24
3 DevOps	27
3.1 Prinzipien des Flow	29
3.1.1 Prinzip 1: Arbeit sichtbar machen	29
3.1.2 Prinzip 2: Work in Progress beschränken	30
3.1.3 Prinzip 3: Flaschenhalse minimieren	31
3.2 Prinzipien des Feedbacks	32
3.2.1 Prinzip 4: Probleme früh erkennen	32
3.2.2 Prinzip 5: Probleme sofort lösen	32
3.2.3 Prinzip 6: Probleme professionell verantworten	33
3.3 DevOps-geeignete Architekturen	33
3.3.1 Randbedingungen für die Entwicklung	34

3.3.2	Nutzung von Orchestrierungsplattformen.....	34
3.3.3	Randbedingungen im Betrieb	35
4	Cloud-native	37
4.1	Definitionen in Industrie und Forschung.....	39
4.2	Die Cloud-native-Definition dieses Buchs.....	40
4.3	Zusammenfassung und Ausblick auf Teil II bis IV	41
	Teil II: Everything as Code.....	45
5	Einleitung zu Teil II.....	47
6	Deployment-Pipelines	49
6.1	Deployment-Pipelines as Code.....	50
6.1.1	Phasen-Pipelines.....	51
6.1.2	Gerichtete Pipelines	52
6.1.3	Hierarchische Pipelines.....	53
6.1.4	Steuerung von Pipelines	54
6.2	DevOps-geeignete Branching-Strategien.....	56
6.2.1	Git-Flow	57
6.2.2	GitHub-Flow	58
6.2.3	Trunk-basierte Entwicklung	59
6.3	Zusammenfassung	60
7	Infrastructure as Code.....	63
7.1	Virtualisierung	65
7.1.1	Virtualisierung von Hardware-Infrastruktur.....	65
7.1.2	Virtualisierung von Software-Infrastruktur.....	66
7.2	Provisionierung.....	68
7.2.1	Immutable Infrastructure	68
7.2.2	IaC-Ansätze	69
7.2.3	Provisionierung von lokalen Umgebungen	72
7.2.4	Provisionierung von Multi-Host-Umgebungen	74
7.3	Zusammenfassung	77
8	Standardisierung von Deployment Units (Container)	79
8.1	Hintergrund (PaaS).....	79
8.2	Betriebssystem-Virtualisierung.....	82
8.3	Container Runtime Environments.....	83
8.3.1	Kernel-Namespaces	84
8.3.2	Process Capabilities	85
8.3.3	Control Groups	86

8.3.4	Union Filesystem	86
8.3.5	High-Level- und Low-Level-Container-Laufzeitumgebungen	87
8.4	Bau und Bereitstellung von Container-Images	88
8.5	Faktoren gut betreibbarer Container	90
8.5.1	Codebase	91
8.5.2	Abhängigkeiten und Konfigurationen	91
8.5.3	Unterstützende Services und Port Binding	92
8.5.4	Build-, Release- und Run-Phase	93
8.5.5	Horizontale Skalierung über Prozesse	94
8.5.6	Umgebungen, Logs und Betrieb	95
8.6	Zusammenfassung	96
9	Container-Plattformen	99
9.1	Scheduling	100
9.1.1	Heterogenität von Workloads	101
9.1.2	Scheduling-Algorithmen	102
9.1.2.1	Einfache Scheduling-Algorithmen	102
9.1.2.2	Multidimensionale Scheduling-Algorithmen	103
9.1.2.3	Kapazitätsbasierte Scheduling-Algorithmen	103
9.1.3	Scheduling-Architekturen	104
9.1.3.1	Monolithischer Scheduler	105
9.1.3.2	2-Level-Scheduler	105
9.1.3.3	Shared-State Scheduler	106
9.2	Orchestrierung	107
9.2.1	Definition von Betriebszuständen	107
9.2.2	Regelkreis: Desired versus Current State	108
9.3	Inside Kubernetes	109
9.3.1	Kubernetes-Architektur	110
9.3.2	Verwaltete Ressourcen und Basis-Blueprint	112
9.3.3	Schedulbare Workloads	114
9.3.3.1	Deployments	114
9.3.3.2	(Cron-)Jobs	116
9.3.3.3	Daemon-Sets	117
9.3.3.4	Stateful-Sets	118
9.3.4	Scheduling Constraints	121
9.3.4.1	Angabe des Ressourcenbedarfs mittels Requests und Limits	121
9.3.4.2	Knoten-Selektoren	122
9.3.4.3	Knotenaffinitäten	123
9.3.4.4	Pod-(Anti-)Affinitäten	124
9.3.5	Automatische Skalierung von Workloads	125
9.3.6	Exponieren von Workloads als interne und externe Services	126
9.3.7	Health Checking	129
9.3.8	Persistenz	132

9.3.9	Isolation von Workloads	133
9.3.9.1	Namespaces und Role-based Access Model (Multi-Tenancy)	133
9.3.9.2	Quotas und Limit Ranges	134
9.3.9.3	Network Policys	135
9.4	Zusammenfassung	137
10	Function as a Service	141
10.1	FaaS-Plattformen	143
10.1.1	Das FaaS-Programmiermodell	145
10.1.2	Zu berücksichtigende Randbedingungen	146
10.1.3	Veranschaulichung des FaaS-Programmiermodells	147
10.2	Plattformagnostische FaaS-Frameworks	149
10.3	Ereignisbasierte Autoskalierung	152
10.4	Zusammenfassung	155
 Teil III: Cloud-native Architekturen		157
11	Einleitung zu Teil III	159
12	Microservice und Serverless-Architekturen	161
12.1	Eigenschaften von Microservices	162
12.2	Integrationsmuster für Microservices	166
12.2.1	Datenbankbasierte Integration	167
12.2.2	(g)RPC-basierte Interprozesskommunikation	167
12.2.3	Representational State Transfer (REST)	170
12.2.4	Ereignisbasierte Integration (asynchron)	173
12.2.5	API-Versioning	175
12.3	Architekturelle Sicherheit	178
12.3.1	Circuit-Breaker	178
12.3.2	Bulkhead	179
12.3.3	Idempotente API-Operationen	180
12.4	Skalierung von Microservices	180
12.4.1	Load Balancing	181
12.4.2	Messaging	181
12.4.3	Skalierung zustandsbehafteter Komponenten	183
12.4.3.1	Scaling for Reads	184
12.4.3.2	Scaling for Writes (Sharding)	184
12.4.3.3	Command Query Responsibility Segregation (CQRS)	185
12.4.4	Caching	186
12.5	Prinzipien zur Entwicklung von Microservices	187
12.5.1	Prinzip 1: Bilde Modelle um Geschäftskonzepte	187
12.5.2	Prinzip 2: Erschaffe eine Kultur der Automatisierung	187

12.5.3	Prinzip 3: Blende interne Implementierungsdetails aus	188
12.5.4	Prinzip 4: Dezentralisiere	188
12.5.5	Prinzip 5: Definiere unabhängig aktualisierbare Einheiten	188
12.5.6	Prinzip 6: Isoliere Fehler	189
12.5.7	Prinzip 7: Baue gut beobachtbare Services	189
12.6	Serverless-Architekturen	190
12.6.1	Architekturelle Konsequenzen von Serverless-Limitierungen	191
12.6.2	Das API-Gateway-Pattern	193
12.6.3	Abgrenzung zu Microservices	195
12.7	Zusammenfassung	196
13	Beobachtbare Architekturen	199
13.1	Konsolidierung von Telemetriedaten	200
13.2	Instrumentierung von Systemen	202
13.2.1	Logging	202
13.2.2	Monitoring	204
13.2.2.1	Metrikarten	206
13.2.2.2	Empfehlungen für die Metrikinstrumentierung	207
13.2.3	Tracing	207
13.2.3.1	Empfehlungen für die Instrumentierung	209
13.2.3.2	Tracing-Instrumentierung und Erzeugung von Spans	212
13.2.3.3	Serverseitiges Tracing und Extraktion von Span-Kontexten	213
13.2.3.4	Clientseitiges Tracing und Weiterreichen von Span-Kontexten	214
13.3	Automatisierte Instrumentierung	215
13.3.1	Eigenschaften von Service-Meshs	216
13.3.2	Traffic-Management	218
13.3.3	Resilienz	221
13.3.4	Sicherheit	223
13.3.5	Management und Analyse von Verkehrstopologien	226
13.4	Zusammenfassung	227
14	Domain-driven Design	229
14.1	Fachlichkeit	230
14.2	Strategisches Design	232
14.2.1	Subdomänen	233
14.2.1.1	Kerndomäne (Core Subdomain)	233
14.2.1.2	Unterstützende Subdomäne (Supporting Subdomain)	234
14.2.1.3	Generische Subdomänen (Generic Subdomain)	234
14.2.1.4	Anmerkungen am Beispiel einer Fallstudie	234
14.2.2	Ubiquitous Language	236
14.2.2.1	Eine gemeinsame Sprache als Schlüssel zu einem gemeinsamen Verständnis	237
14.2.2.2	Mehrdeutige und synonyme Begriffe	238

14.2.3	Bounded Contexts	239
14.2.4	Context Mapping	241
14.2.4.1	Partnerschaftliche Kooperationsmuster (Partners und Shared-Kernel)	241
14.2.4.2	Customer-Supplier-Kooperation	243
14.2.4.3	Separate Ways	244
14.2.4.4	Context Maps als Landkarte von Machtverhältnissen	245
14.3	Taktisches Design	246
14.3.1	Oft genutzte Pattern für Geschäftslogik	246
14.3.1.1	Das ETL-Pattern (primär Supporting Subdomains)	246
14.3.1.2	Das Active Record-Pattern (primär Supporting Subdomains)	247
14.3.1.3	Das Domain Model-Pattern (primär Core Subdomains)	248
14.3.1.4	Das Event-Sourcing-Pattern (primär Core Subdomains)	250
14.3.2	Oft genutzte Pattern für die Architektur	251
14.3.2.1	Die Ebenen-Architektur	252
14.3.2.2	Das Ports & Adapter-Pattern	253
14.3.2.3	Das CORS-Pattern	253
14.4	Zusammenfassung	256

Teil IV: Sichere Cloud-native Anwendungen 259

15 Einleitung zu Teil IV 261

16 Härtung Cloud-nativer Anwendungen 263

16.1	Härtung (virtueller) Infrastrukturen	267
16.1.1	Tool-gestütztes System Hardening	268
16.1.2	Kontinuierliche Aktualisierung von virtuellen Maschinen	270
16.1.3	Sichere Authentifizierung mittels SSH	271
16.1.4	Kontinuierliche Überwachung virtueller Maschinen	273
16.1.4.1	Verhaltensbasierte Intrusion Detection mittels Auditing	273
16.1.4.2	Signaturbasierte Intrusion Detection	274
16.1.4.3	Log Forwarding	276
16.1.5	Einsatz von Sicherheitsgruppen und Firewalls	277
16.2	Härtung containerisierter Workloads	279
16.2.1	Absicherung von Public Endpoints mittels Ingresses	280
16.2.2	Namespace-basierte Netzwerkisolation	286
16.2.3	Pod Hardening	289
16.2.4	Erhöhung der Container Runtime Isolation	299
16.2.5	Volume Hardening	300
16.2.6	Workload Policing	303
16.2.7	Intrusion Detection	307
16.2.8	Sicherung der Supply Chain	310

16.2.8.1	Statische Software Composition Analysis (SCA)	311
16.2.8.2	Static Application Security Testing (SAST)	314
16.2.8.3	Kontinuierliches Schwachstellen-Scanning von Container- Plattformen	316
16.3	Zusammenfassung	319
17	Regulatorische Anforderungen	321
17.1	Cloud Compliance und Zertifizierungen	322
17.1.1	ISO 9001	324
17.1.2	BSI-IT-Grundschutz und BSI-Standards	325
17.1.3	BSI-C5-Zertifizierung	325
17.1.4	ISO/IEC 27001 und 27017/27018	326
17.1.5	CSA STAR	327
17.1.6	CISPE Code of Conduct	328
17.1.7	EU Cloud Code of Conduct	329
17.1.8	SOC 1-3 (Service Organization Control)	330
17.1.9	FedRAMP	331
17.1.10	HIPAA	331
17.1.11	PCI DSS	332
17.1.12	Zusammenfassung	333
17.2	Aus der DSGVO sich ergebende Anforderungen	335
17.2.1	Personenbezogene Daten	335
17.2.2	Grundsätze der Verarbeitung personenbezogener Daten	336
17.2.3	Auftragsverarbeitung	338
17.2.4	Datenschutz-Folgenabschätzungen	340
17.2.5	Internationale Datentransfers in Drittländer	341
17.3	Europäischer Datenschutz und Drittländer	343
17.3.1	Probleme am Beispiel des CLOUD Act	344
17.3.2	Lösungen trotz SCHREMS I + II	345
17.4	Zusammenfassung	346
18	Schlussbemerkungen	349
	Literaturverzeichnis	359
	Stichwortverzeichnis	365

Vorwort

Dieses Buch basiert auf zwei Vorlesungen, „*Cloud-native Programmierung*“ und „*Cloud-native Architekturen*“, die ich an der Technischen Hochschule Lübeck gebe. Während der Recherchen für diese beiden Hochschulmodule war ich natürlich auch auf der Suche nach geeigneter Literatur. Das Resultat war ein Literaturumfang, der – auf einem Schreibtisch gestapelt – leider mehr als einen halben Meter Höhe eingenommen hätte.

