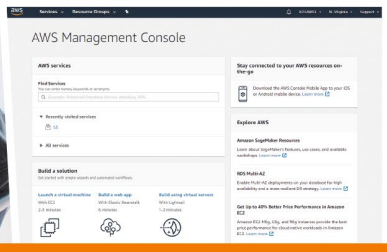
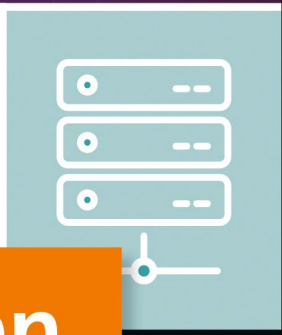
Kevin Welter
Daniel Stender

IaaS mit Azure, AWS und GCE





Cloud-Infrastrukturen

Das Handbuch für DevOps-Teams und Administratoren

- Infrastrukturen in der Cloud aufbauen
- Toolchains für Rollout, Orchestrierung und Monitoring
- Mit Praxiswissen zu Python, Go, Ansible, Terraform, Docker und Kubernetes

Alle Beispielprojekte zum Download

Rheinwerk Computing

Auf einen Blick

Inhalt

Einleitung

»The Cloud is the great equalizer because it gives small businesses access to technologies that previously only large companies could afford«
– Kevin O'Leary

Stellen Sie sich vor, Sie drücken einen Knopf, und mit wenigen Zeilen Code wird ein ganzes Rechenzentrum für Sie erschaffen – und ohne dass Sie die schweren Server in Racks einbauen oder endlose Kabelstränge verlegen müssen. Was einst nach Science-Fiction klang, ist heute Realität und liegt buchstäblich in Ihren Händen. Die Cloud-Technologie hat sich mit atemberaubender Geschwindigkeit weiterentwickelt und bietet uns eine beispiellose Freiheit: Infrastruktur und Skalierung lassen sich heute so flexibel und effizient managen, dass sie fast nahtlos in den Hintergrund treten.

Ich erinnere mich noch gut an meinen ersten Besuch in einem echten Rechenzentrum. Es war in Mainz im Jahr 2012, bei IBM. Damals war das Betreten dieser Welt aus Hardware ein richtiges Ereignis – man betrat einen riesigen Raum voller Schränke mit surrenden Maschinen, blinkender Lichter und einer geradezu elektrisierenden Geräuschkulisse. Reihen um Reihen an Servern, jeder nur wenige Zentimeter voneinander entfernt, waren verbunden durch ein Netz endloser Kabel, die durch den Boden geleitet werden. Die Anlage musste aufwendig gekühlt werden, um die heiße Luft der Server auf ein erträgliches Maß zu bringen, und die Infrastruktur, die für einen ausfallsicheren Betrieb notwendig war, ähnelte der Versorgung einer kleinen Stadt. Damals war diese Szenerie das Herzstück der digitalen Welt, und nur wenige hatten das Privileg, sie aus nächster Nähe zu erleben oder sogar dort zu arbeiten.

Heute, mehr als ein Jahrzehnt später, ist das alles in vielerlei Hinsicht nostalgisch geworden. Die Arbeit in Rechenzentren, wie ich sie damals gesehen habe, gehört für die meisten IT-Fachleute kaum noch zum Alltag. Nur die wenigsten Unternehmen betreiben so aufwendige Infrastrukturen noch selbst, sie lagern sie lieber an spezialisierte Anbieter aus. Das hat natürlich seinen Charme. Das Rechenzentrum muss nicht mehr selbst unterhalten werden, und die Cloud-Anbieter machen den Einstieg sehr einfach. Mit wenigen Klicks läuft bereits ein erster Prototyp.

Doch diese Einfachheit bringt auch neue Herausforderungen mit sich. Die Cloud macht uns zwar die Arbeit leicht, doch ganz schnell wird sie auch eine kostspielige Falle. Die Prinzipien bleiben die gleichen wie damals im Rechenzentrum: Ressourcen

müssen gut geplant und verantwortungsbewusst verwaltet werden. Es geht immer um die zentrale Frage: Wie gestalten wir unsere Infrastruktur und Arbeitsweise, damit sie uns und unsere Projekte wirklich voranbringen?

In einer Welt, in der nahezu alles automatisiert und abstrahiert werden kann, liegt die Kunst darin, bewusst zu steuern, welche Tools und Prinzipien wir wählen, um langfristig Mehrwert zu schaffen. Und das geht über die Auswahl der richtigen Werkzeuge hinaus – es fordert uns, Cloud-Infrastruktur nicht nur als technischen Fortschritt zu verstehen, sondern als Weg, unsere Ideen schneller und verlässlicher in die Welt zu bringen. Dazu braucht es jedoch zuerst das richtige Know-how.

Das Buch, das Sie in den Händen halten, ist mehr als eine Anleitung – es soll ein Begleiter auf Ihrem Weg in die Welt der modernen Cloud-Infrastruktur sein. Mit praxisnahen Anleitungen, bewährten Strategien und Einblick in aktuelle Technologien hilft es Ihnen, das volle Potenzial der Cloud zu erkennen und effizient auszuschöpfen. Dabei lenkt es den Blick immer wieder auf das Wesentliche: eine Infrastruktur aufzubauen, die Sie verstehen und die Ihre Projekte nicht nur unterstützt, sondern aktiv stärkt und voranbringt.

Die 2. Auflage

Die erste Auflage dieses Buchs erschien im Frühjahr 2020 zu Beginn der Corona-Zeit. Daniel Stender hatte mit praxisnahen Beispielen und vielen Erklärungen ein tolles Nachschlagewerk für Cloud-Infrastrukturen geschrieben, das sein tiefes Verständnis des Themas zeigte. Es war und ist darauf ausgelegt, den Einstieg zu erleichtern und gleichzeitig fortgeschrittenen Anwendern wertvolle Einblicke zu bieten. Viele der Inhalte haben auch heute noch Bestand und sind weiterhin relevant und das, obwohl sich die Welt der IT so schnell verändert.

Nach dem Tod von Daniel habe ich die Überarbeitung der Inhalte übernommen. Die zweite Auflage baut also auf dieser soliden Grundlage auf. Jedes Kapitel habe ich sorgfältig überarbeitet und an die neuesten Versionen der Software angepasst, damit Sie die Übungen weiterhin problemlos durchführen können. Darüber hinaus wurden an einigen Stellen – wie im Kapitel 9 über SDKs – neue Inhalte hinzugefügt, um Ihnen zusätzliche Möglichkeiten zur Automatisierung und Integration zu bieten.

Das sollten Sie schon wissen

Ihnen ist vielleicht aufgefallen, dass der Verlag dieses Buch in die Abteilung *IT-Administration* eingereiht hat, und zwar unter *Linux*. Das freie Betriebssystem spielt die Hauptrolle für das Public Cloud Computing, und das hat einen bestimmten Grund: Infrastruktur-Ressourcen aus der Cloud sind gemietete virtuelle Server, die über das Internet gesteuert werden. Und bei Servern ist Linux das wichtigste Betriebssystem.

Gleichzeitig sollten Sie Linux für das Public Cloud Computing aber auch auf Ihrem Arbeitsrechner einsetzen. Der Grund dafür ist, dass die vorgestellten Werkzeuge und Frameworks alle Open-Source-Software sind, und Linux stellt dafür nun einmal die offenste Plattform dar. Da macOS im Kern selbst ein *unixoides* (von Unix abstammendes) System ist, gibt es auch dort einfache Wege, dieselbe Software zum Laufen zu bringen, sodass Apple-Benutzer die Linux-Literatur meist mit Gewinn lesen können.

Sofern es möglich ist, kommen hier aber auch Windows-Benutzer nicht zu kurz, da es immer auch Hinweise für die Installation der besprochenen Software-Lösungen unter Windows gibt. Das ist allerdings leider nicht immer möglich; so unterstützt z. B. Ansible, das einen Schwerpunkt in diesem Buch darstellt, Windows als Betriebsplattform bisher nicht offiziell. Es ist daher wirklich nützlich, wenn Sie ein Linux-System nutzen oder sich zumindest mit dem *Windows Subsystem for Linux* (WSL) behelfen können.

Ich möchte allerdings den Eindruck vermeiden, dass Cloud-Computing nur mit einem Linux-Arbeitsrechner möglich ist. Das ist ganz und gar nicht der Fall! Nichtsdestotrotz gehen die Anwendungsbeispiele von einem Arbeitsrechner aus, der unter Ubuntu läuft. Der Umgang mit Windows-Servern aus der Cloud hingegen ist ein Spezialgebiet, auf das hier leider nicht eingegangen werden kann.

Vor diesem Hintergrund sollten Sie zumindest ein Grundwissen im Bereich Linux-Anwendung und -Administration mitbringen. Linux-Enthusiasten wissen ohnehin, wie nützlich die Shell im Alltag sein kann, und für den Umgang mit Cloud-Servern gilt das auf jeden Fall. Je besser Sie die Shell-Werkzeuge kennen, desto leichter wird Ihnen der Umgang mit den Cloud-Systemen fallen.

Alle hier vorgestellten Tools für das Cloud Computing und das Cloud Engineering sind nämlich CLI-Werkzeuge, sodass die Linux-Kommandozeile in diesem Buch die Hauptrolle spielt. Die vielfältigen Möglichkeiten, die Ihnen dort für den Einsatz der Tools zur Verfügung stehen, werden bei jeder sich bietenden Gelegenheit hervorgehoben. Wenn Sie das Buch durcharbeiten, werden Sie deshalb mit Sicherheit Ihre Kenntnisse in der Arbeit auf der Linux-Kommandozeile erweitern. Am meisten haben Sie von den Beispielen, wenn Sie möglichst viele detailliert nacharbeiten und selbst ausprobieren.

Der Linux-Schwerpunkt führt dazu, dass dieses Buch auch zusammen mit der auf dem Markt verfügbaren Literatur über Linux-Administration gelesen werden kann. Falls Sie ein allgemeines Linux-Handbuch hinzuziehen möchten, dann sei Ihnen der Titel »Linux-Server: Das umfassende Handbuch« (Rheinwerk Verlag, ISBN 978-3-8362-9615-1) wärmstens empfohlen. Das ist mit Sicherheit einer der einschlägigsten deutschsprachigen Titel für dieses Gebiet.

Kosten

Ein wichtiger Punkt zum Schluss: Wenn Sie die hier vorgestellten Cloud-Services nutzen, dann entstehen Ihnen Kosten. Die Maschinen, die ich in den Beispielen verwende, haben allerdings ausschließlich kleine und kleinste Größen, deren Mietpreis möglichst niedrig ist. Meistens werden sie von den Freikontingenten abgedeckt, die die Anbieter bei der Neuanmeldung für Sie einrichten. Wenn Sie die Beispiele ausprobieren, dann nutzen Sie die Maschinen deshalb entweder kostenlos (d. h. im Rahmen des Freikontingents), oder Sie landen im niedrigen zweistelligen Euro-Bereich. Falls Ihnen allerdings unerwartete Kosten entstehen, weil Sie andere Maschinengrößen einrichten als diejenigen aus den Beispielen, dann müssen Sie selbst dafür sorgen, dass Sie den Überblick behalten und kostenbewusst mit den Ressourcen umgehen. Zusätzliche Kosten entstehen auch, wenn Sie vergessen, die Maschinen nach dem Ausprobieren wieder zu terminieren.

Danksagung

Mein tiefster Dank gilt meiner Frau Nicole und meinen Kindern Levi Ace und Killua Caden. Danke, dass ihr mir den Rücken stärkt und mir immer wieder zeigt, was im Leben wirklich zählt. Ohne eure Unterstützung und Geduld wäre dieses Buch nicht möglich gewesen.

Kevin Welter

Kapitel 1
Cloud Computing

»There is no cloud, it's just someone else's computer.«
– Sprichwort in der Cloud-Computing-Szene

Ich kann mich noch daran erinnern, wie beeindruckt ich war, als ich vor einigen Jahren in Hamburg das erste Mal mit einem Wagen von einem großen Carsharing-Anbieter unterwegs war. Diese Spielart von öffentlichen Verkehrsmitteln war damals noch ganz neu. Mich faszinierte vor allem, wie zu dieser Zeit bereits vorhandene Techniken eingesetzt und kombiniert wurden, um Mietwagen auf diese damals neue Art und Weise anbieten zu können.

Eine spezielle App auf dem Smartphone erkennt über GPS die eigene Position und öffnet eine Karte, auf der alle im Stadtgebiet und vor allem im näheren Umkreis parkenden und freien Fahrzeuge des Anbieters verzeichnet sind. Der Kunde kann mit der App direkt Informationen über einzelne Wagen, zum Beispiel den Füllstand des Tanks, abrufen. Er sucht sich dann einen passenden Wagen aus, und der wird dann für einen gewissen Zeitraum für diesen Kunden reserviert. Die Wagen stehen mit der Zentrale über Mobilfunk-Netzwerke in ständiger Verbindung (in den Großstädten besteht dafür eine hinreichende Netzabdeckung) und erhalten über diesen Weg die Information, dass sie reserviert sind. Das Auto ist dann für die Benutzerkarten anderer Carsharing-Kunden gesperrt, sodass niemand dem Kunden den Wagen wegschnappen kann, während er gerade auf dem Weg zu ihm ist. Der Kunde schaltet den Wagen dann mit seiner Karte frei, setzt sich hinein und fährt los. Am Ziel stellt er den Wagen ab und loggt sich wieder aus. Nur ein paar Sekunden später erscheinen die genauen Daten der Fahrt und der für die zurückgelegte Entfernung berechnete Preis im eigenen Kundenkonto auf der Homepage des Anbieters im Internet.

Im Winter sprang einmal ein Wagen nicht an, und ich konnte mich nicht ausbuchen, weil der Bordcomputer keine Verbindung zur Zentrale mehr bekam: Das Auto stand wohl ungünstig in einem Funkschatten. Für solche Fälle gibt es aber einen speziellen Serviceknopf, und wenn der gedrückt wird, baut der Wagen über Satellitenfunk eine Serviceverbindung zu der Zentrale auf. Die freundliche Support-Mitarbeiterin am anderen Ende buchte dann den Wagen für mich aus. Sie meinte, er werde jetzt von ihr stillgelegt, und ein Techniker komme zur aktuellen GPS-Position, um den Wagen auf seinen Transporter zu laden und in eine Servicewerkstatt zu bringen. Der nächste freie Wagen, mit dem es keine Schwierigkeiten gab, stand nur ein paar Meter weiter.

»Eigentlich braucht man heutzutage gar kein ein eigenes Auto mehr«, dachte ich bei mir. Und wie einfach es sein kann, wenn auf eine solch einfallsreiche Weise Techniken wie normale Autos, Software, das Internet, Smartphones, das Global Positioning System GPS, Handy- und Satelliten-Funk miteinander in einem Mietmodell zu einem innovativen und zukunftsweisenden Produkt kombiniert werden. Der Kunde ist damit von allen Lasten befreit, die der Besitz eines Kraftfahrzeugs mit sich bringt. Und er bezahlt, weil er selbst fährt, auch nur einen Bruchteil von dem, was ein Taxi kostet. Der Anbieter macht gleichzeitig durch viele kleine Summen gute Gewinne und baut das Angebot immer weiter aus, was es zunehmend attraktiver macht.

