

HANSER



Leseprobe

zu

Cloud-native Computing

von Nane Kratzke

Print-ISBN: 978-3-446-46228-1

E-Book-ISBN: 978-3-446-47284-6

epub-ISBN: 978-3-446-47285-3

Weitere Informationen und Bestellungen unter

<https://www.hanser-kundencenter.de/fachbuch/artikel/9783446462281>

sowie im Buchhandel

© Carl Hanser Verlag, München

Inhalt

Vorwort	XI
----------------------	-----------

1 Einleitung	1
1.1 An wen sich dieses Buch richtet	2
1.2 Was dieses Buch behandelt	3
1.3 Sprachliche Konventionen	4
1.4 Notationskonventionen	5
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	16
2.2.1 Eignung von unterschiedlichen Arten von Workloads	17
2.2.2 Effekt von Zuteilungsdauer und Ressourcengröße	19
2.3 Entwicklung der letzten Jahre	21
3 DevOps	23
3.1 Prinzipien des Flow	25
3.1.1 Prinzip 1: Arbeit sichtbar machen	25
3.1.2 Prinzip 2: Work in Progress beschränken	26
3.1.3 Prinzip 3: Flaschenhalse minimieren	26
3.2 Prinzipien des Feedbacks	27
3.2.1 Prinzip 4: Probleme früh erkennen	27
3.2.2 Prinzip 5: Probleme sofort lösen	28
3.2.3 Prinzip 6: Probleme professionell verantworten	28
3.3 DevOps-geeignete Architekturen	29
3.3.1 Randbedingungen für die Entwicklung	29

3.3.2	Nutzung von Orchestrierungsplattformen	30
3.3.3	Randbedingungen im Betrieb	30
4	Cloud-native	33
4.1	Definitionen in Industrie und Forschung	34
4.2	Die Cloud-native-Definition dieses Buchs	35
4.3	Zusammenfassung und Ausblick auf Teil II und Teil III	37
Teil II: Everything as Code		39
5	Einleitung zu Teil II	41
6	Deployment-Pipelines	43
6.1	Deployment-Pipelines as Code	44
6.1.1	Phasen-Pipelines	45
6.1.2	Gerichtete Pipelines	46
6.1.3	Hierarchische Pipelines	47
6.1.4	Steuerung von Pipelines	48
6.2	DevOps-geeignete Branching-Strategien	50
6.2.1	Git-Flow	51
6.2.2	GitHub-Flow	52
6.2.3	Trunk-basierte Entwicklung	53
6.3	Zusammenfassung	54
7	Infrastructure as Code	57
7.1	Virtualisierung	59
7.1.1	Virtualisierung von Hardware-Infrastruktur	59
7.1.2	Virtualisierung von Software-Infrastruktur	60
7.2	Provisionierung	62
7.2.1	Immutable Infrastructure	62
7.2.2	IaC-Ansätze	63
7.2.3	Provisionierung von lokalen Umgebungen	66
7.2.4	Provisionierung von Multi-Host-Umgebungen	68
7.3	Zusammenfassung	71
8	Standardisierung von Deployment Units (Container)	73
8.1	Hintergrund (PaaS)	73
8.2	Betriebssystem-Virtualisierung	76
8.3	Container Runtime Environments	77
8.3.1	Kernel-Namespaces	78
8.3.2	Process Capabilities	79
8.3.3	Control Groups	80

8.3.4	Union Filesystem	80
8.3.5	High-Level- und Low-Level-Container-Laufzeitumgebungen	81
8.4	Bau und Bereitstellung von Container-Images	82
8.5	Faktoren gut betreibbarer Container	84
8.5.1	Codebase	85
8.5.2	Abhängigkeiten und Konfigurationen	85
8.5.3	Unterstützende Services und Port Binding	86
8.5.4	Build-, Release- und Run-Phase	87
8.5.5	Horizontale Skalierung über Prozesse	88
8.5.6	Umgebungen, Logs und Betrieb	89
8.6	Zusammenfassung	90
9	Container-Plattformen	93
9.1	Scheduling	94
9.1.1	Heterogenität von Workloads	95
9.1.2	Scheduling-Algorithmen	96
9.1.2.1	Einfache Scheduling-Algorithmen	96
9.1.2.2	Multidimensionale Scheduling-Algorithmen	97
9.1.2.3	Kapazitätsbasierte Scheduling-Algorithmen	97
9.1.3	Scheduling-Architekturen	98
9.1.3.1	Monolithischer Scheduler	99
9.1.3.2	2-Level-Scheduler	99
9.1.3.3	Shared-State Scheduler	100
9.2	Orchestrierung	101
9.2.1	Definition von Betriebszuständen	101
9.2.2	Regelkreis: Desired versus Current State	102
9.3	Inside Kubernetes	103
9.3.1	Kubernetes-Architektur	104
9.3.2	Verwaltete Ressourcen und Basis-Blueprint	106
9.3.3	Schedulbare Workloads	108
9.3.3.1	Deployments	108
9.3.3.2	(Cron-)Jobs	110
9.3.3.3	Daemon-Sets	111
9.3.3.4	Stateful-Sets	112
9.3.4	Scheduling Constraints	115
9.3.4.1	Angabe des Ressourcenbedarfs mittels Requests und Limits	115
9.3.4.2	Knoten-Selektoren	116
9.3.4.3	Knotenaffinitäten	117
9.3.4.4	Pod-(Anti-)Affinitäten	118
9.3.5	Automatische Skalierung von Workloads	119
9.3.6	Exponieren von Workloads als interne und externe Services	120
9.3.7	Health Checking	123
9.3.8	Persistenz	126

9.3.9	Isolation von Workloads.....	127
9.3.9.1	Namespaces und Role-based Access Model (Multi-Tenancy).....	127
9.3.9.2	Quotas und Limit Ranges.....	128
9.3.9.3	Network Polycys.....	129
9.4	Zusammenfassung.....	131
10	Function as a Service.....	135
10.1	FaaS-Plattformen.....	137
10.1.1	Das FaaS-Programmiermodell.....	139
10.1.2	Zu berücksichtigende Randbedingungen.....	140
10.1.3	Veranschaulichung des FaaS-Programmiermodells.....	141
10.2	Plattformagnostische FaaS-Frameworks.....	142
10.3	Ereignisbasierte Autoskalierung.....	145
10.4	Zusammenfassung.....	148