Meine Recherche mag unzureichend oder meine Anforderungen zu spezifisch gewesen seien, aber ich fand leider nicht die eine oder zwei geeigneten Quellen, die man jemandem als Lehrbuch zum Thema Cloud-native Computing hätte empfehlen und an die Hand geben können; nur eben diesen *Bücherstapel*. Diese Literaturliste hätte mir aber vermutlich diverse kritische Blicke meiner Studentinnen und Studenten eingebracht. Auch wenn ich grundsätzlich kein Freund des Prinzips „*Setze dich zwischen zweier Bücher Mitte und schreib das Dritte*“ bin, war genau dies in diesem Fall der Anstoß zum Schreiben eines ersten Skripts, aus dem letztlich dieses Buch für die beiden oben genannten Lehrveranstaltungen entstanden ist.

Dieses Buch hat somit auch einen gewissen Handbuch-Charakter, auch wenn es kein Handbuch im klassischen Sinne ist. Es kann dennoch bis zu einem gewissen Grad als Nachschlagewerk genutzt werden, da es eine Vielzahl an hervorragender – aber eben leider isolierter – Literatur zum Thema Cloud-native Computing zusammenfasst.

Ich möchte mich an dieser Stelle u. a. bei Dr. Josef Adersberger von der QAware GmbH bedanken, der eine ähnliche Publikationsidee hatte, dann aber letztlich keine Zeit fand, sein Projekt auch umzusetzen, und der mich daraufhin mit dem Hanser Verlag in Kontakt brachte, um es an seiner Stelle zu versuchen. Zu danken ist auch seinen Mitarbeitern. Deren auf GitHub bereitgestellte Vorlesungsunterlagen „*Cloud Computing*“ (Adersberger u. a. 2018) waren insbesondere für den Teil II dieses Buchs wertvolle Inspiration und Gliederungshilfe. Dank gebührt daher auch dem Hanser Verlag und hier vor allem Sylvia Hasselbach, die sich auf diese Kontaktvermittlung und das damit einhergehende Wagnis denn auch eingelassen hat und insbesondere in der Produktionsphase viel Unterstützung geleistet hat.

Besonderer Dank gebührt auch meinen Studierenden, die die undankbare Betatester-Rolle für die praktischen Anteile (Labs) dieses Buchs übernommen haben und mir während der – aufgrund Corona leider nur online stattfindenden – Vorlesungen und Praktika dennoch mit vielen wertvollen Rückmeldungen geholfen haben, die Struktur und den Inhalt des Manuskripts für die anvisierte Zielgruppe zu optimieren. Dabei sind insbesondere Jannik Kühnemundt, Felix Lohse, Lucian Schultz und Jana Schwieger zu nennen, die mehrere vertiefende Labs entwickelt und für Folgejahrgänge zur Verfügung gestellt haben.

Nane Kratzke

2

Cloud Computing

„It's the economy, stupid!“

Bill Clinton, 42. Präsident der USA

Gemäß der sogenannten NIST-Definition versteht man unter Cloud Computing einen „all-gegenwärtigen, bequemen, bedarfsgerechten Netzwerkzugriff auf einen gemeinsamen Pool konfigurierbarer Rechenressourcen, die schnell und mit minimalem Verwaltungsaufwand oder Interaktion mit Service-Providern bereitgestellt, aber auch wieder freigegeben werden können“ (Mell und Grance 2011).

Cloud Computing ordnet sich damit im Spektrum verteilter Systeme im Bereich des Service Computings und weniger im Bereich des High Performance bzw. Super-Computings ein, auch wenn die Einflussfaktoren mittlerweile mannigfaltig und keinesfalls mehr als trennscharf zu bezeichnen sind (siehe Bild 2.1). Insbesondere im NoSQL- sowie Machine Learning-/Big-Data-Bereich gehen Super-Computing und Service Computing zunehmend mehr ineinander über.

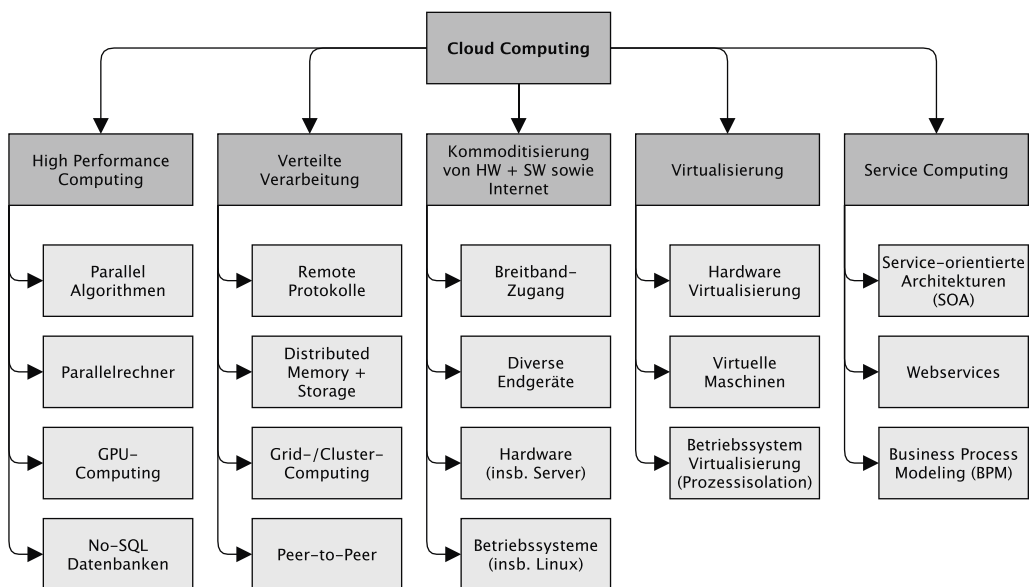


Bild 2.1 Einflussfaktoren auf das Cloud Computing

Während Super-Computing eine wichtige Rolle im Bereich der computergestützten Wissenschaften (Computational Science) spielt und für eine Vielzahl rechenintensiver wissenschaftlicher Aufgaben in verschiedensten Bereichen eingesetzt wird (z. B. Quantenmechanik, Wettervorhersage, Klimaforschung, physikalische Simulationen usw.), verstehen wir unter Service Computing eher einen interdisziplinären Ansatz, der sich mit der Frage beschäftigt, wie Informationstechnologien die geschäftsrelevante Erzeugung von Produkten und Dienstleistungen substantziell unterstützen können. Dabei finden im Service Computing u. a. Webservices, Service-orientierte Architekturen (SOA), Geschäftsprozessmodellierung, Transformations- und Integrationstechnologien – aber eben auch vermehrt „Enabling Technologies“ wie Cloud Computing – Anwendung, die durchaus substantziellen Einfluss auf Architekturen und Systeme haben. So hat sich beispielsweise SOA aufgrund des Cloud Computing-Einflusses in den letzten Jahren mehr und mehr zu einem Microservice-basierten Architekturansatz fortentwickelt. Warum das so ist, werden wir unter anderem in Abschnitt 2.3 und Abschnitt 2.4 sehen.

■ 2.1 Service-Modelle

Im Allgemeinen werden, wie in Bild 2.2 gezeigt, im Cloud Computing fünf wesentliche Service-Merkmale, vier Deployment-Modelle und drei Service-Modelle unterschieden (Mell und Grance 2011). Wir werden im weiteren Verlauf sehen, dass diese Darstellung an der ein oder anderen Stelle verfeinert werden kann (siehe beispielsweise Abschnitt 8.1 und Bild 8.3). Dennoch ist das zugrunde liegende NIST-Modell des Cloud Computings (Mell und Grance 2011) so prägend, dass es Sinn macht, sich an diesem Modell, seinen Merkmalen, Bereitstellungsformen und Service-Modellen zu orientieren.

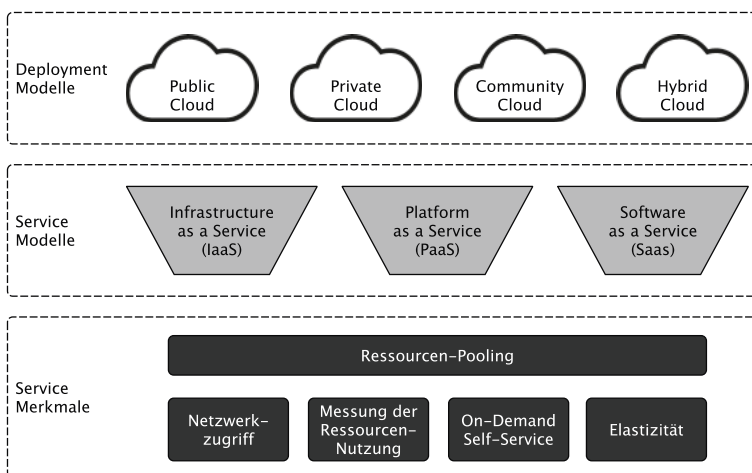


Bild 2.2 NIST-Modell des Cloud Computings

Zu den fünf wesentlichen Merkmalen des Cloud Computings sind die folgenden zu zählen:

1. **On-Demand Self-Service:** Ein Verbraucher kann Ressourcen, wie z. B. Serverzeit und Netzwerkspeicher, nach Bedarf automatisch anfordern, ohne dass hierfür eine manuelle Tätigkeit aufseiten des Cloud-Service-Providers erforderlich ist.
2. **Netzwerkzugriff:** Die Ressourcen werden über öffentliche Netzwerke bereitgestellt und der Zugriff auf diese Ressourcen erfolgt über standardisierte und weitverbreitete Internetprotokolle, die die Nutzung von Cloud-Ressourcen durch heterogene Client-Plattformen ermöglichen.
3. **Elastizität:** Ressourcen können schnell und bedarfsgerecht bereitgestellt, aber auch wieder freigegeben werden. Für den Verbraucher erscheinen die für die Bereitstellung verfügbaren Ressourcen virtuell unbegrenzt und können in beliebiger Menge und zu jeder Zeit angefordert werden. Dies fördert horizontale Skalierungsformen.
4. **Messung der Ressourcennutzung:** Cloud-Systeme steuern und optimieren automatisch ihre Ressourcennutzung, indem sie den Ressourcenverbrauch auf einer geeigneten Abstraktionsebene messen (z. B. Speicherverbrauch, Processing-Cycles, Bandbreite, aktive Benutzerkonten usw.). Die Überwachung und Messung der Ressourcennutzung schafft sowohl für den Service-Provider als auch für den Nutzer von Cloud Services Transparenz.
5. **Ressourcen-Pooling:** Die Computing-Ressourcen des Providers werden gepoolt, um mehrere Kunden mit einem Multi-Tenant-Modell zu bedienen. Dabei werden physische und virtuelle Ressourcen dynamisch den Nutzern zugewiesen und bei Bedarf auch reallokiert. Der Kunde hat im Allgemeinen keine detaillierte Kontrolle oder Kenntnis über den genauen Standort der bereitgestellten Ressourcen, kann aber den Standort auf einer höheren Abstraktionsebene (z. B. Land, Region oder Rechenzentrum) angeben.

Cloud Services werden zumeist in Private- bzw. Public Cloud-Formen unterschieden. Die ebenfalls existierenden Hybrid- und Community-Formen sind oft nicht so präsent in der öffentlichen Diskussion, vermutlich weil sie im Service Computing kaum ihre Stärken ausspielen können.