Es gibt beim Carsharing und ähnlichen Angeboten einige Parallelen zum Cloud Computing. Und dasselbe Gefühl, damit plötzlich in die Zukunft versetzt zu werden, mag sich beim ersten Kennenlernen dieser Vermietungstechnik vielleicht genauso einstellen (mittlerweile vielleicht aber auch nicht mehr).

1.1 Welcome to the Cloud

Beim *Cloud Computing* beziehungsweise beim *Public Cloud Computing* (wie sich das gegen die anderen Varianten abgrenzt, werde ich weiter unten besprechen) handelt es sich zunächst einmal genauso um ein Mietangebot. Und bei dem Angebotsmodell *Infrastructure-as-a-Service* (das im Mittelpunkt dieses Buchs steht) werden statt Kraftfahrzeugen reine Computing-Ressourcen wie CPU-Leistung, Arbeitsspeicher, Massenspeicher und Netzwerke für die allgemeine Benutzung zur Verfügung gestellt.

Als Grundlage dafür dient natürlich das Internet. Denn anstatt Serverracks auf den Hof ausgeladen zu bekommen, werden beim Public Cloud Computing die angemieteten Ressourcen im weltweiten Breitband-Netzwerk über Standardprotokolle zur Verfügung gestellt. Damit können Sie mit beliebigen Geräten auf diese Ressourcen zugreifen, und es spielt – abgesehen von eventuellen Latenzzeiten und rechtlichen Fragen – grundsätzlich keine Rolle, wo genau sich die genutzten Rechner befinden.

1.1.1 Virtualisierung

Eine weitere zentrale Technik, die Cloud Computing überhaupt erst möglich macht und dafür sorgt, dass Computing-Ressourcen in einer solchen Art und Weise über das Internet angeboten werden können, ist die Virtualisierung von Servern. Dafür setzen die Anbieter spezielle Softwarelösungen ein (Hypervisoren), welche die Hardware von physischen Servern abstrahiert zur Verfügung stellen können. Die Hypervisoren laufen entweder direkt auf Rechner-Hardware (eine Formulierung dafür ist »auf dem Blech«) als Betriebssystem-Ersatz oder werden auf dem Betriebssystem installiert, das auf einer Maschine läuft.

Das macht es möglich, ein System in mehrere kleine, beliebig zugeschnittene Stücke zu unterteilen. Diese sind dann als einzelne virtualisierte Server benutzbar, die sich den Betriebssystemen gegenüber, die wiederum auf ihnen installiert werden, wie eigenständige Rechner verhalten. Ohne dass der darunter liegende Server neu gebootet werden muss, können virtuelle Maschinen (VMs) beliebig oft hoch- und heruntergefahren werden. Es handelt sich um eigenständige, unabhängige Einheiten, die grundsätzlich nichts miteinander zu tun haben.

Dabei wird gemeinhin mit Images (im Sinne von »Maschinenabbildern«) gearbeitet, die unterschiedliche anbieterspezifische Dateiformate haben. Die Images dienen als Pseudo-Festplatten, von denen die VMs gestartet werden. Auch wird der Zustand von heruntergefahrenen VMs mithilfe der Images auf dem Host-Betriebssystem aufbewahrt.

1.1.2 Hypervisoren

Hypervisoren bilden die abstrahierende Schicht zwischen Gast- und Hostsystem. Sie stellen dem Gast eine virtuelle Umgebung zur Verfügung und verwalten die Erzeugung von virtuellen Maschinen. Es gibt eine ganze Reihe von Hypervisoren von verschiedenen Herstellern. Im Cloud Computing werden aber immer wieder dieselben bewährten und frei lizenzierten Open-Source-Lösungen von Infrastruktur-Anbietern eingesetzt.

Freie Software

Die Begriffe *Open Source* und *freie Software* gelten gemeinhin als Synonyme. Open Source ist dabei der gängigere Ausdruck und wird ganz bewusst für die Vermarktung von nicht proprietären Softwareprodukten eingesetzt. Allerdings gibt es quelloffene Software, die nicht frei lizenziert ist. Darüber hinaus wird unter »freier Software« meist Software verstanden, die umsonst und frei von Lizenzkosten ist. Der Freiheitsbegriff bei Softwarelizenzen ist allerdings weiter gefasst. Er meint vor allem vom Hersteller beziehungsweise vom Entwickler eingeräumte Freiheiten im Umgang mit dem Quellcode. Das betrifft spezielle urheberrechtliche Fragen wie etwa, ob Sie den Code verändern und danach unter demselben Namen weiterverteilen dürfen.

Frei lizenzierte Softwareprodukte sind allerdings in der Regel auch frei von Lizenzgebühren. Die Open-Source-Hersteller verdienen ihr Geld meist durch Support-Verträge oder durch kommerzielle Varianten mit einem erweiterten Funktionsumfang. Allerdings gilt als Minimalstandard für freie Software, dass sie ohne Einschränkungen vertrieben werden darf. Es bedeutet aber nicht in jedem Fall, dass Sie diese Software ohne jegliche Einschränkungen kommerziell verwenden dürfen. Informieren Sie sich also genau über die jeweiligen Lizenzbedingungen, wenn Sie frei lizenzierte Softwareprodukte professionell einsetzen wollen. Nur weil eine Software nichts kostet, können Sie damit nicht automatisch machen, was Sie wollen.

Zu den weitverbreiteten Lösungen zählt zum einen die *Kernel-Based Virtual Machine* (KVM), die 2007 von dem israelischen Unternehmen Qumranet vorgestellt wurde, das im Jahr darauf von Red Hat aufgekauft wurde. Dieser Hypervisor ist seit Version 2.6.20 als Modul fester Bestandteil des Linux-Kernels und bildet den Quasi-Standard der Maschinenvirtualisierung unter Linux. KVM wird oftmals zusammen mit dem Paravirtualisierungstreiber *Virtio* in der VM eingesetzt. Dieser ermöglicht unmittelbare I/O-Zugriffe auf die unvirtualisierte Hardware des darunter liegenden Hosts, was einen stattlichen Zugewinn an Performance bietet.

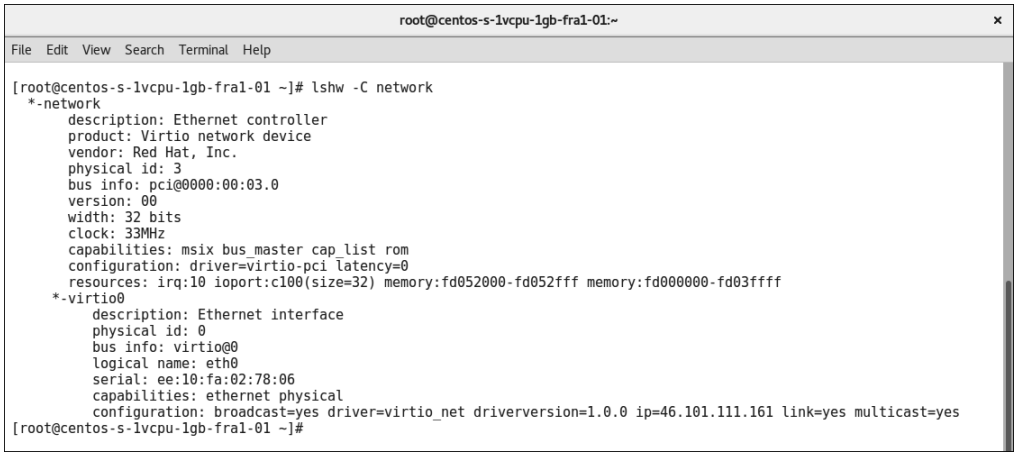


Abbildung 1.1 Per Virtio-Treiber eingebundene Netzwerkkarte bei DigitalOcean

Ein weiterer von Cloud-Anbietern auf breiter Front eingesetzter Hypervisor für Linux-Systeme ist *Xen*, der im Jahr 2007 vom US-amerikanischen Softwareunternehmen Citrix aufgekauft wurde. Xen läuft sozusagen neben dem Betriebssystem, von dem es gestartet worden ist, unmittelbar auf dem Host. Diese Lösung bietet damit eigene Paravirtualisierungskapazitäten und ist der Standard-Hypervisor zum Beispiel bei den Instanzen von Amazons *EC2*. (Dieser IaaS-Dienst wird in Abschnitt 3.1 noch ausführlich thematisiert.)

Beide Hypervisoren nutzen die in modernen x86-Prozessoren gemeinhin eingebaute Hardwareunterstützung für die Virtualisierung. Der Chiphersteller Intel verbaut dafür die Technik *Intel Virtualization Technology (Intel VT)*, und AMD produziert seine *AMD Virtualization (AMD-V)*.

Ein weiterer Hypervisor, der besonders beim Cloud-Computing auf Azure eingesetzt wird, ist *Hyper-V* von Microsoft.

Im Gegensatz zur Technik der Emulation, die systemfremde Architekturen nachstellt, findet sich bei der Virtualisierung im Gastsystem immer dieselbe Maschinenarchitektur wie diejenige des darunter liegenden Servers. Kostengünstige und die allge-

meine IT-Welt dominierende x86-Maschinen spielen auch beim Public Cloud Computing die Hauptrolle.

Cloud-Anbieter betreiben manchmal selbst entwickelte Lösungen für die Virtualisierung. So setzt Amazon zum Beispiel bei EC2 neben Xen den eigenen kompakten Hypervisor *Nitro* ein, der auf KVM-Technologie basiert.

1.1.3 Rechenzentrum

Auch in »klassischen« Rechenzentren ohne Cloud-Ausrichtung hat es sich mittlerweile als Betriebskonzept durchgesetzt, eine Virtualisierungsschicht über der vorhandenen Hardware zu betreiben. Diese Technik hat mehrere Vorteile: Die Ressourcen der vorhandenen Infrastruktur werden optimal ausgenutzt, die Konfiguration und Administration werden vereinfacht, und der Betrieb wird effektiver und damit kostengünstiger.

Darüber hinaus bildet dieses Verfahren aber auch das Fundament für das Cloud Computing:

- Anbieter mit einheitlicher Hardware können unterschiedliche virtualisierte Servergrößen zur Vermietung anbieten.
- Instanzen können ad hoc, also nach aktuellem Bedarf, gebildet und schnell auch wieder aufgelöst werden.
- Je nach Kundenwunsch können dabei auch unterschiedliche Betriebssysteme zum Einsatz kommen.

Auf der Virtualisierungsschicht steht ein umfassender Ressourcen-Pool zur Verfügung, aus dem sich unterschiedliche Server über mehrere physische Rechner hinweg aufteilen oder zusammenfassen lassen.

Viele Infrastruktur-Anbieter haben mittlerweile auch unvirtualisierte Instanzen als sogenanntes *Bare Metal* im Programm. Diese werden zu einem Aufpreis als rein physische Hardware ohne Hypervisor zur Verfügung gestellt und eignen sich besonders für bestimmte Anwendungszwecke (siehe weiter unten). Public Cloud Computing findet in den meisten Fällen aber mit virtualisierten Servern statt.

1.1.4 Die »Wolke«

Vor diesem Hintergrund ist der Begriff von der »Cloud« als einer »Datenwolke« oder vielleicht besser gesagt »Rechnerwolke« meiner Meinung nach durchaus treffend geprägt. Handelt sich bei diesen Gebilden doch um amorphe, sich ständig verändernde und selbstorganisierende riesige Massen von virtualisierten Hardwareressourcen. Die »Wolke« bildet dabei einen Pool von virtualisierten Computing-Rohstoffen, aus

welchen Server-Instanzen bei Anforderung vom Anbieter dynamisch zugewiesen werden.

Mit der tatsächlich vorhandenen Hardware kommen Sie dabei normalerweise nicht mehr in Berührung, und Ihre Softwarestacks sind nicht mehr an physische Rechner gebunden. Sie wissen auch nicht – und Sie müssen es für den erfolgreichen Einsatz der angebotenen Technik auch gar nicht wissen –, wo genau sich die Maschine befindet, auf die Sie gerade Softwarepakete installieren. Irgendwo in einem großen Rechenzentrum (vielleicht in Frankfurt am Main, vielleicht aber auch woanders), am Ende eines langen Gangs voller prall gefüllter Server-Racks flimmert dabei ein Betriebslämpchen. Wenn Sie Ihre Instanz dann löschen und sich direkt eine neue zulegen, dann ist diese vielleicht in einem ganz anderen Rechenzentrum gehostet – wie auch immer die Cloud das gerade benötigt.

1.1.5 Die »Cloud Computing Definition«des NIST

Cloud Computing ist hip und äußerst komplex. Deshalb kommen Begriffe wie »Cloud-native« oder »Cloud-ready«, die sich nur schwer konkret definieren lassen, in der IT-Szene heutzutage gefühlt in jeder Präsentation vor, und zwar in nahezu beliebigen Zusammenhängen. Damit soll meist demonstriert werden, dass man auf dem neuesten Stand der Technik agiert. Und in der Anfangszeit dieser technischen Entwicklung, gegen Ende der 90er-Jahre des vergangenen Jahrhunderts, wurde der damals noch allzu schwammige Begriff »in der Cloud sein« auch gern in der Werbung eingesetzt, um etwa ein omnipotentes Betriebssystem zu suggerieren.

Wohltuend konkret ist dagegen die *Cloud Computing Definition (Special Publication 800-145)* des US-amerikanischen National Institute of Standards and Technology (NIST) von 2001:

https://csrc.nist.gov/publications/detail/sp/800-145/final

Diese kurz gefasste Broschüre hat das Verständnis der Grundkategorien des Cloud Computing maßgeblich geprägt und ist auch heutzutage immer noch sehr einflussreich. Die NIST-Definition unterscheidet unter anderem vier grundsätzliche Bereitstellungsvarianten (*deployment models*), von denen die *Public Cloud* den maßgeblichen Schwerpunkt dieses Buchs darstellt. Doch zunächst – und auch zur Abgrenzung davon – folgen einige Bemerkungen zum Thema *Private Cloud*.

1.1.6 Private Cloud

Eine der Bereitstellungsvarianten, die von der NIST definiert wurden, ist die *Private Cloud*. Dies ist eine Cloud-Umgebung, die nicht öffentlich zugänglich ist und ausschließlich für ein bestimmtes Unternehmen oder eine Organisation betrieben wird.

Oftmals wird der Begriff *Private Cloud* so verstanden, als wäre damit gleichzeitig auch die Infrastruktur gemeint, mit der ein Unternehmen diese Cloud betreibt. Dabei schwingt mit, dass die Cloud dafür hausintern zu sein habe und sich auf dem firmeneigenen Gelände (Stichwort: *On-premises*) befindet. Geschlossene Cloud-Umgebungen für Firmen und Organisationen werden aber nicht nur von diesen selbst, sondern auch von Outsourcing-Anbietern betrieben. Die Definition der Private Cloud der NIST schließt deshalb ausdrücklich mit ein, dass sich ein für geschlossene Cloud-Dienstleistungen benötigtes Rechenzentrum auch in der Firma eines externen Anbieters befinden kann. Ein treffenderes Unterscheidungsmerkmal zwischen Private und Public Cloud ist also eher, ob Services daraus über das Internet oder ein Intranet genutzt werden.