Teil III: Cloud-native Architekturen 151

11	Einleitung zu Teil III.....	153
12	Microservice und Serverless-Architekturen.....	155
12.1	Eigenschaften von Microservices.....	156
12.2	Integrationsmuster für Microservices.....	160
12.2.1	Datenbankbasierte Integration.....	161
12.2.2	(g)RPC-basierte Interprozesskommunikation.....	161
12.2.3	Representational State Transfer (REST).....	164
12.2.4	Ereignisbasierte Integration (asynchron).....	167
12.2.5	API-Versioning.....	169
12.3	Architekturelle Sicherheit.....	172
12.3.1	Circuit-Breaker.....	172
12.3.2	Bulkhead.....	173
12.3.3	Idempotente API-Operationen.....	174
12.4	Skalierung von Microservices.....	174
12.4.1	Load Balancing.....	175
12.4.2	Messaging.....	175
12.4.3	Skalierung zustandsbehafteter Komponenten.....	177
12.4.3.1	Scaling for Reads.....	178
12.4.3.2	Scaling for Writes (Sharding).....	178
12.4.3.3	Command Query Responsibility Segregation (CQRS).....	179
12.4.4	Caching.....	180
12.5	Prinzipien zur Entwicklung von Microservices.....	181
12.5.1	Prinzip 1: Bilde Modelle um Geschäftskonzepte.....	181
12.5.2	Prinzip 2: Erschaffe eine Kultur der Automatisierung.....	181

12.5.3	Prinzip 3: Blende interne Implementierungsdetails aus	182
12.5.4	Prinzip 4: Dezentralisiere	182
12.5.5	Prinzip 5: Definiere unabhängig aktualisierbare Einheiten	182
12.5.6	Prinzip 6: Isoliere Fehler	183
12.5.7	Prinzip 7: Baue gut beobachtbare Services	183
12.6	Serverless-Architekturen	184
12.6.1	Architekturelle Konsequenzen von Serverless-Limitierungen	185
12.6.2	Das API-Gateway-Pattern	187
12.6.3	Abgrenzung zu Microservices	189
12.7	Zusammenfassung	190
13	Beobachtbare Architekturen.	193
13.1	Konsolidierung von Telemetriedaten	194
13.2	Instrumentierung von Systemen.	196
13.2.1	Logging	196
13.2.2	Monitoring.	198
13.2.2.1	Metrikarten	200
13.2.2.2	Empfehlungen für die Metrikinstrumentierung	201
13.2.3	Tracing.	201
13.2.3.1	Empfehlungen für die Instrumentierung	203
13.2.3.2	Tracing-Instrumentierung und Erzeugung von Spans	206
13.2.3.3	Serverseitiges Tracing und Extraktion von Span-Kontexten	207
13.2.3.4	Clientseitiges Tracing und Weiterreichen von Span-Kontexten. .	208
13.3	Automatisierte Instrumentierung.	209
13.3.1	Eigenschaften von Service-Meshs.	210
13.3.2	Traffic-Management.	212
13.3.3	Resilienz	215
13.3.4	Sicherheit	217
13.3.5	Management und Analyse von Verkehrstopologien	220
13.4	Zusammenfassung	221
14	Domain-driven Design	223
14.1	Fachlichkeit	224
14.2	Strategisches Design.	226
14.2.1	Subdomänen	227
14.2.1.1	Kerndomäne (Core Subdomain)	227
14.2.1.2	Unterstützende Subdomäne (Supporting Subdomain)	228
14.2.1.3	Generische Subdomänen (Generic Subdomain)	228
14.2.1.4	Anmerkungen am Beispiel einer Fallstudie.	228
14.2.2	Ubiquitous Language	230
14.2.2.1	Eine gemeinsame Sprache als Schlüssel zu einem gemeinsamen Verständnis	231
14.2.2.2	Mehrdeutige und synonyme Begriffe	232

14.2.3 Bounded Contexts	233
14.2.4 Context Mapping	235
14.2.4.1 Partnerschaftliche Kooperationsmuster (Partners und Shared-Kernel)	235
14.2.4.2 Customer-Supplier-Kooperation	237
14.2.4.3 Separate Ways	238
14.2.4.4 Context Maps als Landkarte von Machtverhältnissen	239
14.3 Taktisches Design	240
14.3.1 Oft genutzte Pattern für Geschäftslogik	240
14.3.1.1 Das ETL-Pattern (primär Supporting Subdomains)	240
14.3.1.2 Das Active Record-Pattern (primär Supporting Subdomains) ...	241
14.3.1.3 Das Domain Model-Pattern (primär Core Subdomains)	242
14.3.1.4 Das Event-Sourcing-Pattern (primär Core Subdomains)	244
14.3.2 Oft genutzte Pattern für die Architektur	245
14.3.2.1 Die Ebenen-Architektur	246
14.3.2.2 Das Ports & Adapter-Pattern	247
14.3.2.3 Das CQRS-Pattern	247
14.4 Zusammenfassung	250
 15 Schlussbemerkungen	 253
 Literaturverzeichnis	 261
 Stichwortverzeichnis	 265

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.

Lübeck, im Oktober 2021

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.

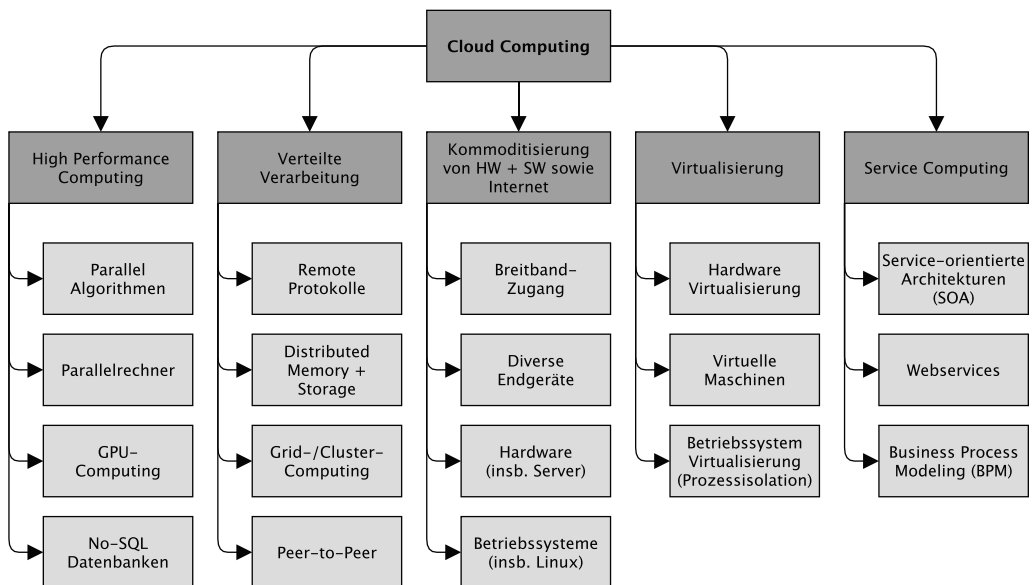


Bild 2.1 Einflussfaktoren auf das Cloud Computing

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 substanziell 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 substanziellen 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.