- Unter einer **Public Cloud** versteht man eine Cloud-Infrastruktur für die offene Nutzung durch die Allgemeinheit. Sie kann im Besitz einer geschäftlichen, akademischen oder staatlichen Organisation oder einer Kombination davon sein und von dieser verwaltet und betrieben werden. Sie befindet sich auf den Liegenschaften des Cloud-Anbieters (d. h. Off-Premise für die Cloud-Nutzer).
- Unter einer **Private Cloud** versteht man hingegen eine Cloud-Infrastruktur, die für die exklusive Nutzung durch eine einzelne Organisation mit mehreren Verbrauchern (z. B. Geschäftseinheiten) betrieben wird. Sie kann sich im Besitz der Organisation, eines Dritten oder einer Kombination aus beiden befinden. Dabei ist es unerheblich, ob die Infrastruktur sich auf den Liegenschaften der Organisation (d. h. On-Premise für die Cloud-Nutzer) oder nicht befindet.
- Unter der weniger bekannten Form der **Community Cloud** wird eine Cloud-Infrastruktur verstanden, die für die exklusive Nutzung durch eine bestimmte Gemeinschaft von Verbrauchern aus Organisationen betrieben wird. Diese Gemeinschaft hat meist gemeinsame Anliegen (z. B. Mission, Sicherheitsanforderungen, Richtlinien und Compliance-Überlegungen). Sie kann im Besitz einer oder mehrerer Organisationen in der Community, einer dritten Partei oder einer Kombination von ihnen sein und von diesen verwaltet und

betrieben werden. Dabei ist es unabhängig, ob die Community Cloud ausschließlich auf den Liegenschaften der Gemeinschaft betrieben wird. Community Clouds können also sowohl On-Premise als auch Off-Premise betrieben werden.

- Schließlich wird als **Hybrid Cloud** eine Cloud-Infrastruktur verstanden, die eine Komposition aus zwei oder mehreren oben genannter Cloud-Infrastruktur-Formen (private, public, community) bildet. Diese bleiben eigenständige Einheiten, werden aber durch standardisierte oder proprietäre Technologie miteinander verbunden, die die Portabilität von Daten und Anwendungen ermöglicht (z. B. Cloud Bursting für den Lastausgleich zwischen Cloud-Infrastrukturen).

Mittels Cloud-Computing lassen sich Teile der IT-basierten Wertschöpfung an externe Dienstleister (Cloud-Provider) auslagern. Der Auslagerungsumfang wird dabei häufig in die Kategorien Infrastructure as a Service (IaaS, siehe Abschnitt 2.1.1), Platform as a Service (PaaS, siehe Abschnitt 2.2) und Software as a Service (SaaS, siehe Abschnitt 2.2.1.1) eingeteilt. Von IaaS über PaaS zu SaaS wird dabei der ausgelagerte Anteil immer größer, wie Bild 2.3 zeigt. Mit dem Umfang der Auslagerung wird allerdings auch die potenzielle Abhängigkeit (Vendor Lock-in) eines Kunden zu einem Cloud-Provider größer. Unter einem Lock-in-Effekt versteht man generell eine enge Kundenbindung an Produkte/Dienstleistungen eines Anbieters in Form einer technisch-funktionalen Kundenbindung, die es dem Kunden wegen entstehender Wechselkosten und sonstiger Wechselbarrieren erschwert, ein Produkt oder einen Service eines Anbieters mit dem Produkt oder Service eines anderen Anbieters auszutauschen. Im Cloud Computing entsteht dieser Effekt meist durch nichtstandardisierte Cloud-Service APIs der einzelnen Provider. Je höher man in den Schichten kommt, desto spezifischer und damit weniger austauschbar werden die bereitgestellten Cloud-Services, und desto höher ist die Lock-in-Gefahr.

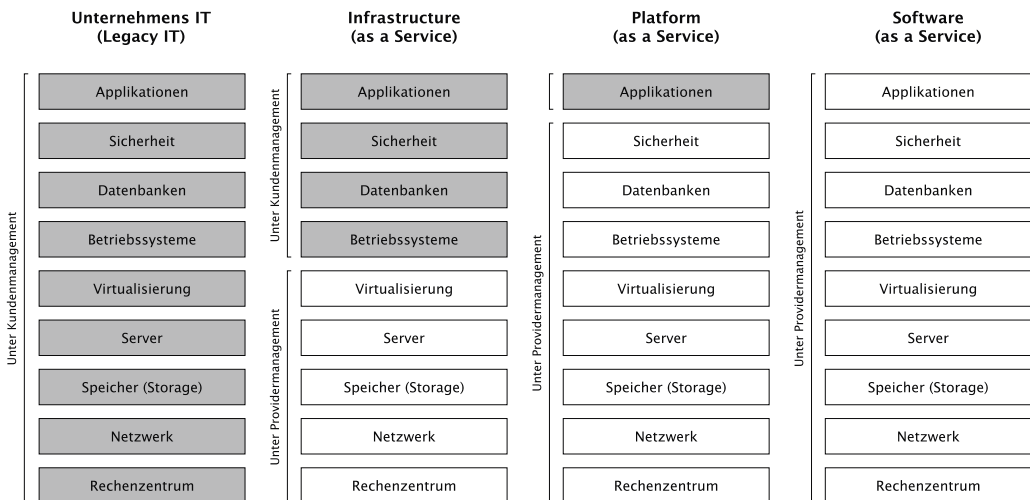


Bild 2.3 Auslagerung der Wertschöpfung bei IaaS, PaaS und SaaS

2.1.1 Infrastructure as a Service (IaaS)

Beim IaaS-Modell bietet ein Provider physische und virtuelle Hardware wie Server, Speicher und Netzwerkinfrastruktur an, die über eine Self-Service-Schnittstelle schnell bereitgestellt und außer Betrieb genommen werden kann. Dies ermöglicht es z. B., im Rahmen von periodischen Workloads mit wiederkehrenden Lastspitzen IT-Ressourcen flexibel und vor allem lastgetrieben bereitzustellen.

Die Fähigkeit, die dem Kunden zur Verfügung gestellt wird, besteht also in der schnellen und elastischen Bereitstellung von Verarbeitungs-, Speicher-, Netzwerk- und anderen grundlegenden Rechenressourcen, auf denen der Kunde beliebige Software, einschließlich Betriebssystemen und Anwendungen, einsetzen und ausführen kann.

Der Kunde verwaltet oder kontrolliert die zugrunde liegende Cloud-Infrastruktur zwar nicht, hat aber die Kontrolle über Betriebssysteme, Speicher und bereitgestellte Anwendungen sowie möglicherweise eine begrenzte Kontrolle über ausgewählte Netzwerkkomponenten (z. B. Host-Firewalls).

In Anlehnung an (Fehling u. a. 2014) bezeichnen wir das zugehörige Service-Offering als **elastische Infrastruktur** zum Zwecke der Bereitstellung von virtuellen Servern, persistenten Speicher und Netzwerkkonnektivität. Eine elastische Infrastruktur bietet zumeist vorkonfigurierte virtuelle Server-Images, persistenten Speicher und Netzwerkkonnektivität, die von Kunden über eine Self-Service-Schnittstelle angefordert werden können. Ferner werden Last- und Nutzungsdaten vom Provider bereitgestellt, um über die Ressourcenauslastung zu informieren, die für eine nachvollziehbare Abrechnung und die Automatisierung von Verwaltungsaufgaben erforderlich ist.

2.1.2 Platform as a Service (PaaS)

Beim PaaS-Modell stellen Provider IT-Ressourcen in Form einer Applikations-Hosting-Umgebung für Kunden bereit. Ein Cloud-Provider bietet hierfür verwaltete Betriebssysteme und Middleware an. Auch viele Betriebsvorgänge werden vom Anbieter übernommen, wie z. B. die elastische Skalierung und Ausfallsicherheit gehosteter Anwendungen.

Die dem Kunden zur Verfügung gestellte Fähigkeit besteht somit darin, in einer Cloud-Infrastruktur vom Kunden erstellte oder erworbene Anwendungen bereitzustellen, die mit vom Anbieter unterstützten Programmiersprachen, Bibliotheken, Diensten und Tools erstellt wurden. Der Kunde verwaltet oder kontrolliert somit zwar nicht die zugrunde liegende Cloud-Infrastruktur, hat aber die Kontrolle über die bereitgestellten Anwendungen.

In Anlehnung an (Fehling u. a. 2014) bezeichnen wir das zugehörige Service-Angebot als **elastische Plattform** und verstehen dies als eine Middleware zur Ausführung benutzerdefinierter Anwendungen, deren Kommunikation und Datenspeicherung über eine netzwerkbasierte Self-Service-Schnittstelle angeboten wird. Auf diese Weise können Anwendungskomponenten verschiedener Kunden auf einer gemeinsamen Middleware gehostet werden, die vom Anbieter bereitgestellt und gewartet wird. Diese Vereinheitlichung ermöglicht die gemeinsame Nutzung von Ressourcen und eine Automatisierung bestimmter Verwaltungsaufgaben auf Provider-Seite, z. B. die Bereitstellung von Anwendungen und die Verwaltung von Updates.

2.1.3 Software as a Service (SaaS)

Beim SaaS-Modell stellen Anbieter IT-Ressourcen in Form von für Menschen nutzbare Anwendungssoftware für Kunden bereit, um Self-Service, schnelle Elastizität und Pay-per-Use-Preise zu ermöglichen. Insbesondere kleine und mittlere Unternehmen verfügen oft nicht über die Arbeitskraft und das Know-how, um individuelle Softwareanwendungen zu entwickeln. Ferner sind viele Anwendungen zu Massenware geworden, die von vielen Unternehmen verwendet werden, aber kaum dazu beitragen, sich von Wettbewerbern abzuheben (siehe Abschnitt 14.2.1). Dies umfasst z. B. Office-Suiten, Software für die Zusammenarbeit oder Kommunikationssoftware.

Die dem Verbraucher zur Verfügung gestellte Fähigkeit besteht also bei SaaS darin, Anwendungen eines Anbieters zu nutzen, ohne die dafür erforderliche Infrastruktur oder Plattform betreiben zu müssen. Der Zugriff auf die Anwendungen erfolgt zumeist von verschiedenen Client-Geräten, wie z. B. einem Webbrowser (z. B. webbasierte E-Mail) oder über eine Programmschnittstelle.

Der Verbraucher verwaltet oder steuert die zugrunde liegende Cloud-Infrastruktur oder Cloud-Plattform einschließlich Netzwerk, Server, Betriebssystem, Speicher oder sogar einzelne Anwendungsfunktionen somit nicht selbst. Es sind jedoch – meist in sehr begrenztem Umfang – benutzerspezifische Konfigurationseinstellungen möglich (z. B. Anpassung der Benutzeroberfläche an Unternehmens-Styleguide-Vorgaben).



Anmerkungen für IT-Manager:

Cloud-natives Denken funktioniert auch „ohne Cloud“

Obwohl Public Cloud Computing in vielen Fällen sehr vorteilhaft sein kann, gibt es auch Anwendungsfälle, die als problematisch gelten und bei denen es schwierig ist, die Vorteile des Deployment-Modells Public Cloud zu nutzen, wie folgende Beispiele zeigen:

- **Kritische Infrastrukturen:** In Bereichen wie der Energieversorgung, dem Gesundheitswesen oder der öffentlichen Sicherheit werden kritische Infrastrukturen betrieben, deren Ausfall schwerwiegende Folgen haben kann. Hier ist oft ein Höchstmaß an Kontrolle und Sicherheit erforderlich, das schwierig mit Public Clouds zu erzielen ist.
- **Datenschutz und Compliance:** Insbesondere Unternehmen, die personenbezogene Daten (DSGVO) verarbeiten, müssen sicherstellen, dass ihre Daten sicher sind und den geltenden Datenschutz- und Compliance-Anforderungen entsprechen. Die Verarbeitung personenbezogener Daten in Public Clouds kann hier problematisch sein (siehe auch *Kapitel 17*). Es kann schwierig sein, sicherzustellen, dass die Daten sicher sind und die geltenden Vorschriften insbesondere bei internationalen Datentransfers eingehalten werden.
- **Kosten:** Obwohl öffentliche Clouds in vielen Fällen kostengünstiger sein können als die Bereitstellung und Verwaltung einer eigenen IT-Infrastruktur, gibt es genauso Anwendungsfälle, in denen die Nutzung der öffentlichen Cloud unwirtschaftlich sein kann. Dies gilt insbesondere dann, wenn die Anwendung hohe

Anforderungen an Leistung, Speicherplatz oder Bandbreite hat (also eine gewisse kritische Größe erreicht hat). Aber es trifft auch für Anwendungsfälle zu, die wenig Skalierungsbedarf haben und durch eine relativ konstante Grundlast gekennzeichnet sind. Hier können Public Clouds ihre Kostenvorteile oft nur teilweise ausspielen.