»On-premise« vs. »On-premises«

Der in deutschen Publikationen oftmals vorkommende Ausdruck »On-premise« für »auf dem Firmengelände« ist schlichtweg falsches Englisch. Denn »premise« ist nicht der Singular von »premises« – engl. »Firmengelände, Gewerberäume« –, sondern bedeutet »Prämisse«.

Eine Private Cloud findet man heutzutage vor allem bei Konzernen und größeren Unternehmen. Besonders häufig trifft man sie in stark regulierten Bereichen wie Banken, Versicherungsgesellschaften und Finanzdienstleistern an, die besondere Auflagen erfüllen müssen. Solche Unternehmen können beziehungsweise dürften ihre Daten nämlich gar nicht in einer Public Cloud verarbeiten (weiter unten dazu mehr).

Es gibt einige Softwarelösungen, um eine eigene Cloud mit Infrastruktur-Service und anderen Diensten in einem Rechenzentrum aufzusetzen. Weit verbreitet hierfür sind:

- die proprietäre Virtualisierungslösung *vSphere* von VMware in Verbindung mit *vCloud*, die mit einem eigenen Hypervisor namens ESXi arbeitet.
- Daneben gibt es das frei lizenzierte *Open Stack* (*https://www.openstack.org/*). Es setzt hauptsächlich auf den Hypervisor KVM auf, aber auch Xen und Hyper-V von Microsoft werden unterstützt.

Bei Open Stack handelt es sich zweifelsohne um eines der Juwelen der Open-Source-Szene. Die Software ist von der US-amerikanischen Raumfahrtagentur NASA zusammen mit dem Hosting-Anbieter Rackspace initiiert worden. Sie wird mittlerweile von einer ganzen Reihe von namhaften Unternehmen und Organisationen eingesetzt. Dazu gehören zum Beispiel das Teilchenforschungsinstitut CERN bei Genf mit seinem hohen Bedarf an Kapazitäten sowie die Volkswagen AG als größter europäischer Konzern.

Open Stack

Open Stack ist eine sehr komplexe Sammlung von mehreren Komponenten, die für die Zusammenarbeit gedacht sind und für sich allein genommen meist keinen Nutzen haben.

Einige Basismodule werden für eine vollständige Private-Cloud-Umgebung benötigt. Den Kern bildet dabei der Authentifizierungs- und Autorisierungsdienst *Keystone*, mit dem sich auch die Komponenten untereinander identifizieren. Der Imagedienst *Glance* bewahrt die Maschinenabbilder für die virtuellen Maschinen auf. Die Komponente *Nova* ist für das Management der virtualisierten Maschinen zuständig.

Wichtige weitere Komponenten sind *Cinder*, das persistenten Speicher zur Verfügung stellt, der Objektspeicher *Swift* (nicht zu verwechseln mit der Apple-Programmiersprache) und die Lösung für virtuelle Netzwerke *Neutron*. Darüber hinaus gibt es zum Beispiel auch die Database-as-a-Service-(DBaaS-)Lösung *Trove*.

Open Stack wird meistens in Form von spezialisierten Distributionen wie *Red Hat OpenStack Platform* (RHOP) oder *SUSE OpenStack Cloud* betrieben. Besonders eng verwoben ist es allerdings mit Ubuntu. Der Ubuntu-Anbieter Canonical trägt durch das starke persönliche Interesse von CEO Mark Shuttleworth viel zur Entwicklung von Open Stack bei, zum Beispiel mit dem Container-Hypervisor *LXD*.

Open Stack ist in Python implementiert und steht unter der Apache-2.0-Lizenz.

1.1.7 Weitere Cloud-Varianten

Wie sich nun eine Public Cloud von einer Private Cloud unterscheidet, ist ohne Weiteres nachvollziehbar: Public-Cloud-Anbieter stellen auf öffentlich zugänglichen Cloud-Servern standardisierte IT-Ressourcen für jedermann über eine Internetverbindung zur Verfügung. Und die dafür benötigten Rechenzentren stehen unter der Kontrolle des jeweiligen Anbieters.

Die NIST-Definition kennt darüber hinaus aber auch noch die *Community Cloud*. Das ist eine Public-Cloud-Lösung, die allerdings nur für einen bestimmten Benutzerkreis zugänglich gemacht wird.

Die vierte Variante von Cloud Computing nach der NIST ist die *Hybrid Cloud*. Das ist eine Lösung, die eine Public und eine Private Cloud auf eine bestimmte Art miteinander kombiniert. Ich werde diese aktuell sehr gefragte Variante weiter unten in Abschnitt 3.6 zusammen mit Multi-Cloud-Lösungen vorstellen.

1.1.8 Services

Eine weitere, äußerst populär gewordene Kategorie des Cloud-Computings nach der NIST sind die grundlegenden Dienst-Modelle (*service models*). Es handelt sich dabei

um *Software-as-a-Service (SaaS)*, *Platform-as-a-Service (PaaS)* und das bereits erwähnte *Infrastructure-as-a-Service (IaaS)*. Letzteres ist das Thema dieses Buches.

- Bei SaaS handelt es sich um den von einem Cloud-Anbieter angebotenen Nutzungszugang zu Softwareprodukten und Anwendungsprogrammen. Die vom Provider auf seiner eigenen Infrastruktur betriebenen Produkte werden meistens über einen Webbrowser benutzt. Zu dieser Serviceklasse gehört zum Beispiel die Office-Suite Office 365 von Microsoft. Nach demselben Prinzip werden aber auch spezielle Business-Lösungen, wie zum Beispiel für ERP (*Enterprise Resource-Planning*) und CRM (*Customer-Relationship-Management*), angeboten.
- Bei PaaS-Diensten handelt es sich um Anwendungsinfrastruktur in Form von technischen Frameworks, die fertige Middleware, Datenbanken und anderes beinhalten. Der Anwender nutzt sie, um beliebige Web-Applikationen dort aufzuspielen und sie so in der Cloud beziehungsweise im Internet zur Verfügung zu stellen. Zu dieser Cloud-Klasse gehört zum Beispiel die *App Engine* von Google. Der auf PaaS spezialisierte Cloud-Anbieter Heroku (*https://www.heroku.com/*) stellt sich mit seinem Angebot als praktischere Alternative zu konventionellen Webhostern auf.
- Beim IaaS werden reine Hardwareressourcen zur Benutzung angeboten. Darum geht es in diesem Buch.

Wenn Sie sich bereits mit Cloud Computing beschäftigt haben, wissen Sie, dass mittlerweile ein kaum noch überschaubares Angebot von verschiedenartigen Diensten existiert. Um diese Tatsache auszudrücken, existiert mittlerweile der oft ironisch gemeinte Begriff *Everything-as-a-Service (XaaS)*. Es gibt sogar Etiketten-as-a-Service aus der Cloud.

Die Kategorien des NIST-Modells treffen dabei nicht mehr auf alles richtig zu. Aber auch die von vielen genutzten Cloud-Services wie der Filehosting-Dienst Dropbox (*https://www.dropbox.com/*) oder das *Serverless Computing* lassen sich keinem der drei NIST-Modelle mehr klar zuordnen. Trotzdem tragen die drei definierten Dienstklassen als Grundkategorien zum Verständnis des Cloud Computing weiterhin maßgeblich bei. Sie machen auch nicht zuletzt deutlich, wie die Cloud intern aufgebaut ist.

Serverless Computing

Das sogenannte *Serverless Computing* beziehungsweise *Function-as-a-Service* (FaaS) ist eine relativ neue Entwicklung im Cloud Computing.

Diese Serviceklasse ist zwischen PaaS und SaaS angesiedelt und bietet einen noch höheren Abstraktionsgrad als PaaS. Der Anbieter kümmert sich dabei um den gesamten Betrieb all der Dinge, die notwendig sind, um reinen Code zum Laufen zu bringen. Der Kunde muss nur noch seine Codeschnipsel hochladen und nimmt einige rudimentäre Einstellungen wie das Timeout vor.

FaaS-Dienste sind zum Beispiel *AWS Lambda*, *Microsoft Azure Functions* und *Google Cloud Functions*, die jeweils verschiedenen Programmiersprachen unterstützen. FaaS-Funktionen können auf alle Services des jeweiligen Anbieters zugreifen. Sie müssen einem bestimmten zustandslosen Programmiermodell folgen und lassen sich in der Cloud durch verschiedene Ereignisse auslösen (zum Beispiel, dass Alexa irgendetwas Bestimmtes gesagt bekommt).

Eine Funktion im FaaS wird nur berechnet, wenn sie auch läuft. Amazon berechnet für Lambda zum Beispiel die Anzahl der Funktionsaufrufe, die tatsächliche Laufzeit und den benötigten Arbeitsspeicher. Die Rechenleistung, die benötigt wird, um alle Abfragen ohne Verzögerung abzuarbeiten, skaliert automatisch, und zwar bei großen Anbietern auch massiv.

Die drei von der NIST definierten Basisdienste können in jeder der Bereitstellungsvarianten (Public Cloud, Private Cloud usw.) vorkommen. Dort beschreiben sie verschiedene Ebenen beziehungsweise aufsteigende Stufen von Abstraktion. Das Modell der drei Dienste macht deutlich, wie auf Anbieterseite das Mittel, Hardware- und Softwareressourcen immer weiter zu abstrahieren, eingesetzt wird, um innovative und nützliche Angebote für verschiedene Anwendungszwecke hervorzubringen.

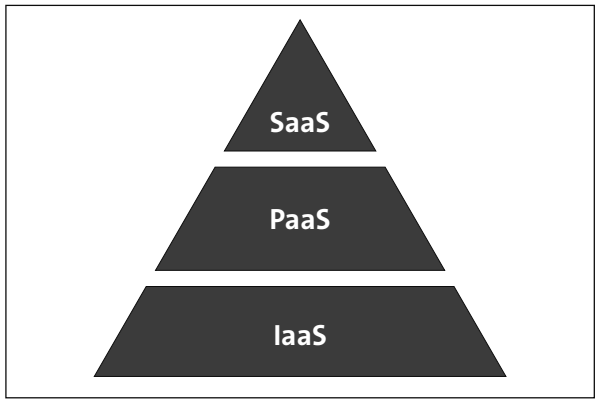


Abbildung 1.2 Die Basis-Services als Pyramide

Abbildung 1.2 zeigt die Basis-Services als eine Pyramide strukturiert. Das soll veranschaulichen, wie der Cloud-Anbieter diese intern selbst einsetzt, um bestimmte Services damit zu konstruieren: Ein SaaS-Dienst baut intern auf eine PaaS auf, die wiederum auf virtualisierter Hardware läuft, die als IaaS konfektioniert ist.

Was sich im Hintergrund eines bestimmten SaaS-Dienstes beim Anbieter technisch abspielt, davon bekommen die Benutzer gar nichts mit. Und sie müssen sich, um den Dienst zu benutzen, ja auch keine eigenen IaaS-Instanzen zulegen. Das soll bei dieser höher abstrahierten Klasse zum Zwecke der Einfachheit ja gerade von Ihnen ferngehalten werden.

Abbildung 1.3 zeigt die Komponenten des gesamten Hard- und Softwarestacks, der für den Betrieb einer Applikation in einer Serviceklasse benötigt wird. Sie sehen, dass bei immer abstrakter werdenden Klassen der Anbieter dem Benutzer die Zuständigkeit für immer mehr Komponenten abnimmt. Bei IaaS müssen die meisten Komponenten vom Benutzer kontrolliert werden, und zwar alle ab dem eingerichteten Server aufwärts.

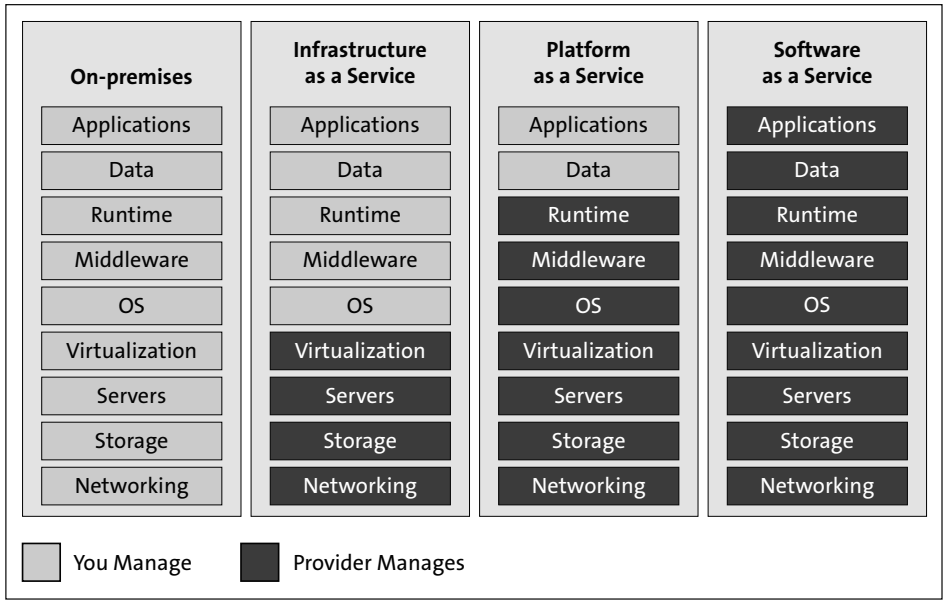


Abbildung 1.3 Wer ist für welche Komponenten bei welchem Service zuständig?

1.1.9 Infrastructure-as-a-Service

Infrastructure-as-a-Service beim Public Cloud Computing ist ein Dienst, der praktisch ein abstrahiertes Rechenzentrum zur Verfügung stellt. Dabei werden reine Hardwareressourcen in Form von virtualisierten Servern angeboten. Die Kontrolle und das Management der unterliegenden physischen Hardware obliegen dabei vollständig dem Anbieter, der sie in eigenen Rechenzentren betreibt. Der Kunde ist für die Auswahl und den Betrieb des Softwarestacks verantwortlich. Er kann auf den gemieteten IaaS-Instanzen ein Betriebssystem seiner Wahl und beliebige Softwarepakete aufspielen und damit dann eigene Applikationen in der Cloud betreiben.

Folgende Grundsätze gelten für alle in diesem Buch besprochenen IaaS-Angebote:

- IaaS ist wie andere Cloud-Dienste auch ein Selbstbedienungsservice, bei dem der Kunde sich bei Bedarf selbstständig benötigte Ressourcen zustellt. Das geschieht über ein Webinterface oder über eine Schnittstelle, auf die händisch mit einem Tool oder automatisiert zugegriffen wird. Eine menschliche Interaktion mit einer

Servicezentrale beim Anbieter ist dabei nicht vorgesehen – und auch gar nicht nötig. Das heißt, Sie sind auf sich allein gestellt, das Angebot zu nutzen. Oder Sie schalten einen spezialisierten Dienstleister ein.