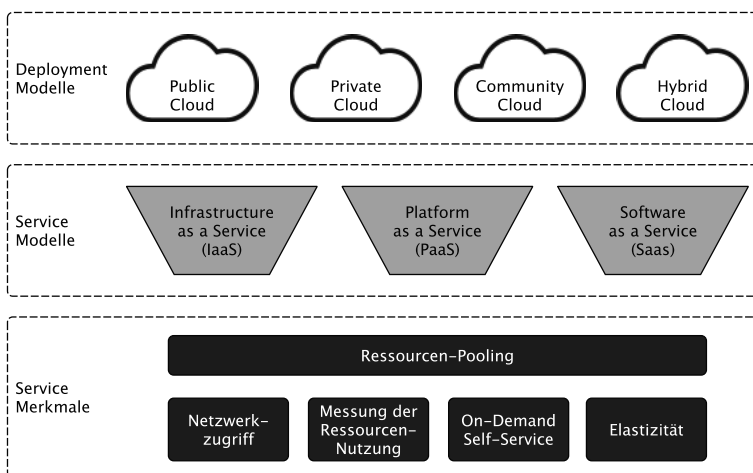


Bild 2.2 NIST-Modell des Cloud Computings

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 realloziert. 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.

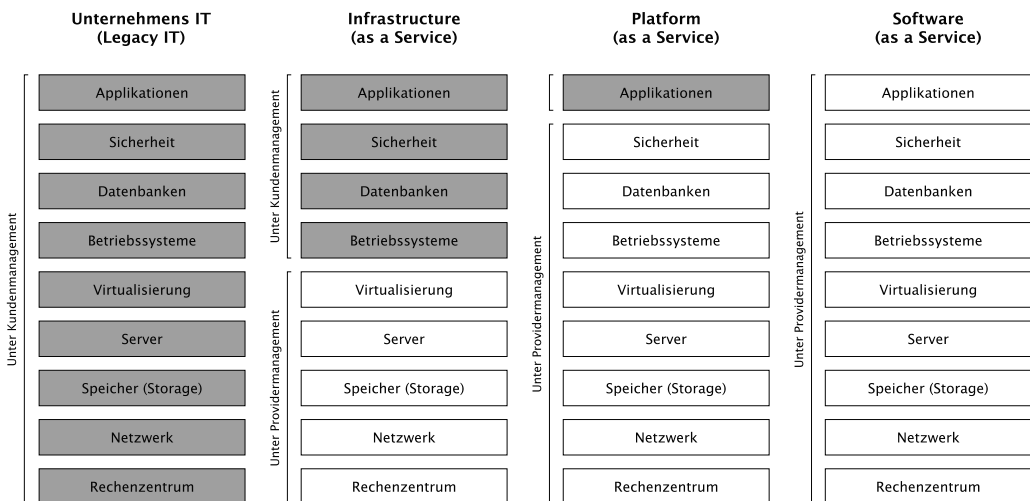


Bild 2.3 Auslagerung der Wertschöpfung bei IaaS, PaaS und SaaS

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).

■ 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.

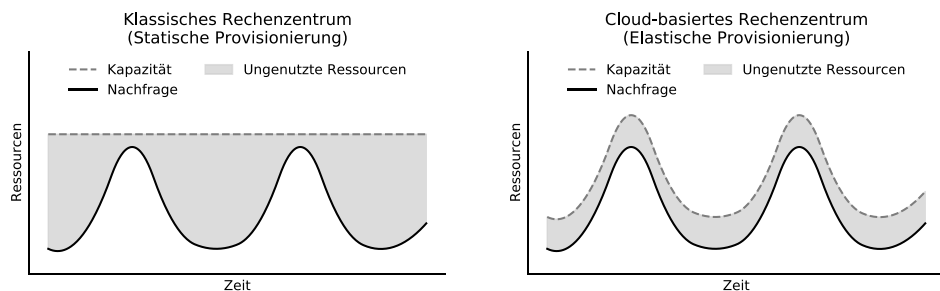


Bild 2.4 Statische und elastische Provisionierung von Ressourcen

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.