Oft wird (unbewusst) unter einem Cloud-nativen Ansatz implizit die Bereitstellung in Public Clouds angenommen. Das ist aber eine verkürzte Betrachtung, denn dabei wird übersehen, dass es auch die Bereitstellungsmodelle Private, Hybrid oder Community Cloud gibt. Cloud-native Anwendungen funktionieren mit allen genannten Bereitstellungsmodellen gleichermaßen und kommen damit grundsätzlich auch für eher als problematisch angesehene Anwendungsfälle in Betracht, die sich oft mittels Private-Cloud-Ansätzen in den Griff bekommen lassen.

Cloud-native fokussiert nicht primär das Bereitstellungsmodell

Auch für Private Clouds bieten Cloud-native Technologien viele Vorteile. Hier geht es dann weniger um die Migration in die Public Cloud, sondern vielmehr um die Modernisierung und Standardisierung einer bestehenden, oft historisch gewachsenen Infrastruktur und die Standardisierung des Betriebs von Software. Insbesondere für Manager, die in einer eher klassisch geprägten Unternehmens-IT groß geworden sind, ist es wichtig zu verstehen, dass Cloud-native Technologien nicht nur mit Public Clouds funktionieren. Vielmehr handelt es sich um einen Ansatz, der auf modernen Entwicklungsmethoden, Containerisierung und Automatisierung basiert und unabhängig davon eingesetzt werden kann, ob die Infrastruktur privat, öffentlich, vor Ort oder gänzlich anders bereitgestellt wird.

Cloud-native fokussiert die Standardisierung des Betriebs von Anwendungen

Der Einsatz von Cloud-nativer Technologie ermöglicht es, eine agile und flexible Infrastruktur aufzubauen, die auf die Bedürfnisse von Entwicklern und Benutzern zugeschnitten ist. Mit Containern und Microservices lassen sich Anwendungen schneller entwickeln und bereitstellen, was zu einer höheren Produktivität und einem besseren Kundenerlebnis führt.

Hierfür werden etablierte Technologien und Methoden wie Container-Laufzeitumgebungen, Container-Orchestratoren wie Kubernetes und Deployment-Pipelines für die Bereitstellung von standardisierten Anwendungskomponenten (Container) in einer privaten Cloud-Umgebung genutzt. Diese Technologien ermöglichen es Unternehmen, ihre Anwendungen in einer hochverfügbaren und skalierbaren Umgebung zu containerisieren und wesentlich standardisierter zu betreiben.

Cloud-native fördert Agilität und You-Build-It-You-Run-It Ansätze

Ein weiterer, oft übersehener Vorteil der Cloud-nativen Technologie ist die Möglichkeit, Entwicklern und Anwendern ein Self-Service-Modell anzubieten, ähnlich wie bei der öffentlichen Cloud. Dadurch können sie schnell und einfach neue Anwendungen und Dienste bereitstellen, ohne auf die Unterstützung zentraler IT-Teams angewiesen zu sein. Dies beschleunigt insbesondere agile Bottom-up-Entwicklungen.

Das Konzept des Service Ownership ist dabei eine hilfreiche agile Arbeitsweise, die auf der Idee basiert, dass das Team, das für die Entwicklung einer Anwendung (Dev) oder eines Dienstes verantwortlich ist, auch für deren Betrieb und Wartung (Ops) zuständig ist und daher oft als DevOps bezeichnet wird.

„Cloud-native Denken“ ist also ...

... etwas anderes als die Auswahl eines Cloud-Anbieters. Bei Cloud-native geht es in erster Linie darum, Anwendungen und Dienste von Grund auf für Cloud-Infrastrukturen und -Plattformen zu entwickeln und zu optimieren. Im Gegensatz zu herkömmlichen Anwendungen, die oft auf lokalen Servern oder in Rechenzentren auf virtuellen Maschinen laufen, werden Cloud-native Anwendungen speziell für die Skalierbarkeit von Cloud-Infrastrukturen und -Plattformen konzipiert und optimiert. Ein Unternehmen kann dabei problemlos sein eigener Provider sein, was manchmal aufgrund von Kritikalität oder regulatorischen Anforderungen durchaus sinnvoll ist. Da insbesondere im Public Cloud Computing eine hohe Preistransparenz aufgrund des Pay-as-you-go-Kostenmodells existiert, basiert das Cloud-Native-Modell auf einer Reihe von Grundsätzen, die darauf ausgerichtet sind, die Ressourcen möglichst effektiv zu nutzen (und damit bspw. auch den CO₂-Footprint zu reduzieren). Davon profitieren auch Anwendungen, die ausschließlich in Private Clouds betrieben werden sollen:

- **Skalierbarkeit:** Cloud-native Anwendungen müssen schnell und effektiv auf sich ändernde Anforderungen reagieren. Sie präferieren horizontale gegenüber vertikaler Skalierung.
- **Verfügbarkeit durch Automatisierung:** Cloud-native Anwendungen müssen in der Lage sein, mit Ausfällen oder Störungen umzugehen, indem sie ihre Funktionalität beibehalten oder schnell wiederherstellen, einschließlich der Verwendung verteilter Architekturen und Automatisierung, um Ausfälle zu minimieren.
- **Agilität:** Cloud-native Anwendungen sollten schnell und agil entwickelt, getestet und bereitgestellt werden können, was eine enge Zusammenarbeit zwischen Entwicklern, IT-Betrieb und anderen beteiligten Teams erfordert.
- **Microservices:** Cloud-native Anwendungen werden häufig in kleinere, unabhängige Dienste, sogenannte Microservices, unterteilt. Diese Dienste können unabhängig voneinander entwickelt, getestet und bereitgestellt werden, was die Flexibilität und Skalierbarkeit erhöht.
- **Standardisierung des Betriebs:** Container sind leichtgewichtige, portable Einheiten, mit denen sich Anwendungen und Dienste schnell erstellen, testen, bereitstellen und standardisiert betreiben lassen.

Cloud-native ist also vielmehr eine Denkweise, wie Anwendungen entwickelt und standardisiert betrieben werden sollen, die ressourceneffizient und ausfallsicher sind. Dabei wird die Betriebserfahrung von Generationen von IT-Administratoren in Form automatisierter Orchestrationsprozesse externalisiert und genutzt. Dabei spielt es keine Rolle, ob nun Amazon, Google, Microsoft, Alibaba oder das eigene Rechenzentrum diese Systeme betreibt. Weder der Standort noch ein Anbieter ist entscheidend für Cloud-natives Denken!

■ 2.2 Cloud-Ökonomie

Alle genannten Service-Modelle (IaaS, PaaS, SaaS) folgen dabei denselben wirtschaftlichen Gesetzmäßigkeiten. Beim sogenannten Pay-as-you-go-Kostenmodell werden nur die Ressourcen abgerechnet, die auch tatsächlich von einem Kunden angefordert werden. Aus Sicht des Kunden besteht also das wirtschaftliche Interesse vor allem darin, Cloud-Systeme mit einem möglichst geringen „Over-Provisioning“ zu betreiben, also Lastkurven mittels Skalierung möglichst eng und schnell folgen zu können (siehe Bild 2.4). Dies ist in klassischen Rechenzentren nicht – oder nur sehr begrenzt – möglich.

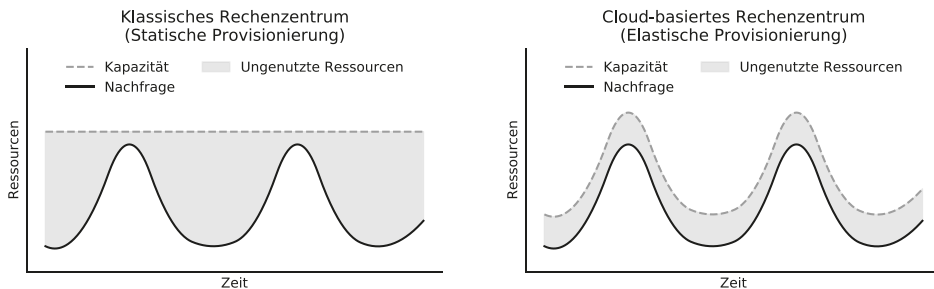


Bild 2.4 Statische und elastische Provisionierung von Ressourcen

2.2.1 Eignung von unterschiedlichen Arten von Workloads

Die Betrachtung von Workloads ist naturgegeben immer sehr anwendungsfallspezifisch, und man muss vorsichtig sein, nicht zu übergeneralisierende Ratschläge zu geben. Dennoch lassen sich unterschiedliche Workload-Arten ausmachen, die ökonomisch unterschiedlich geeignet für Cloud Computing sind. Dem Leser sei an dieser Stelle das Studium von (Weinman 2011) empfohlen, dessen Überlegungen hier zusammenfassend dargestellt werden.

Eine Pay-per-Use-Lösung macht immer dann offensichtlich Sinn, wenn die Stückkosten für On-Demand-Cloud-Services c niedriger sind als dedizierte, eigene Kapazitäten d . Oft können Cloud-Provider diesen Kostenvorteil bieten – aber nicht immer. Dies hängt leicht nachvollziehbar von den internen Kostenstrukturen eines Unternehmens ab und ist somit hochgradig unternehmensspezifisch.

Obwohl es kontraintuitiv erscheint, macht eine reine Cloud-Lösung aber auch in Szenarien Sinn, in denen die Stückkosten c höher als die Kosten für eigene Kapazitäten d sind. Allerdings nur, solange das Verhältnis von Spitzenlast p zu Durchschnittslast a der Nachfragekurve höher ist als das Kostenverhältnis der Stückkosten von On-Demand-Kapazität c zu dedizierter Kapazität d .

$$\frac{c}{d} < \frac{p}{a} \Leftrightarrow c < d \frac{p}{a} \Rightarrow c_{\max} := d \frac{p}{a}$$

Mit anderen Worten: Selbst wenn Cloud-Dienste doppelt so viel kosten wie In-House-Dienste, ist eine reine Cloud-Lösung für solche Bedarfskurven sinnvoll, bei denen das Verhältnis von

Spitzenwert zu Durchschnittswert zwei zu eins oder höher ist. Dies ist in einer Vielzahl von Branchen öfter der Fall, als man annehmen würde. Der Grund dafür ist, dass die dedizierte Lösung mit fester Kapazität für den Spitzenbedarf gebaut werden muss, während die Kosten der On-Demand-Pay-per-Use-Lösung proportional zum Durchschnitt sind (siehe auch Bild 2.4).

Je größer das Peak-to-Average-Verhältnis $\frac{P}{a}$ also ist, desto eher ist ein Anwendungsfall (rein ökonomisch betrachtet) für cloud-basierte Lösungen interessant. Betrachten wir vor diesem Hintergrund einmal die folgenden prototypischen Workloads, die so entweder in Reinform oder in überlagerten Kombinationen (z. B. periodischer Workload, der durch einen kontinuierlich steigenden Workload überlagert wird) im echten Leben häufig anzutreffen sind.

Statische Workloads (siehe Bild 2.5 A) sind durch ein mehr oder weniger flaches Lastprofil über die Zeit innerhalb bestimmter Grenzen gekennzeichnet. Eine Anwendung mit statischem Workload wird kaum von elastischen Infrastrukturen oder Plattformen profitieren können, da die Anzahl der benötigten Ressourcen konstant ist. Diese Arten von Workloads sind aber eher selten.

Häufiger sind hingegen periodische Aufgaben und Routinen (siehe Bild 2.5 B), zum Beispiel monatliche Gehaltsabrechnungen, monatliche Telefonrechnungen, jährliche Autoinspektionen, wöchentliche Statusberichte oder die tägliche Nutzung der öffentlichen Verkehrsmittel während der Hauptverkehrszeit. Solche Aufgaben und Routinen treten in wohldefinierten Intervallen auf und erzeugen daher **periodische Workloads** in der Nutzung involvierter IT-Systeme. Aus Kundensicht besteht das Kosteneinsparungspotenzial bei periodischen Lasten in der Außerbetriebnahme von Ressourcen in Nicht-Spitzenzeiten.

Als Spezialfall der periodischen Workloads können die Spitzen der periodischen Auslastung in einem sehr langen Zeitraum auch in Form **einmaliger/seltener Workloads** auftreten (siehe Bild 2.5 C). Oft ist diese Spitze im Voraus bekannt, da sie mit einem bestimmten Ereignis (z. B. olympische Spiele alle vier Jahre) oder einer Aufgabe korreliert. In solchen Szenarien können die Bereitstellung und Außerbetriebnahme von IT-Ressourcen oft als manuelle Aufgaben realisiert werden, da sie zu einem bekannten Zeitpunkt erfolgen.