- Die zur Verfügung stehenden Computing-Ressourcen stammen aus einem allgemein zugänglichen Pool, aus dem alle Benutzer ihre Instanzen gleichzeitig beziehen. In diesem Pool werden die zur Verfügung stehenden Ressourcen grundsätzlich unprivilegiert und allen Kunden gleichberechtigt angeboten.
- IaaS steht meistens ohne Vorabüberweisung und allgemein ohne Grundgebühren zur Verfügung. Es wird lediglich eine Kreditkarte verlangt, und abgerechnet wird dabei als Standardmodell immer nur der tatsächliche Verbrauch. Dabei kommen unterschiedliche Parameter wie der Zeitraum der Nutzung, aber auch die Größe des übertragenen und beim Anbieter gespeicherten Datenvolumens zum Tragen. Als Kunde müssen Sie selbst darauf aufpassen, keine Kosten zu erzeugen, die Ihren finanziellen Rahmen sprengen.
- Als IaaS-Instanzen angeboten werden standardisierte Server-Layouts verschiedenen Zuschnitts für allgemeine Anwendungsfälle. Kennwerte für Instanz-Größen sind dabei die Anzahl der Kerne, die Größe des Arbeitsspeichers und die Größe des angehängten Blockspeichers. Provider haben oft aber auch spezielle computing- oder speicheroptimierte Systeme im Angebot, etwa für Deep-Learning-Anwendungen. Instanzen mit GPU-Karten werden angeboten, Supercomputer-Rechenzeit und viele andere Dinge mehr. Mittlerweile gibt es praktisch alles in der Cloud.

Um noch mal auf das Carsharing zurückzukommen: Dabei gibt es natürlich auch Nachteile gegenüber einem eigenen Auto. Die von einem Anbieter in Hamburg als smarter »Airport Transfer« beworbenen Fahrzeuge der kleinsten Kraftfahrzeugklasse haben zum Beispiel gar nicht genug Stauraum für große Koffer. Genauso ist die Frage, ob Infrastruktur aus der Cloud eigene Infrastruktur ersetzen soll beziehungsweise kann, jeweils Abwägungssache und eine Frage des konkreten Anwendungsfalls. Die Cloud ist günstiger, allerdings ist diese Art der Infrastruktur durch die Internet-Anbindung behäbiger. Außerdem erzeugt es zusätzliche stattliche Kosten, wenn ständig große Datenmengen in die oder aus der Cloud geladen werden sollen. Tatsächlich geht der Trend bei größeren Unternehmen zurzeit stark zum hybriden Cloud Computing hin. Bei ihm werden eine eigene Infrastruktur und eine Cloud-Infrastruktur miteinander vermischt, um die Vorteile beider Welten auszuschöpfen. Schauen Sie sich dazu Abschnitt 3.6 an.

1.2 Public Cloud Computing

Die im Rahmen des Cloud Computing angebotenen Dienstleistungen umfassen mittlerweile das gesamte Spektrum der Informationstechnik. Dazu zählen unter anderem

IT-Infrastruktur, Softwareplattformen und Anwendungen. Die Infrastruktur hat aus Computing-Ressourcen eine Alternative zu physisch vorhandener Hardware gemacht, das als Verbrauchsgut genau wie Wasser, Strom, Benzin und Gas zur Verfügung steht. Damit wurde ein massiver Wachstumsmarkt geschaffen, und das Cloud Computing ist nicht zuletzt wegen den Durchbrüchen im Bereich der KI-Forschung nicht mehr aus der IT wegzudenken.

Die Cloud bietet den Kunden Kostenvorteile, Ergebnisverbesserungen und Innovationspotenziale. Mittlerweile wandert ein nicht unerheblicher Teil von IT-Budgets weltweit zu den Cloud-Anbietern, während die größten Unternehmen in diesem Bereich sich immer mehr Marktanteile untereinander aufteilen. Dazu stampfen sie weitere Rechenzentren aus dem Boden und bieten immer neue Produkte in der Cloud an. Sie verdrängen damit zunehmend die jahrelang gewohnten, mittlerweile aber als rückständig und zu unflexibel verpönten IT-Strukturen.

Die Liste der Unternehmen, die maßgebliche Teile ihrer IT in der Cloud unterhalten und ihre Vorteile daraus ziehen, ist lang. Der US-amerikanische Videostreaming-Dienst Netflix zum Beispiel, auf den zu Spitzenzeiten ein nicht unerheblicher Anteil des weltweiten Internetverkehrs zurückgeht, unterhält seine Infrastruktur bei Amazon Web Services.

In Deutschland erscheint jährlich der von *Bitkom Research* im Auftrag des Wirtschaftsprüfungs- und Beratungsunternehmens KPMG erstellte *Cloud-Monitor*. Die Ergebnisse dieser Umfrage zeigen ein repräsentatives Bild der Unternehmen in Deutschland mit 20 und mehr Mitarbeitern. Die bei Abfassung dieses Kapitels aktuelle Ausgabe von 2023 finden Sie unter dieser Webadresse:

https://hub.kpmg.de/de/cloud-monitor-2023

Im Jahr 2023 setzten fast alle befragten deutschen Unternehmen Public Cloud Computing ein (Abschnitt 1.1 des Cloud-Monitors). Mehr als die Hälfte, nämlich 57 %, verfolgen sogar eine »Cloud-First-Strategie«.

1.2.1 Digitalisierung

In Deutschland wird Cloud Computing heutzutage als Wegbereiter und Basis-Technologie für die auf breiter Front angestrebte Digitalisierung beziehungsweise für die digitale Transformation von Unternehmen betrachtet. Dabei fungiert die Cloud nicht nur als ausgelagertes Rechenzentrum, in dem IaaS-Dienste verwendet werden.

Sie lässt sich in der Gesamtheit der angebotenen Services darüber hinaus auch einsetzen, um Geschäftsprozesse zu digitalisieren – und damit zu entschlacken und effizienter zu machen. Zum Beispiel werden mit Techniken wie Datenanalyse und Machine Learning neue Wertschöpfungsreserven erschlossen und neue digitale Geschäfts-

Kapitel 5
Cloud-Instanzen konfektionieren

»Vorsorge ist besser als Nachsorge.«
– Sprichwort

In diesem Kapitel geht es um Werkzeuge, mit denen Sie Cloud-Instanzen konfigurieren können, bevor sie in Betrieb genommen werden. Die beiden hier vorgestellten Werkzeuge, *Hashicorp Packer* und *Cloud-Init*, setzen dabei an verschiedenen Punkten an. Packer dient dazu, das einer Instanz zugrunde liegende Maschinenimage nach den eigenen Bedürfnissen zu gestalten, während Cloud-Init eine neue Instanz direkt beim ersten Hochfahren anhand eines beim Launch mitgegebenen Templates justiert.

Beide Methoden unterscheiden sich von der Maschinenprovisionierung zum Beispiel mit Ansible, das im nächsten Kapitel die Hauptrolle spielen wird. Damit werden bereits gelaunchte Instanzen, die über SSH zugänglich sind, konfiguriert. Ansible können Sie als Einmal-Durchläufer (zu dieser Thematik erfahren Sie in Kapitel 6, »Cloud-Instanzen mit Ansible konfigurieren«, mehr) aber auch für Packer einsetzen, um damit nicht erst die laufende Maschine, sondern das zugrunde liegende Image vorbereitend zu bestücken.

Man kann also sagen, dass Packer und das selbstständig eingesetzte Ansible alternative Werkzeuge für das Deployment einer Geschäftsapplikation sind. Der Funktionsumfang von Packer steht durch die Einbindung von Ansible oder anderer Provisionierer, wozu auch das mächtige Werkzeug des Shellskriptings gehört, aufwendiger konzipierten Werkzeugen in nichts nach. Wenn ein Deployment konzipiert wird, dann spielen eventuell bestimmte IT-Philosophien eine Rolle, zum Beispiel ob man dem Prinzip von unveränderlichen oder veränderlichen Servern den Vorzug gibt. Diese konkurrierenden Konzepte wurden in Abschnitt 1.3.4 bereits thematisiert.

Auf einer pragmatischeren Ebene spielen dann Dinge wie die Abwägung des Zeitaufwands eine Rolle. Braucht die Vorkonfektionierung mehr Zeit beim Deployment oder die Nachkonfektionierung? Wird immer nur eine Maschinenumgebung benötigt, oder braucht man mehrere identische? Auf einer noch anderen Ebene mag aber auch die Geschäftsapplikation ein bestimmtes Verfahren verlangen. Wenn zum Beispiel Services beim Deployment gelauncht sein müssen und sich gegenseitig ansprechen, um sich zu konfigurieren, dann geht es nur mit einer bereits gelaunchten Maschinen-

landschaft – und nicht rein auf der Basis von Maschinenimages, die Packer immer nur individuell bestücken kann.

Sie können alle hier vorgestellten Tools zwar auch kombinieren und ein komplexes Deployment auf sie verteilen, um ihre jeweiligen Stärken zu nutzen. Allerdings sollten Sie dem Prinzip folgen, die Anzahl der eingesetzten Werkzeuge möglichst klein zu halten. Ein komplexes Gewebe von Deployment-Schritten auf mehrere Frameworks zu verteilen, bringt eigene Probleme mit sich, und Sie sollten immer darauf hinarbeiten, möglichst viel mit einem einzigen Werkzeug erledigen zu lassen. Denn es handelt sich bei jeder der hier vorgestellten Lösungen um in sich abgeschlossene und umfassende Systeme, die unabhängig voneinander entwickelt werden. Wenn es notwendig wird, etwas zu verbinden, was nicht vorgesehen ist, dann sind Sie meistens auf sich allein gestellt.

Eine Richtschnur für den Einsatz der hier vorgestellten Werkzeuge aufzusetzen, ist sehr schwierig und drückt wahrscheinlich auch persönliche Vorlieben und unbewusste Workflow-Erfahrungen aus. Cloud-Init eignet sich meiner Meinung nach eher für die Konfiguration des zugrunde liegenden Linux als für das Deployment einer Applikation. Das mögen andere Engineers aber vollkommen anders sehen und dieses Tool sehr erfolgreich für diesen Zweck einsetzen.

Letztendlich entscheidet immer der konkrete Anwendungsfall (und das Format der Deployment-Artefakte) darüber, welche der in diesem Buch vorgestellten Mittel sich mit bestimmten Aspekten dafür eignen, eingesetzt zu werden, und welche eher nicht. Werkzeuge in ausgefallenen Situationen kreativ auch »gegen den Strich gebürstet« einzusetzen – das ist gerade auch die hohe Kunst von DevOps- und Cloud-Engineering. Ihr Ziel sollte das allerdings nicht sein: Sparsamer Umgang mit Ressourcen (Zeit, Geld und Nerven), den Administrationsaufwand möglichst niedrig zu halten, alles zu automatisieren, überflüssige Werkzeuge einzusparen und kein für Mitarbeiter und Nachfolger undurchdringliches Dickicht aufzusetzen (manche verfolgen das ganz bewusst als Prinzip, um sich unverzichtbar zu machen), das sind einige der Leitlinien, die auf jeden Fall umfassend gelten.

5.1 Hashicorp Packer

Packer (*https://www.packer.io/*) ist ein starker Vertreter der hoch innovativen Open-Source-Toolchain für DevOps und Cloud-Computing des Software-Herstellers HashiCorp. Dieses kleine, aber versatile Tool dient der automatisierten und reproduzierbaren Herstellung von Maschinenabbildern und Containern. Mit seinen eingebauten Buildern liefert Packer Artefakte für verschiedene Anwendungsbereiche. Packer kann virtuelle Festplattenimages für eine ganze Reihe von Public-Cloud-Anbietern, für ver-

schiedene Private-Cloud-Lösungen und für eine Reihe von Virtualisierungslösungen generieren:

- Die Private-Cloud-Lösungen sind *Open Stack*, *Apache CloudStack* und *Joyent Triton*.
- Die Virtualisierungslösungen, die Packer mit Images versorgen kann, sind *Hyper-V*, *Parallels* für macOS, *Qemu*, *Vagrant*, *VirtualBox*, *vSphere* und die *Proxmox*-Plattform.
- Im Bereich Softwarecontainer schließlich lässt sich Packer auch für *Docker*, *LXC* und *LXD* einsetzen.

Außerdem können Builder von Drittanbietern in Form von Plug-ins verwendet werden, was Packer universell einsetzbar macht.

Die von Packer vorgefertigten Maschinenabbilder enthalten fertig installierte Betriebssysteme und aufgespielte Applikationen. Je nach eingesetztem Builder kann dieses Tool ganz neue Images *from scratch* erzeugen. Für automatisierte Betriebssysteminstallationen können die Skripting-Fähigkeiten etwa des Debian-Installers (Stichwort: *Preseeding*) oder des Red-Hat-Installers (Stichwort: *Kickstarting*) genutzt werden. Andere Builder, wie etwa diejenigen für AWS EC2, funktionieren so, dass ein bereits vorhandenes Image nach eigenem Bedarf weiter ausgebaut wird und ein neues Image aus diesem Vorgang resultiert.

Für die Provisionierung in beiden Verfahren können Sie von Packer aus gesteuert die Konfigurationsmanager Ansible, Chef, Puppet, Salt und Converge einsetzen. Außerdem können Sie hier für die Bestückung von Maschinenimages auch mit Shellskripten arbeiten, wofür die Linux-Shell, die Windows-Shell und die PowerShell unterstützt werden.

Um ein ganz neues Maschinenabbild zu erzeugen oder ein bereits vorhandenes Image mit weiterer Software zu bestücken, werden von Packer selbstständig Cloud-Instanzen oder lokal virtualisierte Maschinen gestartet. Der Zugang zu den virtuellen Maschinen und Boxen, die während des Baus eingesetzt werden, erfolgt dann über SSH oder WinRM.

Packer ist in Google Go geschrieben, wie es typisch ist für DevOps-Werkzeuge im Allgemeinen und HashiCorp-Tools im Speziellen. Es zeigt sich hier wieder, welchen Vorteil Tools in dieser Sprache haben, nämlich dass alle verwendeten Bibliotheken in einem monolithischen Binary eingebunden sind. Denn Packer wird gegen eine stattliche Anzahl von Bibliotheken und SDKs für alle unterstützten Gegenstände kompiliert, wodurch sich der große Funktionsumfang dieses Tools erklärt.