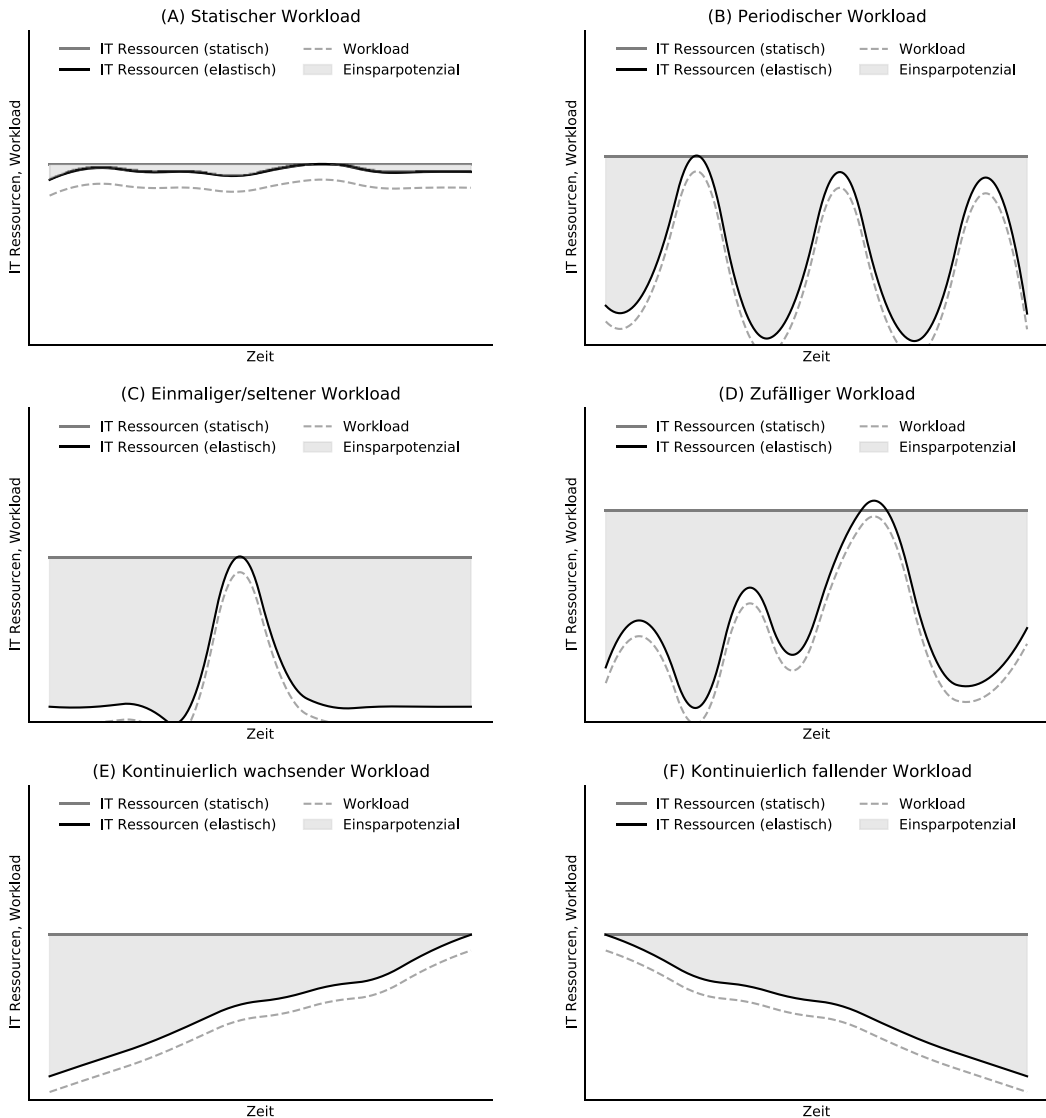


Bild 2.5 Zu berücksichtigende Workloads im Cloud Computing

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 persistenten 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 grundsätzlich folgende rein ökonomische Eignungsreihenfolge von Workloads für das 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.

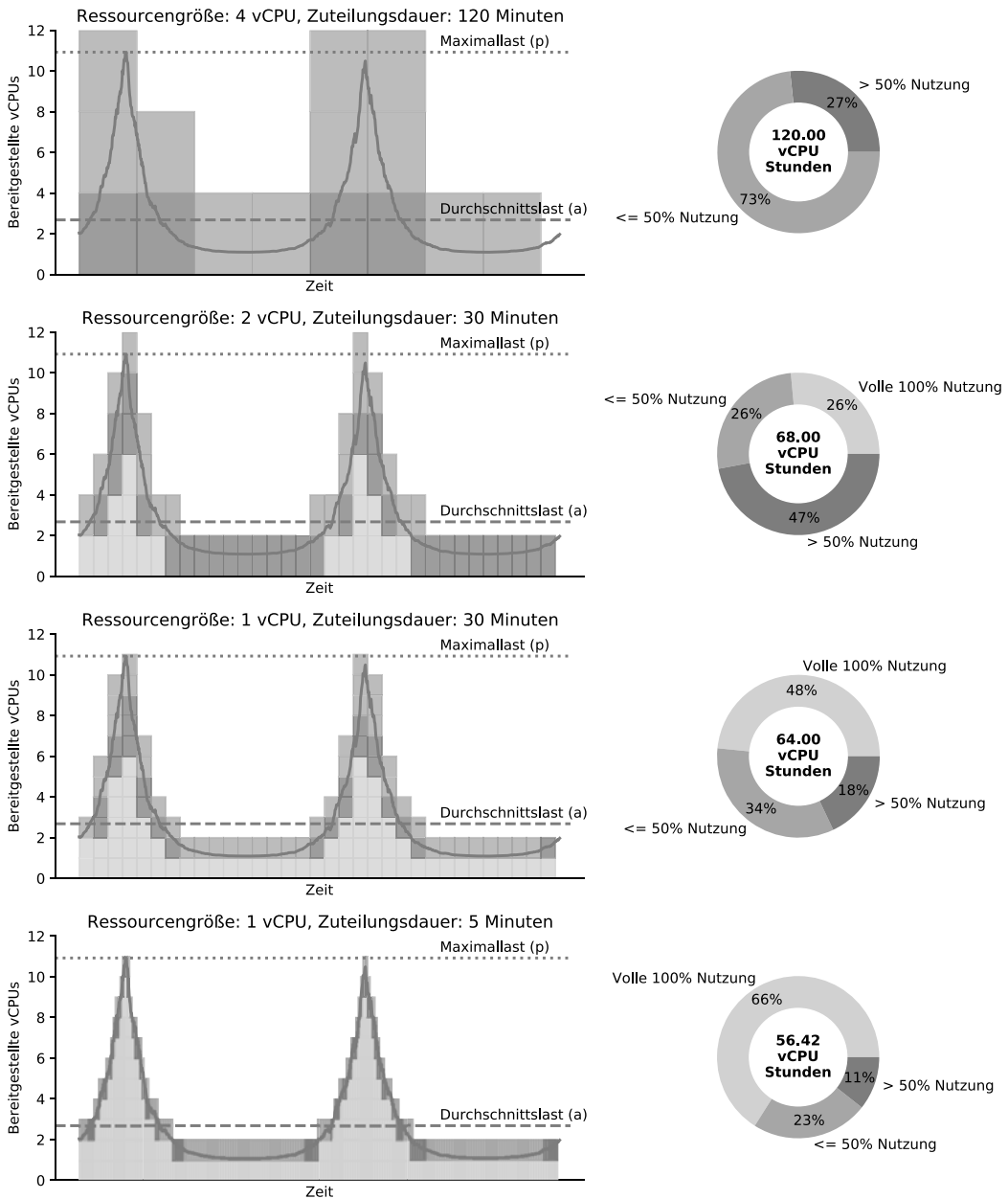


Bild 2.6 Effekt von Ressourcengröße und Zuteilungsdauer

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 durch-

aus 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.