Zufällige Workloads sind eine Verallgemeinerung der periodischen Workloads, da sie Elastizität erfordern, aber nicht vorhersehbar sind (siehe Bild 2.5 D). Solche Workloads treten in der realen Welt recht häufig auf. Hier sind die ungeplante Bereitstellung und Außerbetriebnahme von IT-Ressourcen erforderlich. Die notwendige Bereitstellung und Außerbetriebnahme von IT-Ressourcen müssen daher automatisiert erfolgen, um die Anzahl der Ressourcen an die sich ändernde Last anzupassen.

Bei vielen Anwendungen ändert sich auch die Last kontinuierlich über einen längeren Zeitraum. Häufig sind solche Lasten in Form eines Basistrends als Hintergrund-Workload in anderen Workloads (z. B. periodischen Workloads) enthalten. Sich **kontinuierlich ändernde Workloads** sind durch ein kontinuierliches Wachstum oder einen kontinuierlichen Rückgang der Auslastung gekennzeichnet (siehe Bild 2.5 E/F). Rein wirtschaftlich ist es dabei egal, ob ein Workload steigt oder sinkt, denn der Flächeninhalt (also die Einsparung) ergibt sich ja aus der Differenz der statischen und elastischen Provisionierungskurven. Der Bedarf persistenter Speichers unterliegt oft solch einem kontinuierlich wachsenden Trend. Es wird in vielen Anwendungsfällen eben mehr gespeichert als gelöscht.

Wenn man diese Workloads hinsichtlich ihres $\frac{P}{a}$ aufsteigend sortiert, erhält man folgende rein ökonomische Eignungsreihenfolge von Workloads für das Cloud Computing:

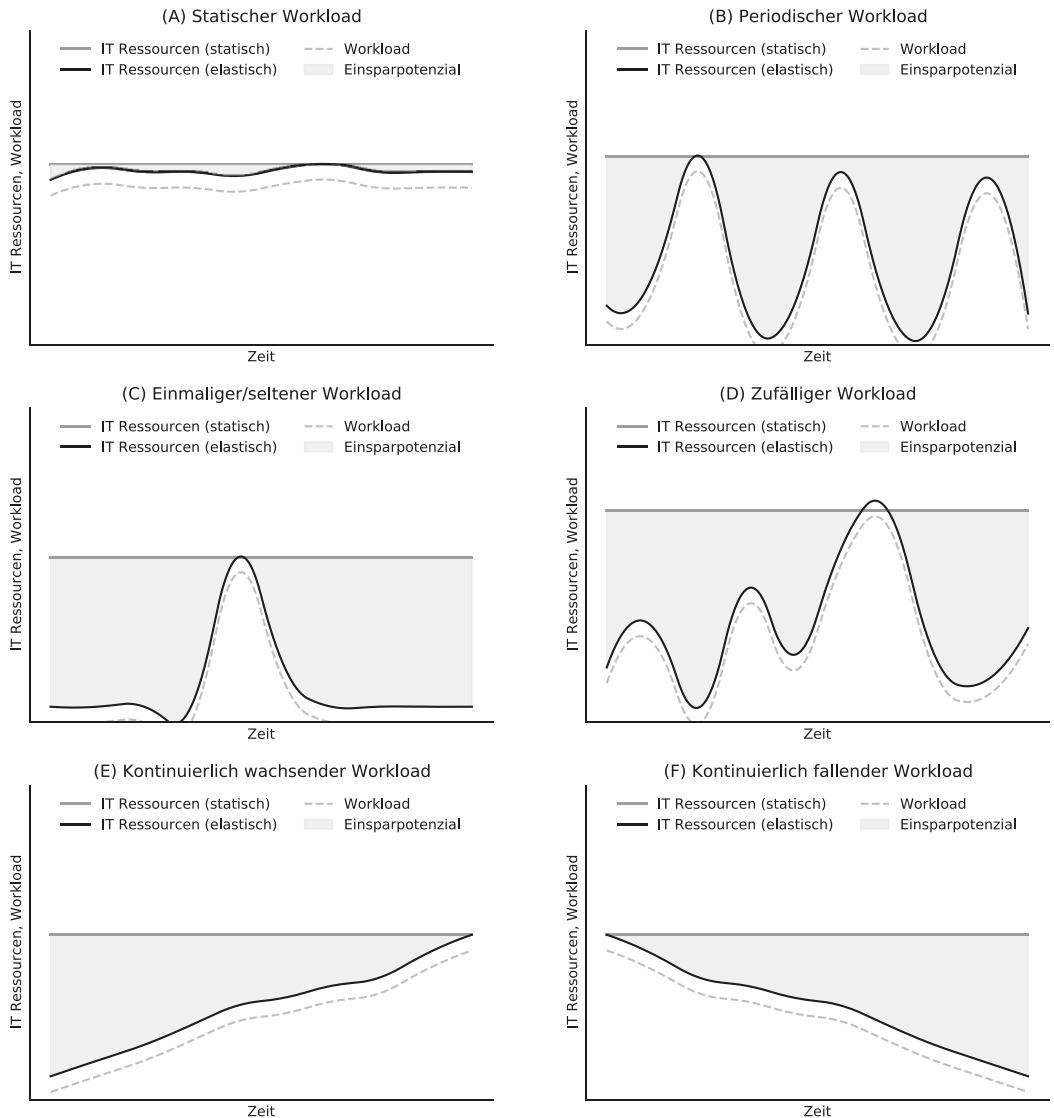


Bild 2.5 Zu berücksichtigende Workloads im Cloud Computing

- Statische Workloads (eher ungeeignet, siehe Bild 2.5 A)
- Kontinuierlich steigende/sinkende Workloads (siehe Bild 2.5 E/F)
- Zufällige und periodische Workloads (siehe Bild 2.5 B/D)
- Einmalige/seltene Workloads (extrem geeignet, Bild 2.5 C)

Für einen konkreten Anwendungsfall ist dieses $\frac{p}{a}$ natürlich immer genau zu bestimmen.

Dennoch hilft das Verständnis dieser grundsätzlichen Zusammenhänge erheblich dabei, überhaupt erst einmal interessante Anwendungsfälle zu identifizieren und uninteressante

Anwendungsfälle auszuschließen. Grundsätzlich ermöglicht die Elastizität von Cloud-Infrastrukturen und -Plattformen, Ressourcen mit der gleichen Rate bereitzustellen oder freizugeben, mit der sich die Arbeitslast eines Dienstes ändert, um diese Effekte für sich zu nutzen.

2.2.2 Effekt von Zuteilungsdauer und Ressourcengröße

Wie wir also sehen, sind Cloud-Ressourcen vor allem dann wirtschaftlich, wenn Lastschwankungen in einem Anwendungsfall auftreten. Die Kosten pro Cloud-Ressource können sogar deutlich höher als die In-House-Kosten liegen – solange das Verhältnis von Cloud zu In-House-Kosten nicht das Verhältnis von Spitzen- zu Durchschnittslast übersteigt.

Ziel ist also, im Betrieb eine möglichst niedrige Durchschnittslast zu ermöglichen (bzw. die Fläche zur Abdeckung der Lastkurve zu minimieren). Hierzu strebt man im Betrieb an, Lastkurven möglichst eng zu folgen. Kann man sich möglichst eng an Lastkurven „anschmiegen“, erzeugt dies wenig Over-Provisioning. Viele Innovationen des Cloud-native Computings wie beispielsweise Container- und FaaS-Technologien sind im Kern auf diese Erkenntnis zurückzuführen. Bei der Ressourcenzuteilung lässt sich dabei letztlich an zwei Stellschrauben drehen.

1. Man kann Ressourcen feingranularer zuteilen (vertikale Stellschraube).
2. Man kann Ressourcen kürzer zuteilen (horizontale Stellschraube).

Bild 2.6 zeigt den Effekt beider Stellschrauben (Ressourcengröße und Zuteilungsdauer) auf den Ressourcenverbrauch (und damit die Kosten) am Beispiel eines synthetischen periodischen Workload-Verlaufs.

Wie Bild 2.6 zeigt, ermöglichen es kleinere Ressourcengrößen und kürzere Zuteilungsdauern, Lastkurven enger folgen zu können. Damit kann das Over-Provisioning verringert werden. Dies spart letztlich Geld im Betrieb eines Cloud-nativen Systems. An dem – zugegeben synthetischen – Beispiel von Bild 2.6 zeigt sich dennoch, dass sich durch die Reduzierung von Ressourcengrößen und kürzere Zuteilungsdauern der rechnerische Ressourcenbedarf durchaus halbieren lässt. Dies ist natürlich immer von den dahinterliegenden Workload-Arten und dem Anwendungsfall abhängig. Auch noch größere Einsparungen sind nicht ungewöhnlich.

Diese einfache Erkenntnis hatte in den letzten Jahren einen tiefgreifenden Einfluss auf Cloud-native Architekturen und Technologien (Kratzke und Quint 2017). So konnte man in den vergangenen Jahren beobachten, wie diese beiden Stellschrauben (Zuteilungsdauer und Ressourcengröße) systematisch reduziert wurden. Während in der Anfangszeit des Cloud Computings virtuelle Maschinen üblicherweise auf Stundenbasis abgerechnet wurden, ist dies im Verlaufe der Zeit auf eine dreißigminütige, dann fünfzehnminütige bis schließlich zu einer minutengenauen oder mittlerweile sogar einer sekundengenauen Abrechnung bei vielen Providern umgestellt worden. Auch die Ressourcengröße wurde durch Technologien reduziert. Mittels IaaS kommt man nicht wirklich effizient unter die Auflösung von einer vCPU. Doch mittels der zunehmend beliebteren Container-Technologie sind wesentlich feingranularere Ressourcen möglich (siehe Kapitel 8), mit denen man problemlos unter diese 1 vCPU-Schwelle kommt. Auch die seit einigen Jahren beliebter werdende Technologie Function as a Service (FaaS, siehe Kapitel 10) kombiniert letztlich feingranularere Container mit einer Reduktion der zeitlichen Zuteilungsdauer im Subsekunden-Bereich. FaaS erlaubt es sogar, Ressourcen komplett auf null zu skalieren, wenn ein System in einem Zeitintervall

keine Aufgaben zu verarbeiten hat. Daran zeigt sich, dass viele Trendtechnologien zur feingranulareren Ressourcenallokation im Cloud-nativen Umfeld ihren Grund auch immer in der innewohnenden Cloud-Ökonomie haben – auch wenn dies häufig nicht (mehr) bewusst wahrgenommen wird.