In Bereich Public Cloud kann Packer mit den APIs der Anbieter AWS, Azure, Alibaba, Google Cloud Platform, DigitalOcean, Linode und Oracle kommunizieren und deren IaaS-Dienste für seine Zwecke einsetzen. Die deutschen Anbieter Hetzner Cloud und 1&1 Ionos gehören ebenfalls dazu, da auch sie SDKs für ihre Cloud-Lösungen in Go-

lang anbieten, auf die HashiCorp aufbauen kann. Der Anwender muss nur das Binary für seine Architektur herunterladen und kann loslegen, ohne dass irgendwelche Abhängigkeiten erfüllt sein müssen. Die eingebundenen Bibliotheken vertreten einen signifikanten Ausschnitt der gegenwärtigen Cloud-Computing-, Container- und Virtualisierungsszene im Open-Source-Bereich. Packer selbst stand bis 2023 unter der *Mozilla Public License 2.0* zur Verfügung. Danach wurde die Lizenz auf die *Business Source License v1.1* geändert, was HashiCorp davor schützt, dass andere Unternehmen in direkte Konkurrenz mit den angebotenen Dienstleistungen gehen. Beachten Sie dazu auch die Hinweise zu Terraform in Abschnitt 4.3, das unter der gleichen Lizenz vertrieben wird.

Die Tools von HashiCorp setzen das Designprinzip der DevOps-Idee prototypisch um, und auch Packer stellt seinem enormen Funktionsumfang eine stark ausgeprägte Einfachheit in der Bedienung gegenüber. Es wird lediglich ein Template benötigt, und dann gibt es einen Startknopf. Vom Benutzer werden dabei so wenige Dinge wie möglich verlangt. Das Template, das Sie für ein Packer-Projekt benötigen, setzen Sie in JSON auf und wenden es dann mit dem Kommandozeilen-Tool `packer` an. Die Templates funktionieren hier genauso wie bei anderen DevOps-Tools auch: Sie wenden darin einen oder mehrere der eingebauten Builder an und tragen die für die Ausführung benötigten Konfigurationsdaten als Optionsfelder ein. Für bestimmte Zwecke wie die Generierung von Zeitstempeln und universelle Identifier (UUID) bietet Packer Funktionen in Form einer eigenen Sprache an. Die Syntax dieser eingebauten Sprache ähnelt dabei derjenigen der Template-Engine *Jinja2* für Python.

5.1.1 amazon-ebs

In dieser Einführung steht ein ganz bestimmter Builder im Mittelpunkt, und zwar `amazon-ebs`. Dieser gehört zu der Klasse von Buildern, mit der Sie neue AMIs (*Amazon Machine Images*) für EC2 (*Elastic Compute Cloud*) von Amazon Web Services erzeugen können. Der IaaS-Dienst dieses Anbieters wurde bereits in Abschnitt 3.1 vorgestellt.

Mit `amazon-ebs` können Sie ein Wurzeldateisystem für EBS-Volumes erzeugen, was die Klasse von Devices ist, die in EC2 am meisten genutzt wird. Der Builder launcht eine neue Instanz mit einer Basis-AMI, führt darauf den Provisionierungsvorgang durch und legt dann ein neues, daraus abgeleitetes AMI in Ihrem AWS-Konto ab. Die Instanz wird nur für den temporären Gebrauch zugestellt und wird danach von Packer wieder terminiert.

Zu diesem Builder gibt es in Packer zwei Varianten in Bezug auf den Erzeugungsvorgang: Der Builder `amazon-chroot` erzeugt ein neues AMI unter Einsatz einer bereits vorhandenen Instanz, während `amazon-ebssurrogate` ein AMI lokal auf Ihrem Arbeits-

rechner erzeugt. Es gibt in Packer darüber hinaus auch noch den Builder `amazon-instance` für AMIs mit Wurzeldateisystemen, die direkt an die Laufwerke einer Instanz angebunden sind (Stichwort: *instance store devices*).

5.1.2 Packer installieren

Packer (aktuelle Version: 1.11.2) ist verfügbar für macOS, FreeBSD und OpenBSD, Linux, Solaris und Windows. Laden Sie das Tool einfach von der Homepage des Anbieters (*https://www.packer.io/downloads.html*) auf Ihren Arbeitsrechner herunter.

Die Befehle für die Ubuntu-Instanz, die Sie dort finden, führen Sie einfach folgendermaßen aus, um den Paketmanager einzurichten und Packer zu laden:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

sudo apt-add-repository "deb [arch=amd64] https://
apt.releases.hashicorp.com $(lsb_release -cs) main"

sudo apt update && sudo apt install packer
Reading package lists... Done
section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  packer
0 upgraded, 1 newly installed, 0 to remove and 19 not upgraded.
Need to get 15.4 MB of archives.
After this operation, 49.7 MB of additional disk space will be used.
Get:1 https://apt.releases.hashicorp.com jammy/main amd64 packer amd64 1.11.2-
1 [15.4 MB]
Fetched 15.4 MB in 1s (20.4 MB/s)
Selecting previously unselected package packer.
(Reading database ... 98086 files and directories currently installed.)
Preparing to unpack .../packer_1.11.2-1_amd64.deb ...
Unpacking packer (1.11.2-1) ...
Setting up packer (1.11.2-1) ...
```

Ist alles richtig eingerichtet, dann können Sie direkt mit Packer loslegen:

```
~$ packer version
Packer v1.11.2
```

Für ein neues Packer-Projekt müssen Sie zunächst ein neues Template aufsetzen. Die Hauptfunktion des CLI-Tools `packer` besteht darin, es mit `packer build` anzuwenden.

5.1.3 Beispielprojekt

Als Beispiel für die Anwendung von Packer soll hier gezeigt werden, wie Sie damit Folgendes anfertigen: eine AMI für AWS EC2 für eine damit gelaunchte Instanz, die einfach nur ihren öffentlichen Hostnamen und ihre öffentliche IPv4-Adresse zurückgibt, wenn Sie sie über das HTTP-Protokoll ansprechen. Ein konkreter Anwendungszweck für ein solches Maschinenimage wäre zum Beispiel, die korrekte Arbeitsweise eines Loadbalancers zu überprüfen. Sie können ein solches Image dann testweise anstelle der Geschäftsapplikation benutzen, um zu überprüfen, von welcher Instanz hinter einem Loadbalancer eine Anfrage beantwortet wird. Oder wenn Sie eine Kette von Anfragen hintereinander abschicken, finden Sie heraus, welche Instanzen vom Loadbalancer bedient werden.

Eine Linux-Installation zu konfigurieren, die als ein solcher einfacher Returnserver arbeitet, ist nicht sonderlich kompliziert. Dafür soll hier das Webapplikations-Framework *Flask* (*https://palletsprojects.com/projects/flask*) für Python zum Einsatz kommen. Beispiele dieser Art setzen oftmals PHP ein, um Webseiten zu generieren, die Informationen über den Server zurückzugeben, von dem sie stammen. Wenn Sie Python aber ohnehin als Werkzeug für das Cloud-Engineering einsetzen, bietet es sich an, praktisch alles in »einem Aufwasch« ohne weitere Sprachen zu erledigen. Sie erfahren also, wie Sie Packer verwenden und mit Python einen kleinen Test- und Antwortserver aufsetzen.

Mit Flask geschriebene Webapps wie `Httpbin` (das wird in Abschnitt 6.7.5 als Beispiel eingesetzt) sind WSGI-Applikationen. Anstatt des WSGI-Servers Gunicorn wie im Ansible-Kapitel soll in diesem Beispiel aber der Webserver Apache verwendet werden, um diese Applikation auszuführen.

Packer wird eingesetzt, um eine neue fertige AMI zu generieren, die mit einem Apache-Webserver ausgestattet ist, der eine WSGI-Webapp betreibt, die den Hostnamen und die IPv4-Adresse der mit diesem Image gelaunchten Instanz zurückgibt.

Die AMI, die in unserem Beispiel als Grundlage für die Bearbeitung mit Packer verwendet wird, hat die ID `ami-01e444924a2233b07`. Es handelt sich um die offiziell von Ubuntu angebotene AMI für Ubuntu 24.04. Und zwar ist das eine AMI für ein EBS-Wurzellaufwerk, für den Prozessor `amd64` und `hvm`-Virtualisierung – also der gewöhnliche Anwendungsfall in EC2. Die auch bei diesem Beispiel verwendete AWS-Region ist `eu-central-1` (Region Frankfurt am Main).

IDs der Releases

Die IDs der offiziellen Ubuntu-AMIs für EC2 können Sie mit diesem Locator des Herstellers recherchieren:

https://cloud-images.ubuntu.com/locator/ec2/

Beachten Sie, dass dieselben AMIs in unterschiedlichen Regionen jeweils eigene IDs haben. Zudem werden diese regelmäßig geupdatet und verändern auch mit jedem Release die ID. Die offiziellen Ubuntu-Images sind darüber hinaus aber unter allen Cloud-Anbietern identisch.

Die Frage ist nun, welche Methode der Provisionierung in diesem Beispiel vorgeführt werden soll. Packer kann auch Ansible einsetzen, das in diesem Buch in Kapitel 6 ausführlich vorgestellt wird. Allerdings ist die Wahl hier auf die Methode der Provisionierung mittels Shellskripting gefallen. Anstatt hier vorwegzunehmen, was später im Ansible-Kapitel noch ausführlich besprochen wird, möchte ich hier die Möglichkeit nutzen, auch diese Variante ausführlich vorzustellen. Shellskripte mit ihren umfangreichen Möglichkeiten eignen sich hervorragend für die Provisionierung von Cloud-Instanzen. Und Packer ist ein Weg, solche Skripte für diesen Zweck zur Ausführung zu bringen. Wenn Sie die Ansible-Einführung in diesem Buch durchgearbeitet haben, dann werden Sie dieses Framework auch mit Packer einsetzen und das Beispielprojekt ohne Schwierigkeiten zu einem Playbook umarbeiten können.

Beide Methoden haben ihre Vorteile. Ansible-Projekte sind aufgeräumter und bieten etwa feste Verzeichnisse für Dateien, die installiert werden sollen. Ein Shellskript hingegen lässt sich wie hier im Beispiel auch so kompakt aufsetzen, dass sich auf dem Zielsystem Code- und Konfigurationsdateien mit ihm installieren lassen.

5.1.4 Template

Das Packer-Template (`serverhello.json`) für das Beispielprojekt sieht so aus:

```
{
    "variables": {
        "aws_access_key": "{{ env `AWS_ACCESS_KEY_ID` }}",
        "aws_secret_key": "{{ env `AWS_SECRET_ACCESS_KEY` }}"
    },
    "builders": [
        {
            "type": "amazon-ebs",
            "access_key": "{{ user `aws_access_key` }}",
            "secret_key": "{{ user `aws_secret_key` }}",
            "region": "eu-central-1",
            "source_ami": "ami-01e444924a2233b07",
            "instance_type": "t2.micro",
            "ssh_username": "ubuntu",
            "ami_name": "serverhello {{ timestamp }}"
        }
    ],
```

```
    "provisioners" : [
        {
            "type" : "shell",
            "script" : "provisioner.bash",
            "execute_command" : "sudo {{ .Path }}"
        }
    ]
}
```

Listing 5.1 Das Template serverhello.json

Ich denke, Sie brauchen nicht lange, um die Logik dieses Konstrukts nachzuvollziehen. Es gibt hier mit `variables`, `builders` und `provisioners` drei Hauptabschnitte, die wir uns in den folgenden Abschnitten genauer ansehen.

variables

Unter `variables` werden hier die Credentials vorbereitet, die Sie bei der Konfiguration eines neuen AWS-Users für den programmgesteuerten Zugriff bekommen haben (siehe Abschnitt 3.1.1), also die Zugriffsschlüssel-ID und der geheime Zugriffsschlüssel. Wie auch `aws-cli` benötigt `packer` beides, um auf Services von AWS zugreifen zu können. Der Transfer dieser Werte in das Template findet hier über Umgebungsvariablen statt, die mit der eingebauten Funktion `env` evaluiert werden. Diese müssen Sie also setzen, bevor Sie `packer` anwenden.

Es ist nicht möglich, die Umgebungsvariablen direkt in `builders` unter `access_key` und `secret_key` auszuwerten, sondern nur innerhalb von `variables`. Das lässt die Struktur von `packer` aus Ordnungsgründen nicht zu.

Der Name der Umgebungsvariablen ist im Grunde beliebig, aber die hier verwendeten Namen sind dieselben, die auch das CLI-Tool `aws` auswertet. Unter `variables` können Sie aber selbstverständlich auch andere Variablen anlegen und nicht nur die Vermittlung der AWS-Credentials aus der Shell-Umgebung heraus regeln, in der Sie `packer` aufrufen.

> **Gut zu wissen**
>
> Wenn Sie `aws` initialisiert haben, dann gibt es auf Ihrem Arbeitsrechner eine Datei *~/.aws/credentials*. Wenn diese vorhanden ist, dann kann der hier verwendete Packer-Builder sie auch auswerten. In diesem Fall können Sie hier auf den `variables`-Block und auch auf die beiden Optionen `access_key` und `secret_key` bei den Optionen für den Builder im Abschnitt `builders` verzichten.

builders

Im Abschnitt `builders` wird der Builder `amazon-ebs` eingesetzt. Sie sehen, dass dessen Eigenschaften hier innerhalb eines JSON-Arrays stehen, was auch den Einsatz von verschiedenen Buildern in ein und demselben Projekt möglich macht oder auch den verschiedenartigen Einsatz desselben Builders.

Als ein mit AWS vertrauter Anwender wissen Sie bereits, welchen Zweck die angegebenen Optionen hier erfüllen. Mit `region` geben Sie die AWS-Region an, in der die neue AMI erzeugt werden soll. Die mit `source_ami` definierte Basis-AMI muss sich in derselben Region befinden. Ein konkreter Einsatzzweck, für den Sie den Builder auch mehrfach hintereinander anwenden, wäre also zum Beispiel, eine bestimmte AMI in mehreren AWS-Regionen zu erzeugen.

Mit `instance_type` geben Sie den Typ für die Instanz an, auf dem der Build-Prozess stattfinden soll. Der kleinste Zuschnitt der T2-Klasse `t2.micro` wird vom *Free-Tier*-Freikontigent abgedeckt. Für dieses Beispiel reicht das vollkommen aus, denn um nur kurz ein Shellskript durchlaufen zu lassen, benötigen Sie normalerweise keine größere Maschine.

Packer benötigt dann den Usernamen für den SSH-Zugang. Bei den Ubuntu-Images ist es der User `ubuntu`, wie es schon oftmals in diesem Buch vorgekommen ist.

Unter `ami_name` geben Sie den Namen an, den die erzeugte AMI hinterher bekommen soll. Da jeder Durchlauf dieses Templates eine neue AMI erzeugt, benötigen Sie einen universellen Identifizierer, der einen eindeutigen Klarnamen für die AMI erzeugt. Die eingebaute Funktion `timestamp` erzeugt dabei einen sekundengenauen Zeitstempel. Dies ist ein Zeitstempel in der sogenannten Unix-Epoche, die in Sekunden nach dem 1. Januar 1970 um 0 Uhr gerechnet wird. Es geht hier also nur darum, einen eindeutigen AMI-Namen zu bekommen.

Eine Alternative für einen eindeutigen AMI-Namen ist die Funktion `uuid`, die eine absolut eindeutige Identifizierung als Namensbestandteil generiert.

provisioners

Unter `provisioners` schließlich setzen Sie ein oder mehrere Konfigurationssysteme ein, die von Packer unterstützt werden. Die im Beispiel verwendete Provisionierungsmethode ist `shell`, in Verbindung mit `script`.

Den Pfad zu einem Skript, das auf der Bauinstanz eingespielt und ausgelöst werden soll, können Sie entweder wie im Beispiel weglassen, oder Sie können einen Pfad relativ zum Template angeben. Das Skript `provisioner.bash` befindet sich im Projektverzeichnis auf derselben Ebene wie `serverhello.json`, also benötigen Sie keine Pfadangabe.

Packer spielt das Skript auf der Instanz in */tmp* ein und löst es dort aus. Allerdings besitzt das Tool auf der Instanz keine Root-Rechte, weil es sich über SSH mit dem

Default-User `ubuntu` anmelden muss. Das Tool `sudo` für Root-Rechte ist in Ubuntu so, wie es aus der Quell-AMI kommt, aber standardmäßig vorhanden und zudem ist es ohne Kennwort verfügbar. Die Option `execute_command` im Beispiel-Template überschreibt die Ausführung des Skripts auf der Bauinstanz dahingehend, dass es mit `sudo` ausgelöst wird. Das verhindert, dass Sie im eingesetzten Shellskript vor jedem dort abgesetzten Befehl, der Root-Rechte benötigt, `sudo` schreiben müssen – und das betrifft die meisten Befehle. Stattdessen lösen Sie das Skript insgesamt damit aus.

Der Block für die Provisionierer im Packer-Template ist ebenfalls ein Array. Also können Sie für ein Projekt auch mehrere der von Packer unterstützten Provisionierer nacheinander einsetzen. Beispielsweise könnten Sie `shell` auch nach oder vor einer Provisionierung mit Ansible oder einer der anderen unterstützten Lösungen einsetzen. Dafür muss es aber gute Gründe geben, denn unterschiedliche Provisionierer sorgen natürlich für mehr Komplexität und damit für mehr Fehlerpotenzial. *Keep it simple* gilt auch hier als Richtlinie.

Die Möglichkeit, Provisionierer zu verketten, können Sie aber auch durchaus kreativ einsetzen, um einen bestimmten Zweck damit zu erreichen. Folgendes Beispiel lädt zunächst mit dem eingebauten Hilfs-Provisionierer `file` eine Datei auf die Instanz in */tmp* hoch. Dazu werden keine Root-Rechte benötigt. Dann wendet es `shell` mit der Option `inline` an, um mittels `sudo` ein Verzeichnis zu erzeugen und dann genauso die Datei dorthin zu kopieren:

```
"provisioners": [
    {
    "type": "file",
    "source": "index.html",
    "destination": "/tmp/index.html"
    },
    {
    "type": "shell",
    "inline": [
        "sudo mkdir /var/www/myhomepage",
        "sudo mv /tmp/index.html /var/www/myhomepage"
    ]
    },
```

5.1.5 provisioner.bash

Das für das Beispielprojekt aufgesetzte Provisionierungsskript gehen wir am besten wieder Stück für Stück durch. Das Skript `provisioner.bash` besteht aus den folgenden Teilen:

Header

```
#!/bin/bash
set -o errexit
```

In der ersten Zeile steht hier so, wie es sich gehört, die sogenannte Shebang-Zeile. Damit geben Sie an, mit welchem Interpreter das Skript ausgeführt werden soll. Im Beispiel ist das die Bash (*Bourne-Again Shell*).

Die zweite Zeile mit `set` ist hier im Buch bereits vorgekommen, und zwar im Kapitel über AWS CloudFormation (siehe Abschnitt 4.1). Sie könnten diesen Shellparameter `errexit` (das Skript bricht bei einem Fehler sofort ab) auch als Option `-e` in der Shebang-Zeile mitgeben. So ist es aber anschaulicher, und der Parameter dokumentiert sich mit seinem Namen selbst. Beachten Sie, dass Sie einzelne Shellparameter mit `set -o` aktivieren und mit `set +o` deaktivieren. (Das ist übrigens eine beliebte Falle bei Linux-Administratorprüfungen.) Die Liste der verfügbaren Shellparameter inklusive deren aktuellen Status können Sie sich mit `set -o` ausgeben lassen.

Bash für Shellskripte

Bei Linux-Distributionen ist es üblich, dass systeminterne Shellskripte nicht mit der Bash laufen, die für User heutzutage meist standardmäßig eingerichtet wird, sondern mit */bin/sh*. Als distributionsübergreifender Standard gilt für Linux, dass sich unter */bin/sh* entweder die Bourne-Shell (der Vorgänger der Bash) befinden muss oder eine dazu kompatible Shell. Auf Ubuntu und anderen von Debian abstammenden Distributionen ist das die Shell Dash.

Der Funktionsumfang der Bash wurde zwar erheblich erweitert, ist aber vollständig abwärtskompatibel zur Bourne-Shell. Der Grund dafür, nicht die Bash als Default-Shell auch für systeminterne Skripte einzusetzen, besteht darin, dass die Dash schneller ist. Bei Shellskripten, die wie im Beispiel zur Provisionierung eingesetzt werden, können Sie ohne Probleme die Bash verwenden und haben dann die gewohnte Umgebung der Shell auf Ihrem Arbeitsrechner. Sie können auf diese Weise dieselben Schleifenkonstruktionen auf der Kommandozeile ausprobieren usw. Eine andere Shell einzusetzen, ist natürlich auch möglich; sie muss natürlich auf dem Zielsystem vorhanden sein.

Probleme kann es geben, wenn Sie Bash-Code schreiben, aber */bin/sh* in der Shebang-Zeile angeben. Das kommt bei vielen Beispielen im Internet vor, die die Shells nicht sauber trennen. Wenn Sie aus einem bestimmten Grund aber */bin/sh* einsetzen wollen oder müssen (wenn zum Beispiel keine Bash vorhanden ist), dann können Sie ein Skript mit dem Tool *checkbashisms* (auf Ubuntu im Paket *devscripts* verfügbar) auf Inkompatibilitäten hin abklopfen.

Pakete einspielen

```
apt-get update
apt-get install -y apache2 libapache2-mod-wsgi-py3 python3-flask
```

Anschließend wird mit `apt-get update` zunächst die intern gehaltene Liste der verfügbaren Linux-Pakete aktualisiert. Dann installiert derselbe Befehl mit `install` die drei benötigten Pakete: den Apache-Webserver, die für Flask benötigte Erweiterung für WSGI-Applikationen und Flask selbst.

Der Schalter `-y` ist notwendig für den Einsatz im Skript, damit der Befehl durchläuft, ohne nachzufragen, ob auch wirklich installiert werden soll.

> **apt oder apt-get?**
>
> Die einzelnen APT-Tools wurden für die Anwender mittlerweile im Befehl `apt` zusammengefasst. Das Entwicklerteam gibt allerdings selbst den Hinweis aus, dass diese praktische Zusammenfassung noch nicht zuverlässig in Shellskripten funktioniert und daher mit Bedacht eingesetzt werden sollte. Das ältere `apt-get` ist zwar etwas umständlicher, funktioniert aber genauso gut.

serverhello.py erzeugen

Das Shellskript für die Provisionierung fährt folgendermaßen fort:

```
cat > /usr/local/lib/python3.12/dist-packages/serverhello.py << 'EOF'
from flask import Flask
from urllib.parse import urljoin
import requests

app = Flask(__name__)

# Metadata URL and token endpoint for IMDSv2
metadata_url = "http://169.254.169.254/latest/meta-data/"
token_url = "http://169.254.169.254/latest/api/token"

# Function to get the token for IMDSv2
def get_token():
    headers = {'X-aws-ec2-metadata-token-ttl-seconds': '21600'}
    token_response = requests.put(token_url, headers=headers)
    return token_response.text

# Function to get metadata using the token
def get_metadata(path, token):
    headers = {'X-aws-ec2-metadata-token': token}
```

```
    response = requests.get(urljoin(metadata_url, path), headers=headers)
    return response.text

@app.route("/")
def serverhello():
    token = get_token()

    ipv4 = get_metadata("public-ipv4", token)
    hostname = get_metadata("public-hostname", token)

    return "Hello from " + hostname + " (" + ipv4 + ")\n"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=80)
EOF
```

Listing 5.2 Die Flask-Anwendung für unser Beispiel

Diese Konstruktion legt die Flask-Applikation als Quellcodedatei auf der Bauinstanz an. Sie nutzt dafür den Trick, dass Sie mit `cat > /path/to/foo <<'EOF'` von einem Shellskript aus Text in einer Datei ablegen können. Dabei wird die Datei neu erstellt; oder Sie nutzen `cat >>`, um die Inhalte an eine bestehende Datei anzuhängen. Der gesamte nachfolgende Code bis zum Marker `EOF` (der Name ist übrigens beliebig) wird dadurch in der Datei */usr/local/lib/python3.12/dist-packages/serverhello.py* abgelegt. Der Ablageort ist einer der Pfade, aus denen der Python3-Interpreter in dieser Distribution per Default importieren kann:

```
ubuntu@ip-172-31-33-155:~$ python3 -c 'import sys; print(sys.path)'
['', '/usr/lib/python312.zip', '/usr/lib/python3.12',
'/usr/lib/python3.12/lib-dynload', '/usr/local/lib/python3.12/dist-packages',
'/usr/lib/python3/dist-packages']
```

Python-Version

In Ubuntu 24.04 ist Python in der Version 3.12 vorinstalliert. In neueren oder älteren Versionen von Ubuntu sind es andere Versionen, und dadurch verändert sich auch der Pfad. Packer wird dann spätestens beim Ausführen einen Fehler verursachen.

Sind Sie sich nicht sicher, welche Version installiert ist, dann starten Sie einfach eine EC2-Maschine mit dem entsprechenden Image, und schauen Sie nach.

Kommen wir nun zur Flask-Applikation selbst. Eine rudimentäre Webapp mit Flask, die lediglich »Hello world!« zurückgibt, besteht aus den folgenden wenigen Komponenten:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello world!"
```

Diese Bestandteile finden Sie in *helloserver.py* als konstituierende Elemente einer Webapplikation mit Flask wieder.

Die Webapp soll zwei Informationen zurückgeben: erstens (als Returnserver) mit `hostname` den Hostnamen der Instanz, die mit der resultierenden AMI gelauncht wird, und zweitens mit `ipv4` die öffentliche IPv4-Adresse. Hierbei gibt es aber ein Problem.

In Python können Sie den Hostnamen und die IPv4-Adresse des darunterliegenden Servers mit der Bibliothek *socket* ermitteln. Für die IPv4-Adresse benötigen Sie dafür einen kleinen Workaround, damit nicht `127.0.0.1` vom Loopback-Device ausgegeben wird. Aber das funktioniert einwandfrei. Das Problem ist nur, dass dabei auf einer EC2-Instanz die Werte für den *internen* Hostnamen und die *interne* IPv4-Adresse in der Cloud zurückgegeben werden. Das kann durchaus erwünscht sein. Das Ziel des Beispiels ist es aber, die *öffentlichen* Varianten auszugeben. Die öffentliche IP-Adresse etwa ist in der Netzwerkkonfiguration auf einer EC2-Instanz aber gar nicht eingestellt:

```
~$ ssh -i ~/.ssh/aws-key ubuntu@3.124.183.136

ubuntu@ip-172-31-33-155:~$ ip a s eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc fq_codel
          state UP group default qlen 1000
    link/ether 06:81:3f:17:b3:7e brd ff:ff:ff:ff:ff:ff
    inet 172.31.33.155/20 brd 172.31.47.255 scope global dynamic eth0
       valid_lft 2160sec preferred_lft 2160sec
    inet6 fe80::481:3fff:fe17:b37e/64 scope link
       valid_lft forever preferred_lft forever
```

Wie also vorgehen, um das Ziel zu erreichen? Der öffentliche Hostname und die öffentliche IPv4-Adresse sind aus den Metadaten verfügbar, die zu einer Instanz gehören. Und die können Sie innerhalb von AWS über eine bestimmte URL abfragen. Unsere Beispielapplikation benutzt für die Abfragen die Python-Bibliothek *Requests* (*https://requests.readthedocs.io/*), die auf Ubuntu mit zur Grundausstattung gehört. Außerdem brauchen Sie noch `urllib.parse.join()`, um zwei verschiedene Endpunkte zusammenzusetzen.

Metadatenserver

Die für eine EC2-Instanz gültigen Metadaten können Sie über die IP-Adresse `169.254.169.254` abrufen, die nur von der Instanz aus erreichbar ist. Es stehen hier (je nach Metadaten-Version) folgende Endpunkte mit Informationen über die Instanz zur Verfügung, von der aus sie abgefragt werden:

```
~$ ubuntu@ip-172-31-33-155:~$ curl http://169.254.169.254/2009-04-04/meta-
data/
ami-id    ami-launch-index    ami-manifest-path
block-device-mapping/  hostname    instance-action
instance-id    instance-type    local-hostname
local-ipv4    placement/    profile
public-hostname    public-ipv4    public-keys/
reservation-id    security-groups
~$ curl http://169.254.169.254/2009-04-04/meta-data/public-ipv4
3.124.183.136
```

Andere IaaS-Anbieter betreiben Metadatenserver genau nach demselben Prinzip und unter derselben IP-Adresse.

Gut zu wissen

In der neuen Version der Metadata-API von AWS wird eine Authentifizierung notwendig. Diese soll vor unberechtigten Zugriffen schützen. In der Flask-App findet das Abrufen des Tokens an folgender Stelle statt:

```
def get_token():
    headers = {'X-aws-ec2-metadata-token-ttl-seconds': '21600'}
    token_response = requests.put(token_url, headers=headers)
    return token_response.text
```

Das zurückgelieferte Token wird dann mit jedem API-Aufruf mitgegeben.

Apache-Webserver

Der dritte Abschnitt des Shellskripts provisioniert Apache dafür, die Flask-Applikation *serverhello.py* bereitzustellen.

```
echo "from serverhello import app as application" > \
   /var/www/serverhello.wsgi
```

Mit Flask geschriebene Apps sind WSGI-(*Web Server Gateway Interface*-)Applikationen, für deren Anwendung mit der WSGI-Erweiterung von Apache (*mod_wsgi*) zu-

nächst ein Starterksript benötigt wird. Dieses enthält nur eine Python-Zeile mit einem bestimmten Import-Statement. Sie können diese Datei einfach mit `echo` wieder aus dem Provisionierungsskript heraus erzeugen, wie in diesem Beispiel.

```
useradd -s /bin/false wsgiuser
```

Danach wird mit `useradd` ein neuer, rein systeminterner Linux-User angelegt. Dieser wird für *mod_wsgi* benötigt, um die Applikation nicht als `root` laufen zu lassen.

```
cat > /etc/apache2/sites-available/serverhello.conf << 'EOF'
<VirtualHost *:80>
 WSGIDaemonProcess serverhello user=wsgiuser group=wsgiuser threads=2
 WSGIScriptAlias / /var/www/serverhello.wsgi
</VirtualHost>
EOF
```

Dann wird wieder mit der Konstruktion `cat > foo << 'EOF'` die für den Betrieb der WSGI-Applikation benötigte Konfigurationsdatei für Apache 2 erzeugt. Die Applikation läuft damit hinterher auf dem HTTP-Port 80.

```
a2dissite 000-default.conf
a2ensite serverhello.conf
```

Schließlich wird die Standardbegrüßungsseite des Apache-Webservers mit dem Hilfsprogramm `a2dissite` deaktiviert. Danach müssen Sie die *serverhello*-Applikation mit `a2ensite` aktivieren. Die Webapp ist danach noch nicht in Betrieb, weil der systemd-Service `apache2.service` zunächst erst einmal neu gestartet oder zumindest neu geladen werden müsste. Bei dieser Provisionierungsmethode spielt das aber, wie gesagt, keine Rolle, und deshalb gibt es dafür keinen Befehl mehr in diesem Skript.