■ 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-Infra-

strukturen 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 21, 78, 136
3-Tier-Architektur 246
12-Faktoren
– Abhängigkeiten 85
– Administrative Prozesse (update, backup, restore) 90
– Build, Release, Run 87
– Codebase 85, 87
– Environment 89
– Horizontale Skalierung 88
– Konfigurationen 85
– Logging 89
– Port Binding 86
– Skalierung über Prozesse 88
– Umgebung 89
– Unterstützende Services 86
12-Faktoren-Methodik 107, 137, 198

A

Ablaufverfolgung 193
A/B-Tests 31
A/B-Testszzenarien 214, 215
Abwärtskompatibilität 169
ACID 177
Active Record 241
Active Record-Pattern 241, 246
Affinität 117
Affinity 116
Aggregat 242, 243, 244
Aggregate Root 243
Aggregatgrenze 243
Aggregatwurzel 243
Alert-Manager 194, 199
ALLOW-Regel 219
Analysemodell 230
Anforderungen 230
Anti-Corruption-Layer 237
Anwendungsschicht 247

Anwendungsvirtualisierung 60, 61
API 169
API-Gateway 171, 187, 188
API-Versioning 161, 169
APM 195
Append-Only-Log 244
Architektur 35
– DevOps-geeignet 29
– Serverless 137
Architekturelle Sicherheit 172
Architekturmuster 245
Asynchrone Architektur 167
Auditierbarkeit 245
Auditing 217
Audit-Protokolle 245
Authentication Policy 218
Authentifizierung 217
– Peer 219
– Request 219
Authorisation 217
Authorisation Policy 218, 219
Automatisierte Instrumentierung 209
Automatisierung 181
Autoskalierung 119
– ereignisbasiert 145
– horizontal, Pod 119
AWS Lambda 144
Azure Lambda 144

B

Backend as a Service (BaaS) 185
BASE 177
Batch-Job 95
Batch-System 201
Beobachtbare Architekturen 193
Beobachtbarkeit 35, 183
Best Practices 140, 254
Betriebssystem-Virtualisierung 60
Betriebszustand 101

- Big Five 1
- Binpack 96
- Blackbox-Monitoring 199
- Black-Box-Tracing 202
- Blackbox-Überwachung 196
- Block-Storage 59
- Blue/Green Deployment 29
- Blue/Green-Release 183
- Blueprint 101, 106
- Borg 99
- Bounded Context 181, 225, 233, 234, 246, 248
- Branching-Strategien 50
- Breaking-Change 160, 170, 182
- Build Phase 43
- Bulkhead 173

C

- CaaS 76
- Caching 88, 164, 180
 - clientseitig 180
 - Proxy-Caching 181
 - serverseitig 180
- Canary 213, 215
- Canary-Release 29, 183
- CAP-Theorem 177
- Chaos Engineering 28
- Checkpoint 249
- Chef 64
- Choreography-over-Orchestration 182
- CI/CD 43
- Circuit-Breaker 172, 183, 216
- C-Level-Funktion 3
- Clientseitiges Tracing 208
- Client-Server 162, 164
- Cloud Computing 11
 - NIST-Definition 11
- Cloud-native 2, 22, 33, 254
 - Definition 35
- Cloud-native Computing Foundation (CNCF) 34, 103
- Cloud-Ökonomie 16, 136
- Cluster 94, 101, 116
- Cluster-Awareness 94
- Cluster-Scheduler 104
- CNCF 103
- CNI 104
- Code Repository 43
- Command 248
- Command Execution-Modell 248

- Command Query Responsibility Segregation (CQRS) 179, 247
- Community Cloud 13
- Config Map 107
- Constraint 116
- Container 22, 35, 36, 60, 73, 77, 103, 104, 182
 - Laufzeitumgebung 78
 - Runtime 78
- Container as a Service 76
- Container-Image 82
- Container Network Interface (CNI) 104
- Container Runtime Environment 82, 106
- Container Storage Interface (CSI) 104
- Content-Delivery-Netzwerks (CDN) 181
- Context Mapping 225, 235
- Continuous Deployment 43
- Continuous Integration 43
- Controller 102
- Control Plane 211
- Conway's Law 159, 223, 235
- Copy-on-Write 80
- Core Subdomain 227, 242, 244, 245, 247, 248
- CQRS 179, 247
- Creative Commons-Lizenz (CC0) 7
- Cron-Job 110
- CRUD 166, 241, 247
- CSI 104
- Current State 102, 119
- Customer-Supplier 235, 237

D

- Daemon-Set 108, 111
- DAG Pipeline 46
- Dapper 202
- Data Plane 211
- Datenbankbasierte Integration 161
- Datenbasierte Integration 161
- Datenkopplung 161, 182
- Defense in Depth 217
- Dekomposition 155, 223
- DENY-Regel 219
- Dependency Injection 247
- Deployment 107, 108, 120
- Deployment-Pipeline 27, 29, 43, 83
 - Job 44
 - Phase 43
 - Trigger 44
- Deployment Unit 35, 36, 60, 73, 159
- Deploy Phase 43
- Desired State 102, 119

- Development 50
- Development-Branch 52
- DevOps 23, 84, 159, 188, 194
 - Flaschenhalse 26
 - Kultur 26
 - Prinzipien des Feedbacks 27, 36
 - Prinzipien des Flow 25, 36
 - Work in Progress 26
 - Zyklus 24, 30
- Docker 77
- Dockerfile 82
- Domain-driven 181
- Domain-driven Design 223
- Domain-Event 226, 243, 244
- Domain Model-Pattern 242
- Domänenmodell 223, 226, 242, 244
- Domänenwissen 230
- Dominant Resource Fairness 97
- Double-Spending-Problem 138, 185, 187
- Downstream-Service 156
- Dumb-Middleware-with-Smart-Endpoints 182

E

- Ebenen-Architektur 246
- Effektives Design 224
- ElasticSearch 195
- Elastisches System 168
- Elastizität 36, 137, 167
- Emulation 60
- Endbenutzer-Choreografie 187
- End-to-End-Tracing 206
- Enterprise-Architektur-Management (EAM) 233
- Entkopplung 59
- Environment 48, 50
- Ereignis-basiert 182
- Ereignisbasierte Integration 161, 167
- Ereignisbasierte Systeme 167
- Ereignisgesteuert 167
- Ereignisquelle 139
- ETL-Pattern 240
- Event-driven 139
- Event-Emitting-Service 167
- Event-Sourcing 248
- Event-Sourcing-Pattern 244
- Event Store 244
- Eventual Consistency 178, 243
- Everything as Code, Deployment Pipeline 44
- Evolutionäres Design 158, 223
- Execution-Monitor 98