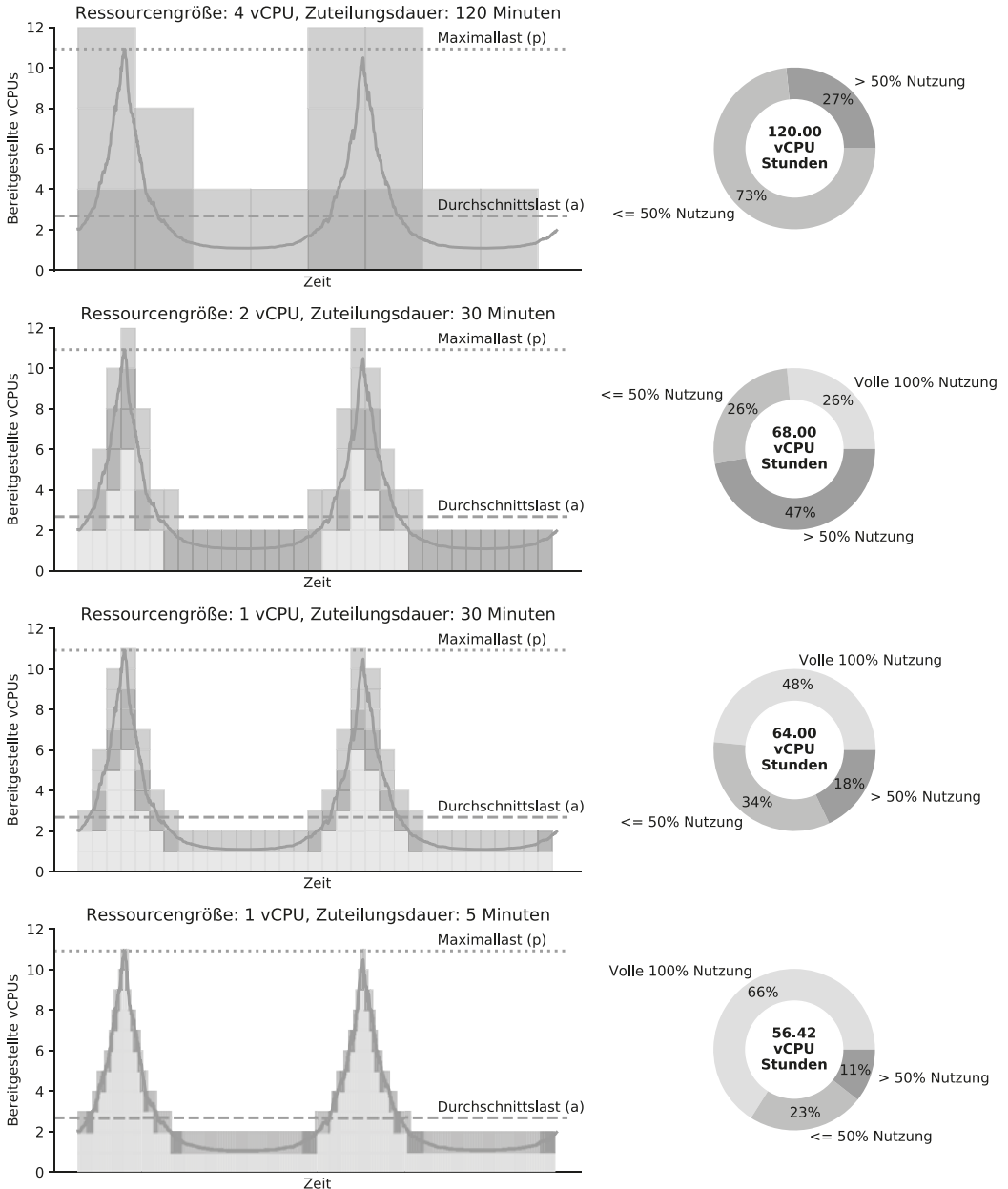


Bild 2.6 Effekt von Ressourcen- und Zuteilungsdauer

■ 2.3 Entwicklung der letzten Jahre

Cloud Computing ist vor etwa zehn bis 15 Jahren entstanden. Dabei wurden in der ersten Adoptionsphase bestehende IT-Systeme lediglich in Cloud-Umgebungen übertragen, ohne das ursprüngliche Design und die Architektur dieser Anwendungen zu ändern. Multi-Tier-Anwendungen wurden lediglich von dedizierter Hardware auf virtualisierte Hardware in der Cloud migriert. Cloud-Systemingenieure haben im Laufe der Jahre allerdings bemerkenswerte Verbesserungen an Cloud-Plattformen (PaaS) und -Infrastrukturen (IaaS) vorgenommen und mehrere technische Trends etabliert, die derzeit zu beobachten sind. Ein wesentlicher Treiber hierfür sind die erläuterten ökonomischen Gesetzmäßigkeiten des Pay-per-use-Prinzips. Wer Cloud-native Systeme wirtschaftlich betreiben will, muss die Ressourcennutzung optimieren und minimieren.

Cloud-Infrastrukturen (IaaS) und -Plattformen (PaaS) sind daher insbesondere für den elastischen Betrieb von Cloud-nativen Anwendungen gebaut, um Over-Provisioning von Ressourcen zu vermeiden. Unter Elastizität versteht man den Grad, in dem sich ein System an Laständerungen anpasst, indem es automatisch Ressourcen bereitstellt und entnimmt. Ohne diese Elastizität ist Cloud Computing aus wirtschaftlicher Sicht sehr oft nicht sinnvoll.

Mit der Zeit lernten Systemingenieure, diese Elastizitätsoptionen moderner Cloud-Umgebungen besser zu verstehen. Schließlich wurden Systeme für solche elastischen Cloud-Infrastrukturen von Grund auf entworfen, die dank neuer Deployment- und Design-Ansätze wie Container (siehe Kapitel 8), Microservices oder serverloser Architekturen (siehe Kapitel 12) den bereitzustellenden Ressourcenbedarf der zugrunde liegenden Computing-Infrastrukturen minimieren. Diese Designabsicht wird oft unbewusst mit dem Begriff „Cloud-native“ ausgedrückt.

Die Maschinenvirtualisierung hat sich insbesondere deshalb durchgesetzt, um eine Vielzahl von Bare-Metal-Maschinen zu konsolidieren und so die physischen Ressourcen in Rechenzentren effizienter nutzen zu können. Diese Maschinenvirtualisierung bildet bis heute das technologische Rückgrat des (IaaS-)Cloud Computings. Virtuelle Maschinen sind zwar leichtgewichtiger als Bare-Metal-Server, aber sie sind nicht unbedingt als leichtgewichtig zu bezeichnen, vor allem in Bezug auf ihre Image-Größen. Diese IaaS-Ebene wird vor allem in Kapitel 7 behandelt.

Vor diesem Hintergrund wurden leichtgewichtiger Container entwickelt. Container erlebten ihren Siegeszug primär, weil sie einerseits die Art und Weise der standardisierten Bereitstellung von Anwendungskomponenten vereinfachen. Container erhöhen aber auch die Auslastung der virtuellen Maschinen, da sie auf leichtgewichtigeren Betriebssystem-Virtualisierungskonzepten beruhen. Man kann also meist deutlich mehr Container auf einem physischen Host betreiben als virtuelle Maschinen. Wir werden uns mit diesen Aspekten vor allem in Kapitel 8 und in Kapitel 9 befassen. Dennoch sind Container, obwohl sie leichtgewichtig und schnell skalierbar sind, immer noch Always-on-Komponenten. Es muss also immer einen „letzten“ Container geben, der Requests bearbeiten kann. Zumindest dieser „letzte“ Container fällt damit weiterhin in den Bereich eines statischen Workloads, also dem aus Kundensicht teuersten Workload für Cloud Computing.

Daher wurden Function-as-a-Service-(FaaS-)Ansätze entwickelt, die eine Art Time-Sharing von Containern auf darunterliegenden Container-Plattformen anwenden. Wir werden uns vor allem in Kapitel 10 mit diesen Aspekten befassen. Bei FaaS werden nur Einheiten (Funktionen) ausgeführt, die Requests zu bearbeiten haben. Durch diese zeitlich geteilte Ausführung von Containern auf der gleichen Hardware ermöglicht FaaS sogar eine Skalierbarkeit bis auf null. Studien konnten diese verbesserte FaaS-Ressourceneffizienz sogar monetär messen (Villamizar u. a. 2017). All dies hat letztlich mit der Minimierung der statischen Workload-Anteile zu tun, die den ineffektivsten Workload für Cloud Computing ausmachen.

Rückblickend betrachtet wurde der Technologie-Stack zur Verwaltung von Ressourcen in der Cloud also im Laufe der Zeit durch zusätzliche Ebenen (Virtualisierung, Container Runtime, FaaS Runtime) erweitert und damit immer komplexer. Das folgte aber einem grundsätzlichen Trend – mehr Workload auf der gleichen Anzahl physischer Maschinen auszuführen, also die Ressourceneffizienz insgesamt zu erhöhen.

Stichwortverzeichnis

Symbole

- 1 vCPU-Schwelle 22, 84, 142
- 3-Tier-Architektur 252
- 12-Faktoren
 - Abhängigkeiten 91
 - Administrative Prozesse (update, backup, restore) 96
 - Build, Release, Run 93
 - Codebase 91, 93
 - Environment 95
 - Horizontale Skalierung 94
 - Konfigurationen 91
 - Logging 95
 - Port Binding 92
 - Skalierung über Prozesse 94
 - Umgebung 95
 - Unterstützende Services 92
- 12-Faktoren-Methodik 113, 143, 204

A

- Ablaufverfolgung 199
- Abstract Syntax Tree (AST) 315
- A/B-Tests 35
- A/B-Testszszenarien 220, 221
- Abwärtskompatibilität 175
- ACID 183
- Active Record 247
- Active Record-Pattern 247, 252
- Admission Controller 303
- Affinität 123
- Affinity 122
- Aggregat 248, 249, 250
- Aggregate Root 249
- Aggregatgrenze 249
- Aggregatwurzel 249
- Agilität 18
- Aktion 147
- Aktionspakete 149
- Alert-Manager 200, 205
- ALLOW-Regel 225
- Analysemodell 236
- Anforderungen 236
- Angriffsoberfläche 279
- Angriffsvektor 261, 264, 265, 280, 309, 310
- Anti-Corruption-Layer 243
- Anwendungsschicht 253
- Anwendungsvirtualisierung 66, 67
- API 175
- API-Gateway 177, 193, 194
- API-Versioning 167, 175
- APM 201
- AppArmor 295
- Append-Only-Log 250
- Architektur 39
 - DevOps-geeignet 33
 - Serverless 143
- Architekturelle Sicherheit 178
- Architekturmuster 251
- Asynchrone Architektur 173
- auditd 274
- Auditierbarkeit 251
- Auditierung 329
- Auditing 223, 273, 280, 308
- Audit-Protokolle 251
- Auftragsverarbeitung 338
- Authentication Policy 224
- Authentifizierung 223, 268, 271, 279, 281, 283, 289
 - Peer 225
 - Request 225
- Authorisation 223
- Authorisation Policy 224, 225
- Automatisierte Instrumentierung 215
- Automatisierung 18, 187
- Autorisierung 268, 279
- Autoskalierung 125
 - ereignisbasiert 152
 - horizontal, Pod 125

- AWS Lambda 151
- Azure Lambda 151
- B**
- Backend as a Service (BaaS) 191
- BASE 183
- Batch-Job 101
- Batch-System 207
- Beobachtbare Architekturen 199
- Beobachtbarkeit 40, 189
- Berechtigtes Interesse 337
- Best Practices 146, 351
- Betriebssystem-Virtualisierung 66
- Betriebszustand 107
- Big Five 1
- Binding Corporate Rules (BCR) 341, 343, 345
- Binpack 102
- Blackbox-Monitoring 205
- Blackbox-Tracing 208
- Blackbox-Überwachung 202
- Block-Storage 65
- Blue/Green-Deployment 34
- Blue/Green-Release 189
- Blueprint 107, 112
- Borg 105
- Bounded Context 187, 231, 239, 240, 252, 254
- Branching-Strategien 56
- Breaking-Change 166, 176, 188
- BSI 325
- BSI-C5 325
- BSI-Grundschutz 325
- BSI-Standard 100-3 325
- BSI-Standard 100-4 325
- BSI-Standard 200-1 325
- Build Phase 49
- Bulkhead 179
- C**
- CaaS 82
- Caching 94, 170, 186
 - clientseitig 186
 - Proxy-Caching 187
 - serverseitig 186
- Canary 219, 221
- Canary-Release 34, 189
- CAP-Theorem 183
- Chaos Engineering 32
- Checkpoint 255
- Chef 70
- Choreography-over-Orchestration 188
- CI/CD 49
- CIDR 288
- Circuit-Breaker 178, 189, 222
- CISA 308
- CISPE (Cloud Infrastructure Services Providers in Europe) 328
- CISPE Code of Conduct 329
- C-Level-Funktion 3
- Clientseitiges Tracing 214
- Client-Server 168, 170
- CLOUD Act 343, 344, 345, 346
- Cloud Compliance 322, 323
- Cloud Computing 11
 - NIST-Definition 11
- Cloud-native 2, 16, 24, 37, 351
 - Definition 40
- Cloud-native Computing Foundation (CNCF) 39, 109
- Cloud-Ökonomie 19, 142
- Cluster 100, 107, 122
- Cluster-Awareness 100
- Cluster-Scheduler 110
- CNCF 109
- CNI 110
- CO₂-Footprint 18
- Code Repository 49
- Command 254
- Command Execution-Modell 254
- Command Query Responsibility Segregation (CQRS) 185, 253
- Common Vulnerabilities and Exposures (CVE) 312
- Community Cloud 13, 17
- Compliance 16
- Config Map 113
- Constraint 122
- Container 24, 40, 41, 66, 79, 83, 109, 110, 188
 - Laufzeitumgebung 84
 - Runtime 84
- Container as a Service 82
- Container Breakout 266
- Container-Image 88
- Container Network Interface (CNI) 110
- Container Runtime Environment 88, 112, 280, 299
- Container Runtime Isolation 299
- Container Storage Interface (CSI) 110
- Content-Delivery-Netzwerks (CDN) 187
- Context Mapping 231, 241
- Continuous Deployment 49

Continuous Integration 49
 Controller 108
 Control Plane 217
 Conway's Law 165, 229, 241
 Copy-on-Write 86
 Core Subdomain 233, 248, 250, 251, 253, 254
 CQRS 185, 253
 Creative Commons-Lizenz (CC0) 7
 Cron-Job 116
 CRUD 172, 247, 253
 CSA Cloud Control Matrix (CCM) 327
 CSA STAR 327
 CSI 110
 Current State 108, 125
 Customer-Supplier 241, 243
 Cyber Attack Lifecycle 264, 273, 280