5.1.6 Anwendung

Das Projekt ist damit so weit, dass Sie es mit `packer` anwenden können. Sie können das Template zunächst einmal mit `packer validate serverhello.json` vorab testen. Exportieren Sie dann funktionierende AWS-Credentials, die Zugriff auf AWS EC2 haben, in die vom Template abgefragten Umgebungsvariablen. Lösen Sie Packer dann aus:

```
~$ export AWS_ACCESS_KEY_ID=AKIAR4Z44I4N65US3VLO

~$ export AWS_SECRET_ACCESS_KEY=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

~$ packer build serverhello.json
```

Das beim Durchlauf ausgegebene Log enthält auch die Rückgaben der Befehle, die durch das Shellskript abgesetzt wurden. Reduziert auf die reinen Meldungen, die von

Packer stammen, sieht das Log einer Anwendung unseres Beispielprojekts aber so aus:

```
==> amazon-ebs: Prevalidating any provided VPC information
==> amazon-ebs: Prevalidating AMI Name: serverhello 1579073525
    amazon-ebs: Found Image ID: ami-01e444924a2233b07
==> amazon-ebs: Creating temporary keypair: packer_5e1ebff5-72e2-80c1-
                9ad0-3b40bb62607e
==> amazon-ebs: Creating temporary security group for this instance:
                packer_5e1ebff6-750f-7b83-f184-3c2d5624ef61
==> amazon-ebs: Authorizing access to port 22 from [0.0.0.0/0] in
                the temporary security groups...
==> amazon-ebs: Launching a source AWS instance...
==> amazon-ebs: Adding tags to source instance
    amazon-ebs: Adding tag: "Name": "Packer Builder"
    amazon-ebs: Instance ID: i-0f29dbcb147f33626
==> amazon-ebs: Waiting for instance (i-0f29dbcb147f33626)
                to become ready...
==> amazon-ebs: Using ssh communicator to connect: 18.185.72.51
==> amazon-ebs: Waiting for SSH to become available...
==> amazon-ebs: Connected to SSH!
==> amazon-ebs: Provisioning with shell script:
                provisioner.bash
{...}
==> amazon-ebs: Stopping the source instance...
    amazon-ebs: Stopping instance
==> amazon-ebs: Waiting for the instance to stop...
==> amazon-ebs: Creating AMI serverhello 1579073525 from
                instance i-0f29dbcb147f33626
    amazon-ebs: AMI: ami-0d451d358a689448a
==> amazon-ebs: Waiting for AMI to become ready...
==> amazon-ebs: Terminating the source AWS instance...
==> amazon-ebs: Cleaning up any extra volumes...
==> amazon-ebs: No volumes to clean up, skipping
==> amazon-ebs: Deleting temporary security group...
==> amazon-ebs: Deleting temporary keypair...
Build 'amazon-ebs' finished.
==> Builds finished. The artifacts of successful builds are:
--> amazon-ebs: AMIs were created: eu-central-1:
                ami-0d451d358a689448a
```

Sie können hier sehr schön sehen, welche Schritte Packer unternimmt, um die neue AMI zu erzeugen:

- Das Tool generiert zunächst ein SSH-Schlüsselpaar und dann eine EC2-Sicherheitsgruppe mit dem freigeschalteten Zugang für den SSH-Port 22. Packer kann sich ja nicht darauf verlassen, dass dieser Ingress bei der Default-Sicherheitsgruppe zur Verfügung steht (er ist per Default aktiviert, kann aber geschlossen werden).
- Dann wird die Bauinstanz gelauncht und gewartet, bis der SSH-Zugang darauf verfügbar ist. Packer verbindet sich mit der Instanz, spielt das Provisionierungsskript ein und löst es aus.
- Nach dem Abschluss des Deployments wird die Instanz gestoppt und ein neues AMI daraus erzeugt. Die Instanz wird dann terminiert, und zum Schluss werden das SSH-Schlüsselpaar und die Sicherheitsgruppe wieder gelöscht.

Der gesamte Vorgang dauert etwa 2 bis 3 Minuten. Die neue AMI ist danach in der Region, in der sie angelegt worden ist, für Sie verfügbar. Sie ist zunächst als privat gekennzeichnet, Sie können sie aber auch veröffentlichen. Sie können die von Ihnen erzeugten AMIs mit folgendem Befehl abfragen:

```
$ aws ec2 describe-images --owners 130587313947
{
 "Images": [
     {
         "Architecture": "x86_64",
         "CreationDate": "2020-01-15T07:33:57.000Z",
         "ImageId": "ami-0d451d358a689448a",
         "ImageLocation": "130587313947/serverhello 1579073525",
         "ImageType": "machine",
         "Public": false,
         "OwnerId": "130587313947",
         "State": "available",
```

Die 12-stellige ID Ihres AWS-Kontos finden Sie auf der Webkonsole oben im Dashboard vom Identity and Access-Management (siehe Abschnitt 3.1.1).

AMI ausprobieren

Wenden Sie die AMI nun einfach an, indem Sie eine neue Instanz mit ihr launchen. Warten Sie dann einen Moment, bis sie komplett hochgefahren ist und der Webserver auf ihr Anfragen entgegennimmt. Sprechen Sie die Instanz über das HTTP-Protokoll an:

```
~$ aws ec2 run-instances --instance-type t2.micro --key-name myawskey \
   --image-id ami-0d451d358a689448a

~$ aws ec2 describe-instances --instance-ids i-005d03b6cf148a609 \
   jq
18.195.76.187
```

```
~$ curl -i 18.195.76.187
HTTP/1.1 200 OK
Date: Wed, 20 Nov 2024 08:08:19 GMT
Server: Apache/2.4.38 (Ubuntu)
Content-Length: 80
Vary: Accept-Encoding
Content-Type: text/html; charset=utf-8

Hello from ec2-18-195-76-187.eu-central-
1.compute.amazonaws.com  (18.195.76.187)
```

Das funktioniert also! JSON-Objekte zurückzugeben anstatt unformatierten HTML-Texts ist mit Flask natürlich auch ohne Weiteres möglich. Dafür steht die Funktion `jsonify` zur Verfügung.

Hinweis

Beachten Sie beim Ausprobieren des Beispiels, dass die zugeteilte Sicherheitsgruppe (wenn Sie nichts anderes angeben, dann ist das Ihre Default-SG) für den Zugriff auf Port 80 freigeschaltet sein muss!

Nicht mehr benötigte AMIs werden nicht gelöscht, sondern deregistriert. Das erreichen Sie mit dem Befehl `aws ec2 deregister-image`.

5.2 Cloud-Init

Cloud-Init (*https://cloud-init.io/*) kommt aus dem Hause Canonical, dem Unternehmen, das hinter der Linux-Distribution Ubuntu steht. Auch wenn die Software meiner Erfahrung nach nicht allzu bekannt ist, handelt es sich dabei um den De-facto-Standard für die automatische Konfiguration von Linux-Systemen in Cloud-Umgebungen.

Als Initialisierungssystem von Cloud-Instanzen in IaaS-Diensten ist Cloud-Init weit verbreitet. Es wird mittlerweile von Cloud-Anbietern weltweit eingesetzt, die dafür von Cloud-Init unterstützt werden müssen. Die Software ist distributionsübergreifend und funktioniert außer unter Ubuntu auch noch auf dem Suse Linux Enterprise Server (SLES) und openSUSE, Red Hat Enterprise Linux und centOS, Fedora, Debian, Gentoo und Arch Linux sowie FreeBSD.

Cloud-Init ist das Bindeglied zwischen IaaS-Services und Linux-Distributionen. Es ist in Python geschrieben und doppelt lizenziert, d. h., es steht gleichzeitig unter der GNU Public License Version 3 und der Apache License Version 2.0.

Cloud-Init ist clever gemacht, ausgereift (aktuell Version 24.2) und besitzt einen stattlichen Funktionsumfang. Das Konfigurationssystem ist mehrstufig ausgebaut und klinkt sich mittels systemd-Targets in den Bootvorgang einer Linux-Distribution ein. Zunächst erkennt es automatisch, in welcher Cloud-Umgebung sich der Server befindet, auf dem es läuft. Dann bezieht es einen Grundstock von Konfigurationsdaten aus den Metadaten-Services, die die IaaS-Anbieter intern betreiben. Dazu gehören der Hostname und der SSH-Schlüssel, der auf die Instanz eingespielt wird, um über SSH darauf zugreifen zu können. Es stellt die Netzwerkkonfiguration ein und nimmt die Lokalisierung der Cloud-Instanz vor. Darüber hinaus lassen sich Konfigurationstemplates mitgeben, um das Linux beim ersten Booten noch weiter zu konfektionieren. Diese Templates werden, wenn Sie vom Cloud-Anbieter selbst kommen, als *vendor data* bezeichnet, und als *user data*, wenn Sie von Ihnen als IaaS-Benutzer stammen. Geeignete Template-Dateien können Sie bei Anbietern, die Cloud-Init einsetzen, beim Launchen einer neuen Linux-Instanz einfach mitgeben. Diese werden dann automatisch verarbeitet.

Cloud-Init nimmt seine Einstellungen beim ersten Hochfahren einer Instanz vor, also im Zuge ihrer Einrichtung. Es schließt seine Anpassungen ab, ohne dass das Linux-System neu gebootet werden muss. Das Konfigurationssystem steht damit zwischen der Methode der Konfektionierung von Images für IaaS-Instanzen mit Packer und der Methode der Provisionierung von bereits fertig hochgefahrenen und laufenden Instanzen mit Ansible (oder alternativer Software).

Cloud-Init verfügt über eine Reihe von eingebauten Modulen, mit denen bestimmte Konfigurationsmaßnahmen vorgenommen werden können. Ein maßgeblicher Einsatzzweck ist aber auch die Ausführung von universellen Shellbefehlen. Diese können kommandoweise abgesetzt werden, Cloud-Init kann aber auch ganze Shellskripte verarbeiten. Die ausführliche Dokumentation zu dieser Software finden Sie unter folgender Adresse:

https://readthedocs.org/projects/cloudinit/downloads/pdf/latest/

Ein typischer Anwendungsbereich von Cloud-Init sind zum Beispiel die Konfiguration von Paketquellen und Sicherheitseinstellungen. Wenn Sie den bei manchen Anbietern per Default voreingestellten Root-Zugang über SSH umkonfigurieren möchten, dann würden Sie nicht Ansible dafür einsetzen. Denn dann müssten Sie einmal als Root auf die neu gelaunchte Instanz zugreifen, den Zugang deaktivieren, und anschließend die restliche Provisionierung separat mittels `become` vornehmen. Dafür bräuchten Sie zwei Projekte, die Sie strikt nacheinander anwenden müssen. Das entspricht aber nicht der Anwendungsweise von Ansible, die grundsätzlich die gesamte Provisionierung in einem zusammenhängenden Projekt zum Ziel hat. Außerdem sind bei der Methode der nachträglichen Konfiguration die Cloud-Instanzen mit SSH-Zugang für `root` eine Zeit lang im Netz. Das ist bei Produktivsystemen aus Sicherheitsgründen aber überhaupt nicht erwünscht.

5.2.1 Beispielprojekt

Für ein kleines Beispielprojekt zum Einsatz von Userdaten mit Cloud-Init wählen wir folgende Aufgabenstellung: Auf einer gelaunchten Cloud-Instanz wollen wir einen neuen Default-User einrichten und den Root-Zugang über SSH deaktivieren. Die Konfigurationsvorgaben werden dann beim Launchen einer neuen Instanz bei DigitalOcean (siehe Abschnitt 3.4) und bei der Hetzner Cloud (siehe Abschnitt 3.5) eingespielt und verarbeitet. (Beide Anbieter sind per Default über SSH als `root` erreichbar.)

Das Ziel wird hier nicht durch den Einsatz eines Shellskripts erreicht, sondern mit in Cloud-Init eingebauten Modulen. (Einige von ihnen setzen allerdings einzelne Shellbefehle ab.) Diese Module wenden Sie mit einem Template an, das in YAML zu schreiben ist.

Viele der eingebauten Module von Cloud-Init sind Betriebssystem-agnostisch. Es spielt bei ihnen also keine Rolle, für welche Linux-Distribution Sie ein Template einsetzen bzw. in welchem Sie es verwenden. Wenn Sie Shellbefehle absetzen, dann müssen Sie nur genau hinsehen, ob die angesprochenen Konfigurationsschalter in dieser Form auch tatsächlich auf der konkreten Instanz mit der dort jeweils laufenden Distribution in der angenommenen Form zur Verfügung stehen. Das Beispiel, das Sie hier durchspielen, lässt sich mit seinen Shellbefehlen auf verschiedenen Distributionen anwenden.

Template

Das YAML-Template *preconf.yaml* als Beispielanwendung für Cloud-Init hat folgende Form:

```
#cloud-config
users:
  - name: myuser
    ssh-authorized-keys:
      - "ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAII17gMseS+cRPFenhdb6cv1CmfOMq
++VN4ZOjWhZ+q9h my@user.com"
    sudo: "ALL=(ALL) NOPASSWD:ALL"
    shell: /bin/bash

runcmd:
  - sed -i '/^PermitRootLogin/s/^.*$/PermitRootLogin no/' /etc/ssh/sshd_config
  - sed -i '/#PasswordAuthentication/s/^.*$/PasswordAuthentication no/' /etc/
ssh/sshd_config
  - sed -i '$a AllowUsers myuser' /etc/ssh/sshd_config
  - systemctl restart sshd.service
```

Gehen wir die hier vorkommenden Elemente der Reihe nach durch:

Die Kopfzeile `#cloud-config` kennzeichnet diese Textdatei als Konfigurations-Template für Cloud-Init. Das Konfigurationssystem kann eine ganze Reihe von Formaten als Userdaten verarbeiten. Eine Shebang-Zeile mit `#!` an dieser Stelle würde zum Beispiel ein Shellskript anzeigen, das Cloud-Init alternativ verarbeiten kann.

users

Das erste Modul von Cloud-Init, das hier eingesetzt wird, ist *users*. Damit können Sie auf dem hochfahrenden System einen neuen Linux-User einrichten. Folgende Optionen für *users* kommen in diesem Anwendungsbeispiel zum Einsatz:

- Mit `ssh-authorized-keys` spielen Sie einen öffentlichen SSH-Schlüssel für den neuen User ein. Generieren Sie dafür ein neues SSH-Schlüsselpaar mit dem dazugehörigen öffentlichen Teil wie im Beispiel-Template mit dem Befehl `ssh-keygen -t ed25519 -f myuserkey -C "my@user.com" -P ""`. Der Schalter `-P ""` stellt eine leere Passphrase ein, was für Produktivsysteme natürlich nicht empfohlen, für ein kleines Beispiel aber akzeptabel ist. Sie finden den Schlüssel dann hinterher in der Datei */home/myuser/.ssh/authorized_keys* auf der Instanz.