- Executor 99
- Exporter 198
- Extract-Transform-Load (ETL) 240
- Extraktion von Span-Kontexten 207

F

- FaaS 120
 - Best Practices 140
- FaaS-Framework 142
- FaaS-Plattform 136, 184
- FaaS-Programmiermodell 139, 184
- Fachlichkeit 223, 224, 242
- Fail early 216
- Fairness 96
- Fallacies of Distributed Computing 249
- Feature-Branch 52
- Feature Release 170
- Feature-Schalter 29
- Fehlertoleranz 96
- File-Storage 59
- Fluentd 195
- Function 194
- Function as a Service (FaaS) 21, 135
- Funktion 139

G

- GAE 74
- GAIA-X 1
- Gegenseitige Authentifizierung 218
- Gegenseitige TLS-Authentifizierung (mTLS) 218
- Generic Subdomain 228, 238
- Generische Subdomäne 228
- Gerichtete Pipeline 46
- Geschäftskonzept 181
- Geschäftslogik 240, 241, 244, 246, 247
- Gesetz von Conway 159
- Git-Flow 51
- GitHub-Flow 52
- GitLab CI/CD 44
- Google App Engine 74
- Google Cloud Functions 141, 144
- Grafana 195
- gRPC (gRPC Remote Procedure Call) 123, 162, 204

H

- Hadoop 98
- HashiCorp Configuration Language 69
- HATEOAS 165
- HCL 69

Health Checking 123
 Heroku 75
 Hexagonale Architektur 247
 Hierarchische Pipeline 47
 High-Level Container Runtime 81
 High-Level-Design 239
 Horizontale Pod-Autoskalierung 119
 Horizontale Skalierung 174
 Horizontal Pod Autoscaler 119
 Horizontal Pod Autoscaling (HPA) 145
 HPA 119
 HTTP-/REST-basierte Integration 161
 Hybrid Cloud 14
 Hypermedia as the Engine of Application State (HATEOAS) 165
 Hyperthread 116
 Hypervisor 59

I

IaC 63
 IDEAL-Modell 34
 Idempotente Operation 174
 Idempotenz 174
 Immutable 242
 Immutable Infrastructure 62
 Implementierungsdetail 160, 182
 Infrastructure as a Service (IaaS) 14, 15
 Infrastructure as Code 57
 Infrastruktur
 – als Code 63
 – elastisch 15
 Infrastrukturkomponente 247
 Infrastrukturschicht 247
 Ingress 107, 122
 In-Process-Komponenten 155
 Instrumentierung 196
 Instrumentierungsbibliothek 203
 Instrumenting Library 194
 Integrations-Branch 52
 Interprozesskommunikation 161
 Isolation 59, 61
 Isolationsmechanismus 82
 Istio 211, 215, 218, 220

J

Jaeger 195
 Job 108, 110, 194, 200, 201

K

Kanban 25

KEDA 145
 – ScaledJob 146
 – ScaledObject 146
 Kerndomäne 227
 Kiali 220
 Kibana 195
 Knotenaffinität 117
 Kohäsion 223
 Kommunikationsmuster 239
 Konfigurations-API-Server 218
 Konfigurationsmanagement 64
 Konformist-Pattern 237
 Kontrollgruppe 214
 Kritischer Pfad 203
 Kubeless 141, 144
 Kubernetes 30, 99, 103, 194, 212
 – Affinität 117
 – API-Server 105, 106
 – Architektur 105
 – Cloud-Manager 105
 – Cluster Role 127
 – Controller-Manager 105
 – Daemon-Set 111
 – Deployment 109
 – Horizontal Pod Autoscaler (HPA) 119
 – Ingress 122, 162, 189
 – Job 110
 – Kubelet 106
 – Kube-Proxy 106
 – Limit 115, 128
 – Master Node 105
 – Namespace 127
 – Network-Plug-in 105
 – Network Policy 129, 219
 – Persistent Volume Claim (PVC) 126
 – Persistent Volume (PV) 126
 – Quota 129
 – RBAC 127
 – Request 115
 – Resource Quota 128
 – Role 127
 – Role Binding 127
 – Scheduler 105
 – Secret 127
 – Selektor 116
 – Service 122, 175
 – Service Account 127
 – Stateful-Set 113
 – Storage-Plug-in 105
 – Worker Node 106

- Workload 108
- Kubernetes-Ressourcen 106

L

- Lambda 141
- Lastausgleich 175
- Laufzeitumgebung 61
- Layered Architecture 246
- Ledger 244
- Limits 115
- Liveness Probe 123
- Load Balancer 121
- Load Balancing 175, 178
- Local Procedure Call (LPC) 162
- Log-Aggregation 196
- Logge auf stdout 198
- Logging 36, 183, 193, 196
- Logikebene 246
- Log-Level 197
 - Debug 197
 - Error 197
 - Fatal 197
 - Info 197
 - Trace 197
 - Warning 197
- Lokalität 96
- Lose Kopplung 158, 223
- Low-Level Container Runtime 81

M

- Machtgefälle 235
- Machtverhältnis 239
- Manifest 104, 106
- Man-in-the-Middle-Angriff 217, 218
- Marathon 108
- Materialien 7, 38, 72, 91, 133, 251
- Materialien (Slides, Handouts)
 - 12-Faktoren-Methodik 91
 - Architektur-Pattern für Core Subdomains (DDD) 251
 - Architektur-Pattern für Supporting Subdomains (DDD) 251
 - Beobachtbarkeit 222
 - Betriebssystemvirtualisierung 91
 - Cloud Computing Historie 38
 - Cloud-native Systeme 38
 - Cloud-Ökonomie 38
 - Container-Orchestrierung 133
 - Context Mapping (DDD) 251
 - Deployment Pipelines 55