D

Daemon-Set 114, 117
 DAG Pipeline 52
 Dapper 208
 Data Breach 280
 Data Plane 217
 Datenbankbasierte Integration 167
 Datenbasierte Integration 167
 Datenkopplung 167, 188
 Datenlokalisierung 328
 Datenminimierung 336
 Datenschutz 16, 321
 Datenschutz-Folgenabschätzung 340
 Datentransfer in Drittländer 341
 Defense in Depth 223
 Dekomposition 161, 229
 Denial of Service 266, 285, 291, 292, 300
 DENY-Regel 225
 Dependency Injection 253
 Deployment 113, 114, 126
 Deployment-Pipeline 17, 31, 34, 49, 89, 266, 269, 298, 312, 315

- Job 50
- Phase 49
- Trigger 50

 Deployment Unit 40, 41, 66, 79, 165
 Deploy Phase 49
 Desired State 108, 125
 Development 56
 Development-Branch 58
 DevOps 18, 27, 90, 165, 194, 200

- Flaschenhalse 31
- Kultur 31

- Prinzipien des Feedbacks 32, 40
 - Prinzipien des Flow 29, 41
 - Work in Progress 30
 - Zyklus 29, 34
 Docker 83
 Dockerfile 88
 Domain-driven 187
 Domain-driven Design 229
 Domain-Event 232, 249, 250
 Domain Model-Pattern 248
 Domänenmodell 229, 232, 248, 250
 Domänenwissen 236
 Dominant Resource Fairness 103
 Double-Spending-Problem 144, 191, 193
 Downstream-Service 162
 DSGVO 16, 261, 321, 328, 329, 332, 335, 338, 343, 344
 Dumb-Middleware-with-Smart-Endpoints 188

E

Ebenen-Architektur 252
 Effektives Design 230
 Einwilligung 337
 Elasticsearch 201
 Elastisches System 174
 Elastizität 40, 143, 173
 Emulation 66
 Endbenutzer-Choreografie 193
 End-to-End-Tracing 212
 Enterprise Architecture Management (EAM) 239
 Entkopplung 65
 Environment 54, 56
 Ereignisbasiert 188
 Ereignisbasierte Integration 167, 173
 Ereignisbasierte Systeme 173
 Ereignisgesteuert 173
 Ereignisquelle 145
 ETL-Pattern 246
 EU Cloud Code of Conduct 329
 EU-US Privacy Shield 343, 345
 Event-driven 145
 Event-Emitting-Service 173
 Event-Sourcing 254
 Event-Sourcing-Pattern 250
 Event Store 250
 Eventual Consistency 184, 249
 Everything as Code, Deployment-Pipeline 50
 Evolutionäres Design 164, 229
 Execution-Monitor 104

Executor 105
 Exploit 264
 Exporter 204
 Extract-Transform-Load (ETL) 246
 Extraktion von Span-Kontexten 213

F

FaaS 126
 – Best Practices 146
 FaaS-Framework 149
 FaaS-Plattform 142, 190
 FaaS-Programmiermodell 145, 147, 190
 Fachlichkeit 229, 230, 248
 Fail early 222
 Fairness 102
 Fallacies of Distributed Computing 255
 Feature-Branch 58
 Feature Release 176
 Feature-Schalter 34
 Federal Risk and Authorization Management
 Program (FedRAMP) 331
 Feed-basierte Trigger 148
 Fehlertoleranz 102
 File-Storage 65
 Firewall 268, 277, 282
 Flooding 292
 Fluentd 201
 Foothold 265
 Function 200
 Function as a Service (FaaS) 22, 141
 Funktion 145, 147

G

GAE 80
 GAIA-X 1
 GDPR 321, 335, 343
 Gegenseitige Authentifizierung 224
 Gegenseitige TLS-Authentifizierung (mTLS) 224
 Generic Subdomain 234, 244
 Generische Subdomäne 234
 Gerichtete Pipeline 52
 Geschäftskonzept 187
 Geschäftslogik 246, 247, 250, 252, 253
 Gesetz von Conway 165
 Git-Flow 57
 GitHub-Flow 58
 GitLab CI/CD 50
 Google App Engine 80
 Google Cloud Functions 151
 Grafana 201

gRPC (gRPC Remote Procedure Call) 129,
 168, 210
 Grundsätze der Verarbeitung 336

H

Hadoop 104
 HashiCorp Configuration Language 75
 HATEOAS 171
 HCL 75
 Health Checking 129
 Health Insurance Portability and Accountability
 Act (HIPAA) 331
 Heroku 81
 Hexagonale Architektur 253
 Hierarchische Pipeline 53
 High-Level Container Runtime 87
 High-Level-Design 245
 Horizontale Pod-Autoskalierung 125
 Horizontale Skalierung 180
 Horizontal Pod Autoscaler 125
 Horizontal Pod Autoscaling (HPA) 152
 HPA 125
 HTTP-/REST-basierte Integration 167
 HTTPS 282
 Hybrid Cloud 14, 17
 Hypermedia as the Engine of Application State
 (HATEOAS) 171
 Hyperthread 122
 Hypervisor 65

I

IaC 69, 270, 271, 279
 IDEAL-Modell 39
 Idempotente Operation 180
 Idempotenz 180
 Image Registry 266
 Immutable 248
 Immutable Infrastructure 68
 Implementierungsdetail 166, 188
 Infrastructure as a Service (IaaS) 14, 15
 Infrastructure as Code 63
 Infrastruktur
 – als Code 69
 – elastisch 15
 Infrastrukturkomponente 253
 Infrastrukturschicht 253
 Ingress 113, 128, 280, 284, 285, 304
 In-Process-Komponenten 161
 Instrumentierung 202
 Instrumentierungsbibliothek 209

Instrumenting Library 200
 Integrations-Branch 58
 Integrität 337
 Interprozesskommunikation 167
 Intrusion Detection 273, 274, 280, 282, 307
 ISO 9001 324
 ISO/IEC 27001 325, 326
 ISO/IEC 27017 327
 ISO/IEC 27018 327
 Isolation 65, 67, 280
 Isolationsmechanismus 88
 Istio 217, 221, 224, 226
 IT-Sicherheit 261

J

Jaeger 201
 Job 114, 116, 200, 206, 207

K

Kanban 29
 KEDA 152
 - ScaledJob 153
 - ScaledObject 153
 Kerndomäne 233
 Kiali 226
 Kibana 201
 Knotenaffinität 123
 Kohäsion 229
 Kommunikationsmuster 245
 Konfigurations-API-Server 224
 Konfigurationsmanagement 70
 Konformist-Pattern 243
 Kontrollgruppe 220
 Kritische Infrastruktur 16
 Kritischer Pfad 209
 Kubeless 151
 Kubernetes 34, 105, 109, 200, 218
 - Affinität 123
 - API-Server 111, 112
 - Architektur 111
 - Cloud-Manager 111
 - Cluster Role 133
 - Controller-Manager 111
 - Daemon-Set 117
 - Deployment 115
 - Horizontal Pod Autoscaler (HPA) 125
 - Ingress 128, 168, 195
 - Job 116
 - Kubelet 112
 - Kube-Proxy 112

- Limit 121, 134
 - Master Node 111
 - Namespace 133
 - Network-Plug-in 111
 - Network Policy 135, 225
 - Persistent Volume Claim (PVC) 132
 - Persistent Volume (PV) 132
 - Quota 135
 - RBAC 133
 - Request 121
 - Resource Quota 134
 - Role 133
 - Role Binding 133
 - Scheduler 111
 - Secret 133
 - Selektor 122
 - Service 128, 181
 - Service-Account 133
 - Stateful-Set 119
 - Storage-Plug-in 111
 - Worker Node 112
 - Workload 114
 Kubernetes Hardening Guide 266, 279, 307
 Kubernetes-Ressourcen 112

L

Lastausgleich 181
 Lateral Movement 264, 279, 289
 Laufzeitumgebung 67
 Layered Architecture 252
 Ledger 250
 Let's Encrypt 283
 LimitRange 292
 Limits 121
 Liveness Probe 129
 Load Balancer 127, 282
 Load Balancing 181, 184
 Local Procedure Call (LPC) 168
 Log-Aggregation 202, 276, 307
 Logge auf stdout 204
 Logging 40, 189, 199, 202, 266
 Logikebene 252
 Log-Level 203
 - Debug 203
 - Error 203
 - Fatal 203
 - Info 203
 - Trace 203
 - Warning 203
 Lokalität 102

Lose Kopplung 164, 229
 Low-Level Container Runtime 87

M

Machtgefälle 241
 Machtverhältnis 245
 Manifest 110, 112
 Man-in-the-Middle-Angriff 223, 224
 Marathon 114
 Materialien 7, 43, 78, 97, 139, 257
 Materialien (Slides, Handouts)
 - 12-Faktoren-Methodik 97
 - Architektur-Pattern für Core Subdomains (DDD) 257
 - Architektur-Pattern für Supporting Subdomains (DDD) 257
 - Beobachtbarkeit 228
 - Betriebssystemvirtualisierung 97
 - Cloud Computing Historie 43
 - Cloud-native Systeme 43
 - Cloud-Ökonomie 43
 - Container-Orchestrierung 139
 - Context Mapping (DDD) 257
 - Deployment-Pipelines 61
 - Deployment Units (Container) 97
 - DevOps 43, 61
 - DevOps-geeignete Architekturen 61
 - Docker 97
 - Domain-driven Design 257
 - Effektives Software-Design 257
 - FaaS-Plattformen 156
 - FaaS-Programmiermodell 156
 - Function as a Service (FaaS) 156, 197
 - Immutable Architectures 78
 - Infrastructure as a Service 78
 - Infrastructure as Code 78
 - Kubernetes 139
 - Kubernetes Blueprints (Manifests) 139
 - Logging 228
 - Materialien (Slides, Handouts) 97
 - Metriken und Monitoring 228
 - Microservices 197
 - Pattern für Geschäftslogiken (DDD) 257
 - Platform as a Service (PaaS) 139
 - Prinzipien des Feedbacks 61
 - Prinzipien des Flow 61
 - Resilienz 228
 - Serverless Computing 156, 197
 - Service-Meshs 228
 - Sheduling 139

- Sicherheit 228
 - Strategisches Design (DDD) 257
 - Subdomains (DDD) 257
 - Taktisches Design (DDD) 257
 - Telemetriedaten 228
 - Terraform 78
 - Tracing 228
 - Traffic-Management 228
 - Ubiquitous Language (DDD) 257
 - Vagrant 78
 - Visualisierung von Verkehrstopologien 228
 - Was ist Cloud Computing? 43
 Mehrdeutiger Begriff 238
 Memory-Ballooning 65
 Mentales Modell 239
 Mesos 34, 103, 106, 110, 114
 Messaging 181
 Metriken 40, 189, 199, 204
 - Messung (Gauge) 206
 - Verteilung (Histogramm) 206
 - Zähler (Counter) 206
 Metrikinstrumentierung 207
 Microservice 18, 31, 39, 162, 231
 Microservice-Architektur 144, 199, 229
 Microservice-basierte Anwendung 163
 Millicore 122
 Monitoring 199, 204, 266
 Monolithische Anwendung 163
 Monopolisierung 285, 291
 Monorepository 54
 mTLS 224, 225
 Multi-Cloud 72
 Multiplizität 65
 Multi-Tenancy 133, 135, 225
 Mutable 248
 Mutual Authentication 224

N

Nachrichtenorientiertes System 174
 Namespace 280, 287
 Network Policy 287, 289, 303
 Netzwerkisolation 279, 286
 Netzwerkpartition 189
 Nomad 34, 110
 NoSQL 185
 NoSQL-Datenbanken 183
 NSA 307, 308

O

OAuth 283, 284

- Objektmodell 254
 - lesend 253
 - schreibend 253
- Objektrelationales Mapping (ORM) 247
- Observability 40, 189, 202, 265, 276
- Observable 174
- OCI 84, 109
- Omega 106
- One-Service-per-Container 189
- Online-System 207
- OPA-Policy 304
- OpenAPI 244
- Open-Container-Initiative 84
- Open-Host-Service 244
- Open Policy Agent 303, 305
- OpenTracing-API 209, 212
- OpenWhisk 147, 151
- Orchestrierung 99, 107
- Orchestrierungsplattform 34, 188
- Orchestrierungsregelkreis 109, 126
- Ortsunabhängigkeit 174
- Out-of-Process-Komponenten 161
- Output Stream 96
- Overlay Network 110
- Over-Provisioning 24