- Deployment Units (Container) 91
- DevOps 38, 55
- DevOps-geeignete Architekturen 55
- Docker 91
- Domain-driven Design 251
- Effektives Software-Design 251
- FaaS-Plattformen 149
- FaaS-Programmiermodell 149
- Function as a Service (FaaS) 149, 191
- Immutable Architectures 72
- Infrastructure as a Service 72
- Infrastructure as Code 72
- Kubernetes 133
- Kubernetes Blueprints (Manifests) 133
- Logging 222
- Materialien (Slides, Handouts) 91
- Metriken und Monitoring 222
- Microservices 191
- Pattern für Geschäftslogiken (DDD) 251
- Platform as a Service (PaaS) 133
- Prinzipien des Feedbacks 55
- Prinzipien des Flow 55
- Resilienz 222
- Serverless Computing 149, 191
- Service-Meshes 222
- Sheduling 133
- Sicherheit 222
- Strategisches Design (DDD) 251
- Subdomains (DDD) 251
- Taktisches Design (DDD) 251
- Telemetriedaten 222
- Terraform 72
- Tracing 222
- Traffic-Management 222
- Ubiquitous Language (DDD) 251
- Vagrant 72
- Visualisierung von Verkehrstopologien 222
- Was ist Cloud Computing? 38
- Mehrdeutiger Begriff 232
- Memory-Ballooning 59
- Mentales Modell 233
- Mesos 30, 97, 100, 104, 108
- Messaging 175
- Metriken 35, 183, 193, 198
 - Messung (Gauge) 200
 - Verteilung (Histogramm) 200
 - Zähler (Counter) 200
- Metrikinstrumentierung 201
- Microservice 27, 35, 156, 225
- Microservice-Architektur 138, 193, 223

Microservice-basierte Anwendung 157
 Millicore 116
 Monitoring 193, 198
 Monolithische Anwendung 157
 Monorepository 48
 mTLS 218, 219
 Multi-Cloud 66
 Multiplizität 59
 Multi-Tenancy 127, 129, 219
 Mutable 242
 Mutual Authentication 218

N

Nachrichtenorientiertes System 168
 Netzwerkpartition 183
 Nomad 30, 104
 NoSQL 179
 NoSQL-Datenbanken 177

O

Objektmodell 248
 – lesend 247
 – schreibend 247
 Objektrelationales Mapping (ORM) 241
 Observability 35, 183, 196
 Observable 168
 OCI 78, 103
 Omega 100
 One-Service-per-Container 183
 Online-System 201
 OpenAPI 238
 Open-Container-Initiative 78
 Open-Host-Service 238
 OpenTracing-API 203, 206
 OpenWhisk 141, 144
 Orchestrierung 93, 101
 Orchestrierungsplattform 30, 182
 Orchestrierungsregelkreis 103, 120
 Ortsunabhängigkeit 168
 Out-of-Process-Komponenten 155
 Output Stream 90
 Overlay Network 104
 Over-Provisioning 21

P

PaaS 73, 76
 Para-Virtualisierung 59
 Partnerschaftliche Kooperation 235
 Partnerschaftsmodell 235
 Partnership 235

Pattern 254
 Pay-as-you-go 2, 16
 Peak-to-Average 17
 Peer-to-Peer Computing 185
 Persistent Volume 126
 Persistent Volume Claim 107, 126
 Persistenzebene 246
 Phasen- 45
 Platform as a Service (PaaS) 14, 15, 73, 76
 Plattform
 – Container 76
 – elastisch 15, 36
 – Function as a Service (FaaS) 137
 – PaaS 74
 Pod 104, 145
 Pod-Affinität 118
 Policy Enforcement Point (PEP) 218
 Polyglotte Persistenz 248
 Polyglott Programming 61
 Ports & Adapter-Pattern 247
 Präsentationsebene 246
 Private Cloud 13
 Probe 123
 Production 50
 Produktivsystem 28
 Projektion
 – asynchron 249
 – synchron 248
 Projektions-Engine 249
 Prometheus 195
 Protocol Buffers 162
 Protokollierung 36, 196
 Provisionierung 62
 – deklarativ 64
 – imperativ 64
 – Pull-basiert 64
 Proxy 210, 218
 Prozessisolation
 – Control Group (cgroup) 80
 – Namensräume für Dateisysteme 80
 – Namespace 78
 – Priorisierung 80
 – Process Capabilities 79
 – Quota 80
 Public Cloud 13
 Publish/Subscribe 176, 244
 Puppet 64
 Push-Gateway 200, 201
 PVC 107
 Python 6

Q

Query 248
 Querying-System 199
 Queueing 175, 244

R

RAFT 113
 RBAC 127, 128, 129
 ReactiveX-Programmiermodell 168
 Readiness Probe 124
 Reaktive Erweiterung (Rx) 168
 Reaktives System 167
 Regel 139
 Regelkreis 102, 119
 Regelkreis-basierte Orchestrierung 103
 Region 57
 Release 29
 Releaserisiken 29
 Remote Procedure Call (RPC) 161, 171
 Replay-Angriff 218
 Replaying Time Machine 245
 Replicas 112
 Replica-Set 108
 Replication Controller 107
 Representational State Transfer (REST) 164
 Requests 115
 Resilient Software Design 28
 Resilienz 28, 167, 215
 Resilienz-Pattern 215
 Responsivität 167
 Ressourceneffizienz 22
 Ressourcengröße 19
 Ressourcenkontingent 129
 REST 35, 123, 164, 174, 182, 186, 190, 204
 REST-API 171
 Restart Policy 111
 Reverse-Proxy 181, 188
 Role-based Access Model (RBAC) 127
 Rolling-Updates 29
 RPC
 – Bidirectional-Streaming 163
 – Client-Streaming 163
 – Server-Streaming 163
 – Unary 163
 Runtime 61

S

Sandbox 75
 Scale-to-Zero 120, 135, 147, 156
 Scaling for Reads 178

Scaling for Writes 178
 Scaling out 174
 Scaling up 174
 Scheduler 94, 116
 – 2-Level 99
 – monolithisch 99
 – Shared-State 100
 Scheduling 93
 – Algorithmus 96
 – Architekturen 98
 – Constraints 115
 – einfache Algorithmen 96
 – kapazitätsbasierte Algorithmen 97
 – multidimensionale Algorithmen 97
 Secret 107
 Security by Default 217
 Selektor 116, 219
 Self-Healing 34, 103
 Self-Service 182, 194
 Self-Service-Cluster 65
 Semantic Versioning 170
 Separate Way 235, 238
 Serverless-Architektur 138, 184, 189
 Serverless Computing 136, 185
 Serverless-Effekt 186
 Serverseitiges Tracing 207
 Service 95, 107, 120, 182
 Service-API 123
 Service Computing 12
 Service-Discovery 120
 Service-Interaktion 201
 Servicekohäsion 182
 Service-Merkmale 13
 Service-Mesh 172, 183, 210
 Service-Mesh Interface (SMI) 212
 Service-Modell 12
 – IaaS 15
 – PaaS 15
 – SaaS 16
 Service-of-Services 36, 155
 Service Ownership 158, 159
 Sharding 178
 Shared-Database-Pattern 161
 Shared-Kernel 236
 Shared Nothing 88
 Sidecar 210, 218
 Single-Responsibility-Prinzip 158, 223
 Single Source of Truth 179, 244, 248
 Skalierbarkeit 36, 137
 Skalierung 119