- P**
- PaaS 79, 82
- Para-Virtualisierung 65
- Partnerschaftliche Kooperation 241
- Partnerschaftsmodell 241
- Partnership 241
- Patching 268, 270, 280
- Pattern 351
- Pay-as-you-go 2, 18, 19
- Payment Card Industry Data Security Standard (PCI DSS) 332
- Peak-to-Average 20
- Peer-to-Peer Computing 191
- Persistent Volume 132
- Persistent Volume Claim 113, 132, 301
- Persistenzebene 252
- Personenbezogene Daten 16, 335
- Phasen- 51
- Phishing 266
- Platform as a Service (PaaS) 14, 15, 79, 82
- Plattform
 - Container 82
 - elastisch 15, 41
 - Function as a Service (FaaS) 143
 - PaaS 80
- Pod 110, 152
- Pod-Affinität 124
- Pod Hardening 289, 297
- Pod Security Policy 296
- Pod Security Standard 297
- Policy 280, 303
- Policy Enforcement Point (PEP) 224
- Polyglotte Persistenz 254
- Polyglott Programming 67
- Ports & Adapter-Pattern 253
- Präsentationsebene 252
- Prinzip des geringsten Privilegs 291
- Private Cloud 13, 17
- Privilege Escalation 289, 297
- Probe 129
- Production 56
- Produktivsystem 33
- Projektion
 - asynchron 255
 - synchron 254
- Projektions-Engine 255
- Prometheus 201
- Protocol Buffers 168
- Protokollierung 40, 202
- Provisionierung 68
 - deklarativ 70
 - imperativ 70
 - Pull-basiert 70
- Proxy 216, 224, 282
- Prozessisolation
 - Control Group (cgroup) 86
 - Namensräume für Dateisysteme 86
 - Namespace 84
 - Priorisierung 86
 - Process Capabilities 85
 - Quota 86
- PSP 296
- PSS 297
- Public Cloud 13, 16
- Publish/Subscribe 182, 250
- Puppet 70
- Push-Gateway 206, 207
- PVC 113
- Python 6

- Q**
- Qualitätsmanagementsystem (QMS) 324
- Query 254
- Querying-System 205

Queueing 181, 250
 Quota 292, 301

R

RAFT 119
 Rate Limiting 281, 284
 RBAC 133, 134, 135, 290
 ReactiveX-Programmiermodell 174
 Readiness Probe 130
 Reaktive Erweiterung (Rx) 174
 Reaktives System 173
 Rechenschaftspflicht 337
 Rechtmäßigkeit 336
 Rechtsordnung 321
 Regel 145, 148
 Regelkreis 108, 125
 Regelkreis-basierte Orchestrierung 109
 Region 63
 Rego 305
 Regulatorische Anforderungen 321
 Release 33
 Releaserisiken 34
 Remote Procedure Call (RPC) 167, 177
 Replay-Angriff 224
 Replaying Time Machine 251
 Replicas 118
 Replica-Set 114
 Replication Controller 113
 Representational State Transfer (REST) 170
 Requests 121
 Resilient Software Design 32
 Resilienz 32, 173, 221
 Resilienz-Pattern 221
 Responsivität 173
 Ressourceneffizienz 25
 Ressourcengröße 22
 Ressourcenkontingent 135
 REST 39, 129, 170, 180, 188, 192, 196, 210
 REST-API 177
 Restart Policy 117
 Reverse-Proxy 187, 194
 Richtigkeit 336
 Richtlinie 280, 303
 Role-based Access Model (RBAC) 133
 Rolling-Updates 34
 Rootkit 274
 RPC
 - Bidirectional-Streaming 169
 - Client-Streaming 169
 - Server-Streaming 169

- Unary 169
 Runtime 67

S

Safe Harbor Abkommen 343
 Sandbox 81
 Scale-to-Zero 126, 141, 154, 162
 Scaling for Reads 184
 Scaling for Writes 184
 Scaling out 180
 Scaling up 180
 Scheduler 100, 122
 - 2-Level 105
 - monolithisch 105
 - Shared-State 106
 Scheduling 99
 - Algorithmus 102
 - Architekturen 104
 - Constraints 121
 - einfache Algorithmen 102
 - kapazitätsbasierte Algorithmen 103
 - multidimensionale Algorithmen 103
 SCHREMS I 343, 344
 SCHREMS II 343, 345, 346
 Schutz personenbezogener Daten 328, 329
 Schwachstellen-Scanning 280, 311, 313,
 316, 317
 Seccomp 295
 Secret 113, 291
 Secure Shell 271
 Security by Default 223
 Security Context 293, 294
 Security Group 277
 Selektor 122, 225
 Self-Healing 39, 109
 Self-Service 188, 200
 Self-Service-Cluster 71
 SELinux 294
 Semantic Versioning 176
 Separate Way 241, 244
 Separation of Duties 291
 Serverless-Architektur 144, 190, 195
 Serverless Computing 142, 191
 Serverless-Effekt 192
 Serverseitiges Tracing 213
 Service 101, 113, 126, 188
 Service-Account 289, 290
 Service-API 129
 Service Computing 12
 Service-Discovery 126

- Service-Interaktion 207
 - Servicekohäsion 188
 - Service-Merkmale 13
 - Service-Mesh 178, 189, 216
 - Service-Mesh Interface (SMI) 218
 - Service-Modell 12
 - IaaS 15
 - PaaS 15
 - SaaS 16
 - Service-of-Services 40, 161
 - Service Ownership 18, 164, 165
 - Sharding 184
 - Shared-Database-Pattern 167
 - Shared-Kernel 242
 - Shared Nothing 94
 - Sicherheitsgruppe 268, 277
 - Sicherheitsrichtlinie 298
 - Sidecar 216, 224
 - Single-Responsibility-Prinzip 164, 229
 - Single Source of Truth 185, 250, 254
 - Skalierbarkeit 18, 40, 143
 - Skalierung 125
 - horizontal 180
 - vertikal 180
 - Skalierungserfordernis 196
 - Social Engineering 263, 266
 - SOC (Service Organization Control) 330
 - Software as a Service (SaaS) 14, 16
 - Software Composition Analysis (SCA) 311
 - Software Supply Chain 280, 310
 - Software-Virtualisierung 66
 - Span 208, 209
 - Span-Kontext 208, 214
 - Speicherdauer 336
 - Spoofing-Angriff 224
 - Spread 102
 - SSH 271
 - SSL/TLS-Terminierung 281
 - Stabilitätsmuster 178
 - Staging 56
 - Standardisierung des Betriebs 18
 - Standardvertragsklauseln (SCC) 341, 342, 343, 345
 - Start-up Probe 131
 - Stateful-Service 183, 197
 - Stateful-Set 114, 118
 - Stateless 87, 170
 - Static Application Security Testing (SAST) 314
 - Storage Class 113, 132
 - Strategisches Design 231, 232
 - Strict Consistency 249
 - Stub 168
 - Subdomain 232
 - Subdomäne 232, 240
 - Supporting Subdomain 234, 246, 247, 251, 252
 - Swarm 34, 102, 105, 110, 114
 - Synonymer Begriff 238
 - Systementwurf 236
 - System Hardening 267, 268
- T**
- Taktisches Design 246
 - Telemetriedaten 32, 35, 40, 199
 - Konsolidierung 200
 - Telemetriedaten Konsolidierung 268
 - Terraform 74
 - Ausführungsplan 74
 - Data Source 75
 - Provider 75
 - Provisioner 76
 - Ressource 76
 - Ressourcengraph 74
 - Ressourcen-Scheduler 75
 - Testing 56
 - Test Phase 49
 - Threat Detection 309, 310
 - Threat Model 280
 - Timeout 189, 221
 - Time-to-Market 80, 195
 - TLS-Endpunkt-Termination 223
 - Token 290
 - Topologieschlüssel 124
 - Trace 207, 208
 - Tracing 40, 189, 199, 207
 - Tracing Backend 212
 - Tracing-Instrumentierung 212
 - Traffic Definition 218
 - Traffic-Management 217, 218
 - Traffic Policy 217
 - Traffic Spec 218
 - Traffic-Split 219
 - Traffic Telemetry 217
 - Transaktion 210, 247
 - Transparenz 336
 - Trigger 145, 148
 - Trigger - Regel - Aktion 148
 - Trunk 59
 - Trunk-basierte Entwicklung 59
 - Typ-1-Virtualisierung 65
 - Typ-2-Virtualisierung 66, 73

U

Ubiquitous Language 231, 236, 237, 239, 244
 Übungen (Labs)
 - Autoskalierung 139
 - Beobachtbarkeit 228
 - Container-Image Builds 97
 - Container-Image Builds durch Deployment-Pipelines 97
 - Container-Image Shrinking 97
 - Containerisierung 97
 - Deployment-Pipeline 61, 139
 - Docker 97
 - FaaS-Programmiermodell 197
 - GitLab CI/CD 61
 - Google Cloud Functions 156
 - Google Compute Engine 78
 - gRPC 197
 - IaC-basierte Provisionierung 78
 - Kubeless 156
 - Kubernetes 139
 - Logging 228
 - Log-Konsolidierung 228
 - Observability 228
 - OpenWhisk 156
 - Orchestrierung 139
 - Publish/Subscribe 197
 - Queuing 197
 - Representational State Transfer (REST) 197
 - Self-Healing 139
 - Service-Meshs und Traffic-Management 228
 - Service-Meshs und Verkehrstopologien 228
 - Software-defined Infrastructure 78
 - Swarm 139
 - Terraform 78
 - Tracing 228
 - Vagrant 78
 - Workload (interaktives Jupyter Notebook) 43
 Umgebungsvariable 54
 Unabhängige Aktualisierbarkeit 163, 229
 Unabhängige Austauschbarkeit 229
 Unattended Upgrade 270
 Uniform Resource Identifier (URI) 170
 Union Filesystem 86
 - Copy-on-Write 86
 - Layer 86
 - Namensraum 86
 Unterstützende Subdomäne 234
 Upstream-Service 162
 US CLOUD Act 1

V

Vagrant 72
 - Box 72
 - Provider 73
 - Provisioner 73
 - Vagrantfile 72
 Value Object 248
 vCPU 65, 122
 Vendor Lock-in 14, 82
 Verarbeitung nach Treu und Glauben 336
 Verfügbarkeit 18
 Verfügbarkeitszone 63
 Verhaltensanalyse 251
 Verkehrsfluss 136
 Verkehrstopologie 226
 Verschlüsselung 223, 301, 302
 Versionierungsschema 176
 Versionsverwaltungssysteme 29
 Vertikale Skalierung 180
 Vertraulichkeit 337
 Verzeichnis von Verarbeitungstätigkeiten 337
 Virtualisierung 24, 65
 - Betriebssystem 82
 - Hardware 65
 Virtual Private Network (VPN) 224
 Virtual Service 221
 Virtuelle Netzwerkschnittstelle 65
 VLAN 65
 Voll-Virtualisierung 66
 Volume Hardening 300
 Volume Provisioner 132
 Volunteer Computing 191
 Vorwärtskompatibilität 175

W

Wegwerf-Komponente 95
 Wegwerf-Umgebung 71
 Wertschöpfungskette 30
 Whitebox-Instrumentierung 202
 Whitebox-Monitoring 205
 Whitebox-Tracing 208
 Whitebox-Überwachung 202
 Widerstandsfähigkeit 174
 Wiederholung (Retry) 222
 Workload 19, 100
 - einmalig/selten 20
 - Heterogenität 101
 - Isolation 133
 - kontinuierlich sinkend 20
 - kontinuierlich steigend 20

- periodisch 20
- statisch 20
- zufällig 20

Workload-Allokation 100, 107

Workload-Ausführung 101

Workload Policing 303

Workload-Queue 104

Workload-Scheduler 104

wsk 147

X

X.509 224

X-Trace 208

Y

YAML, Notation 6

YARN 104, 105

You build it, you run it 33, 229

Z

Zeitbasierte Trigger 148

Zeitreihe 204

Zeitreihen-Datenbank 200, 205

Zero-Day Exploit 296

Zero-Trust Networking 223

Zertifikat-Handling 223, 282

Zertifizierung 322

Zertifizierungsstelle (CA) 224

Zone 63

Zugriffskontrolle 218

Zustandsanalyse 251

Zustandslosigkeit 143, 170

Zuteilungsdauer 22

Zweckbindung 336, 338

Zwei-Wege-Authentifizierung 224

Zwölf-Faktoren-Methodik 90

Zwölf-Faktoren-Modell 39