- horizontal 174
- vertikal 174
- Skalierungserfordernis 190
- Software as a Service (SaaS) 14, 16
- Software-Virtualisierung 60
- Span 202, 203
- Span-Kontext 202, 208
- Spoofing-Angriff 218
- Spread 96
- Stabilitätsmuster 172
- Staging 50
- Start-up Probe 125
- Stateful-Service 177, 191
- Stateful-Set 108, 112
- Stateless 81, 164
- Storage Class 107, 126
- Strategisches Design 225, 226
- Strict Consistency 243
- Stub 162
- Subdomain 226
- Subdomäne 226, 234
- Supporting Subdomain 228, 240, 241, 245, 246
- Swarm 30, 96, 99, 104, 108
- Synonymer Begriff 232
- Systementwurf 230

T

- Taktisches Design 240
- Telemetriedaten 27, 30, 35, 193
 - Konsolidierung 194
- Terraform 68
 - Ausführungsplan 68
 - Data Source 69
 - Provider 69
 - Provisioner 70
 - Ressource 70
 - Ressourcengraph 68
 - Ressourcen-Scheduler 69
- Testing 50
- Test Phase 43
- Timeout 183, 215
- Time-to-Market 74, 189
- TLS-Endpunkt-Termination 217
- Topologieschlüssel 118
- Trace 201, 202
- Tracing 35, 183, 193, 201
- Tracing Backend 206
- Tracing-Instrumentierung 206
- Traffic Definition 212
- Traffic-Management 211, 212

- Traffic Policy 211
- Traffic Spec 212
- Traffic-Split 213
- Traffic Telemetry 211
- Transaktion 204, 241
- Trigger 139, 142
- Trunk 53
- Trunk-basierte Entwicklung 53
- Typ-1-Virtualisierung 59
- Typ-2-Virtualisierung 60, 67

U

- Ubiquitous Language 225, 230, 231, 233, 238
- Übungen (Labs)
 - Autoskalierung 133
 - Beobachtbarkeit 222
 - Container-Image Builds 91
 - Container-Image Builds durch Deployment Pipelines 91
 - Container-Image Shrinking 91
 - Containerisierung 91
 - Deployment Pipeline 55, 133
 - Docker 91
 - FaaS-Programmiermodell 191
 - GitLab CI/CD 55
 - Google Cloud Functions 149
 - Google Compute Engine 72
 - gRPC 191
 - IaC-basierte Provisionierung 72
 - Kubeless 149
 - Kubernetes 133
 - Logging 222
 - Log-Konsolidierung 222
 - Observability 222
 - OpenWhisk 149
 - Orchestrierung 133
 - Publish/Subscribe 191
 - Queuing 191
 - Representational State Transfer (REST) 191
 - Self-Healing 133
 - Service-Meshes und Traffic-Management 222
 - Service-Meshes und Verkehrstopologien 222
 - Software-defined Infrastructure 72
 - Swarm 133
 - Terraform 72
 - Tracing 222
 - Vagrant 72
 - Workload (interaktives Jupyter Notebook) 38
- Umgebungsvariable 48
- Unabhängige Aktualisierbarkeit 157, 223

Unabhängige Austauschbarkeit 223
 Uniform Resource Identifier (URI) 164
 Union Filesystem 80
 – Copy-on-Write 80
 – Layer 80
 – Namensraum 80
 Unterstützende Subdomäne 228
 Upstream-Service 156
 US CLOUD Act 1

V

Vagrant 66
 – Box 66
 – Provider 67
 – Provisioner 67
 – Vagrantfile 66
 Value Object 242
 vCPU 59, 116
 Vendor Lock-in 14, 76
 Verfügbarkeitszone 57
 Verhaltensanalyse 245
 Verkehrsfluss 130
 Verkehrstopologie 220
 Verschlüsselung 217
 Versionierungsschema 170
 Versionsverwaltungssysteme 25
 Vertikale Skalierung 174
 Virtualisierung 22, 59
 – Betriebssystem 76
 – Hardware 59
 Virtual Private Network (VPN) 218
 Virtual Service 215
 Virtuelle Netzwerkschnittstelle 59
 VLAN 59
 Voll-Virtualisierung 60
 Volume Provisioner 126
 Volunteer Computing 185
 Vorwärtskompatibilität 169

W

Wegwerf-Komponente 89
 Wegwerf-Umgebung 65
 Wertschöpfungskette 26

Whitebox-Instrumentierung 196
 Whitebox-Monitoring 199
 Whitebox-Tracing 202
 Whitebox-Überwachung 196
 Widerstandsfähigkeit 168
 Wiederholung (Retry) 216
 Workload 17, 94
 – einmalig/selten 18
 – Heterogenität 95
 – Isolation 127
 – kontinuierlich sinkend 19
 – kontinuierlich steigend 19
 – periodisch 17
 – statisch 17
 – zufällig 18
 Workload-Allokation 94, 101
 Workload-Ausführung 95
 Workload-Queue 98
 Workload-Scheduler 98

X

X.509 218
 X-Trace 202

Y

YAML, Notation 6
 YARN 98, 99
 You build it, you run it 28, 223

Z

Zeitreihe 198
 Zeitreihen-Datenbank 194, 199
 Zero-Trust Networking 217
 Zertifikat-Handling 217
 Zertifizierungsstelle (CA) 218
 Zone 57
 Zugriffskontrolle 212
 Zustandsanalyse 245
 Zustandslosigkeit 137, 164
 Zuteilungsdauer 19
 Zwei-Wege-Authentifizierung 218
 Zwölf-Faktoren-Methodik 84
 Zwölf-Faktoren-Modell 35