  Die Metadaten der IaaS-Dienste der beiden Zielanbieter transportieren lediglich den SSH-Schlüssel für den Default-User `root`, sodass Sie den Schlüssel für den neuen User auf diesem Wege mitgeben müssen. Den Key auch über die SSH-Schlüsselverwaltung im jeweiligen IaaS-Dienst laufen zu lassen und ihn daraus über noch einen anderen Weg als über den Metadatenserver zu beziehen, ist mit Cloud-Init nicht möglich. Aber grundsätzlich gibt es an dieser Methode nichts auszusetzen.

  SSH-Schlüssel im Format *ed25519* haben gegenüber RSA kürzere Schlüsseltexte. Das ist praktisch, wenn Sie SSH-Schlüssel als Text und nicht als Datei übertragen müssen, wie hier im Beispiel.

- Mit der Option `sudo` richten Sie den neuen User dafür ein, temporäre Superuser-Rechte zu bekommen. Das ist notwendig, um über SSH überhaupt noch administrative Maßnahmen auf der Instanz vornehmen zu können und etwa auch noch Ansible mit seinem `become`-Mechanismus auf ihr einsetzen zu können. Zugriff auf `sudo` für den Default-User kann vielleicht aber auch explizit nicht erwünscht sein. Sie können den *root*-User auf einem Linux-System nicht löschen, aber Sie können als Sicherheitsmaßnahme dafür sorgen, dass mit Bordmitteln zumindest niemand die Root-Rolle annehmen kann. Die hier vorgenommene Einstellung für `sudo` legt aber fest, dass *myuser* den Befehl ohne Kennwortabfrage nutzen kann. Das ist notwendig, weil für diesen neuen Default-User überhaupt kein Kennwort eingerichtet wird.

- Mit der Option `shell` legen Sie die Shell für den neuen User fest. Das ist hier die Linux-Default-Shell für Benutzer */bin/bash*. Fehlt diese Option, dann stellt Cloud-Init */bin/sh* ein.

runcmd

Das nächste eingesetzte Modul von Cloud-Init ist `runcmd`. Es dient dazu, um Shellbefehle auf der Instanz als Konfigurationsmittel abzusetzen. Die Befehle werden mit der Shell */bin/sh* ausgeführt.

Drei der vier hier abgesetzten Befehle benutzen das Linux-Texttool `sed`, um die Konfigurationsdatei */etc/ssh/sshd_config* vom SSH-Server in jeweils einem separaten Schritt zu bearbeiten. `sed` beherrscht reguläre Ausdrücke, was es in Linux zu einem Standardwerkzeug für das skriptgesteuerte Suchen und Ersetzen von Textzeilen macht. Die Option `-i` (`--inplace`) legt fest, dass die jeweilige Operation in der Datei vollzogen werden soll und nicht nur die veränderte Fassung auf `STDOUT` als Ergebnis ausgegeben wird.

Folgende drei Operationen an *sshd_config* finden hier mit `sed` statt:

- Der Ausdruck bei der ersten Anwendung sucht zunächst die Zeile, in der der String `PermitRootLogin` am Anfang der Zeile (`^`) steht. Diese Eingrenzung findet hier statt, weil der Text in dieser Datei auch noch an anderer Stelle im darin eingebetteten Dokumentationstext vorkommt. Dort würde er sonst auch ersetzt werden. Es macht grundsätzlich keine Probleme, wenn zweimal dieselbe Zeile in dieser Konfigurationsdatei vorkommt, nur ganz sauber ist es so nicht. Das Tool ersetzt (`s`) dann die gesamte Zeile (`^.*$`) durch den String `PermitRootLogin no`. Ein Root-Login über SSH ist mit dieser Einstellung nicht mehr möglich.
- Die zweite Anwendung von `sed` sucht die auskommentierte Zeile (`#`) mit dem Ausdruck `PasswordAuthentication` und stellt diesen Konfigurationsschalter aktiv und auf `no`. Auch wenn die User auf diesem System überhaupt keine Kennwörter eingestellt haben, bewirkt die explizite Einstellung dieses Schalters ein etwas saubereres Verhalten vom SSH-Server und die unmittelbare Ablehnung von gesperrten Usern.
- Die dritte Anwendung von `sed` fügt der Datei am Ende (`$a`) den Ausdruck `AllowUsers myuser` hinzu. Obwohl dem User `root` bereits der Zugang verwehrt worden ist, macht das als zusätzliches Hardening-Merkmal hier durchaus Sinn.

Als viertes abgesetztes Kommando wird mit `systemctl` der SSH-Server neu gestartet. Das muss im Gegensatz zu Packer, dessen Anwendung immer mit einem Shutdown abgeschlossen wird, hier ausdrücklich stattfinden. Die Konfigurationsmaßnahmen am SSH-Server bleiben sonst ohne Wirkung, bis der nächste Reboot stattfindet.

die Daten visualisieren können. Wenn Sie sich die GUI von Prometheus ansehen, werden Sie die API wiedererkennen.

Die GUI von Prometheus fühlt sich manchmal etwas rudimentär an, dadurch ist es gut, dass Sie jetzt schon das Steuern über die Kommandozeile üben. Trotzdem empfehle ich Ihnen, sich mal die GUI anzusehen und ein Gefühl dafür zu bekommen.

8.3 Service Discovery

Prometheus bleibt mit der statischen Konfiguration von Targets für das Scraping weit unter seinen Möglichkeiten. Genauso wie bei dem statischen Inventar von Ansible (siehe Abschnitt 6.4) passt diese Methode nicht gut zum Cloud Computing. Es handelt sich dabei um dynamische Infrastruktur, in der Instanzen kommen und gehen und bei der eine feste Liste mit den IP-Adressen von Servern keinen Sinn mehr macht. Prometheus verfügt aus diesem Grund über eine Reihe von eingebauten Service-Discovery-Mechanismen für bestimmte Umgebungen. Die können Sie einsetzen, um Prometheus darin selbstständig nach Knoten suchen zu lassen, die Exporter betreiben und als Targets eingebunden werden können.

Sie können Prometheus so konfigurieren, dass er in Azure, AWS EC2 und Google Compute Engine nach geeigneten Targets recherchiert. Darüber hinaus kann Prometheus auch die Service-Discovery-Applikationen Consul, Nerve von Air B'n'B und die Serversets-Bibliothek für Golang ansprechen, um daraus Listen mit relevanten Targets zu erstellen. Prometheus kann sich auch noch in anderen dynamischen Umgebungen selbstständig orientieren und Targets in OpenStack und innerhalb eines Kubernetes-Clusters finden.

Consul

Consul von Hashicorp (*https://www.consul.io/*) ist eine universelle, verteilte Lösung für die Verbindung von Services und für die automatische Konfiguration von Applikationen in dynamischer Infrastruktur. Es ist ebenfalls in Google Go implementiert und steht unter der Business Source License 1.1.

Derselbe Consul-Agent kann in verschiedenen Modi als Server oder Client betrieben werden. Mehrere Instanzen mit einem darauf betriebenen Consul in einem bestimmten Netzwerk (sinnvollerweise ein privates Netzwerk bei einem IaaS-Dienst) formen automatisch einen Cluster.

Eine der Hauptfunktionen von Consul ist die automatische Dienstauffindung. Irgendwo im Cluster vorhandene Services registrieren sich in Consul und können dann zentral über das eingebaute DNS- oder HTTP-Interface auf dem Server gefunden und angesprochen werden.

Consul nimmt darüber hinaus eigene Health Checks vor und prüft selbstständig, ob angemeldete Services auch funktionieren. Das verhindert ein Ansprechen von irgendwann eventuell nicht mehr vorhandenen Knoten.

Ein weiteres starkes Feature ist der verteilte Key/Value-Store, mit dem etwa Konfigurationsparameter mittels Consul im Cluster verteilt werden können. Clients können auch von sich aus Einträge im KV-Store verändern.

Das folgende Beispiel soll die spezielle Funktionsweise verdeutlichen.

8.3.1 Konfiguration

Schreiben Sie die Konfigurationsdatei */etc/prometheus/prometheus.yml* so um, dass sie folgenden Text enthält:

```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
    - targets: ['localhost:9090']
  - job_name: ec2
    ec2_sd_configs:
    - region: eu-central-1
      access_key: AKIAR4Z44I4N65US3VLO
      secret_key: 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
    relabel_configs:
    - source_labels: [__meta_ec2_public_ip]
      regex: '(.*)'
      target_label: __address__
      replacement: '${1}:9100'
    - source_labels: [__meta_ec2_tag_aws_cloudformation_logical_id]
      regex: '(.*)'
      target_label: instance
      replacement: '${1}'
```

Listing 8.1 Änderungen an der Datei /etc/prometheus/prometheus.yml

Es gibt hier weiterhin einen Eintrag unter `static_configs`, nämlich dass sich der Prometheus-Server selbst nach seinen eigenen Metriken fragt. Anstatt diese automatisch mit einzuspeisen, ist das ein durchaus eleganter Weg, um die eigenen Metriken mit in die Zeitreihendatenbank zu bekommen. Wenn Sie die Metriken des Prometheus-Servers selbst aber absolut nicht benötigen, dann können Sie den Eintrag mit dem Jobnamen `prometheus` auch einfach entfernen.

Unter dem Jobnamen `ec2` findet hingegen eine Service Discovery statt, die die IP-Adressen von möglichen Targets innerhalb in AWS EC2 ermittelt. Das dafür benötigte Konfigurationselement ist `ec2_sd_configs`. Als Optionen erwartet es die Region, in der die Erkennung stattfinden soll. Wie bei allen EC2-Beispielen in diesem Buch ist das weiterhin *eu-central-1* (Region Frankfurt am Main).

Außerdem müssen Sie hier die Credentials von einem Benutzer aus Ihrem Konto bei AWS eintragen, die für die Diensterkennung innerhalb von EC2 benötigt werden. In Abschnitt 3.1.1 habe ich beschrieben, wie Sie einen solchen Benutzer aufsetzen und die Credentials mit den beiden Schlüsseln für den programmgesteuerten Zugriff bekommen können. Prometheus benötigt für die Service Discovery im Zusammenhang mit AWS EC2 nur eine IAM-Richtlinie, und zwar `AmazonEC2ReadOnlyAccess`.

Unter `relabel_configs` sind zwei Operationen aus dem Funktionsbereich `Relabeling` definiert, die Prometheus an den automatisch angesetzten Labels für die eingebundenen EC2-Instanzen vornehmen soll:

- Die erste Operation tauscht die hinterher unter `__adress__` automatisch abgelegte *private* IPv4-Adresse der Instanz gegen die *öffentliche* IPv4-Adresse aus, die aus dem Label `__meta_ec2_public_ip` verfügbar ist. Denn über die private Adresse innerhalb von EC2 können Sie die Targets von Ihrem Arbeitsrechner aus (auf dem der Prometheus-Server auch für dieses Beispiel immer noch läuft) nicht ansprechen.

  Sie verwenden die private AWS-interne IPv4-Adresse deshalb standardmäßig als Scraping-Ziel, weil der Prometheus-Server eigentlich in einem produktiven Setup auf einer Instanz innerhalb desselben virtuellen Netzwerks laufen soll. Von dort aus kann er die Targets über ihre private IPv4-Adresse auch erreichen.

  Im Zuge des Austauschs der IP-Adressen wird auch noch der Port 9100 für den *node_exporter* mit eingestellt.
- Die zweite Operation tauscht den automatisch eingesetzten Wert für das Label `instance` gegen den Wert unter dem Label `aws_cloudformation_logical_id` aus. Das Beispielsetup ziehe ich mit einem CloudFormation-Template hoch (in Abschnitt 4.1 ist beschrieben, wie Sie das machen). Nach diesem Austausch steht hier dann der Ressourcenname aus dem CloudFormation-Template (die Ressourcen heißen hier *node1*, *node2* und *node3*) anstatt der IP-Adresse des Knotens. Das macht die Abfragen hinterher angenehmer und übersichtlicher.

Warten Sie bitte noch einen Moment ab, das konkret angewendete Beispiel wird Ihnen den Sinn dieser Austauschoperationen noch weiter verdeutlichen.

8.3.2 Anwendung

Die Anwendung der Service Discovery ist sehr einfach, und dass alles dabei automatisiert abläuft, ist ja auch der Sinn der Sache.

Als Ausgangssituation laufen drei neue Cloud-Instanzen in AWS EC2, die den *node_exporter* installiert haben. Sie können AWS CloudFormation mit *UserData* einsetzen, um diese automatisiert aufzusetzen. Sie können auch dreimal hintereinander `aws ec2 run-instances` auslösen und dann Ansible mit dem Modul `command` oder `apt` benutzen, um das Linux-Paket mit dem Exporter überall zu installieren. Die Eckwerte für die Instanzen sind dieselben wie bei den Beispielen aus dem CloudFormation-Kapitel (siehe Abschnitt 4.1). So wird hier auch das Amazon Machine Image mit der ID *ami-01e444924a2233b07* verwendet.

Hinweis

Beachten Sie, dass die eingesetzte Sicherheitsgruppe den Port 9100 per Ingress-Regel freigeschaltet haben muss, sonst kann Prometheus den *node_exporter* auf diesen Instanzen nicht aus dem Internet ansprechen.

Starten Sie dann Prometheus mit dem Befehl `systemctl restart prometheus` neu. Dabei wird die veränderte Konfigurationsdatei mit dem Job `ec2` ausgewertet. Prometheus findet die Targets dann vollkommen automatisch. Es dauert nur einen Augenblick, und die drei EC2-Instanzen mit dem *node_exporter* darauf sind dann zusammen mit dem Prometheus-Server selbst als Targets für Prometheus verfügbar:

```
~$ curl -s localhost:9090/api/v1/targets | jq '.data.activeTargets[] \
   | .scrapeUrl + ": " + .health'
"http://3.120.159.51:9100/metrics: up"
"http://52.59.255.249:9100/metrics: up"
"http://3.121.233.104:9100/metrics: up"
"http://localhost:9090/metrics: up"
```

Der Gesundheitsstatus `up` zeigt an, dass die Knoten erfolgreich gescrapt werden.

Sie können sich einmal den API-Endpunkt `targets` insgesamt ausgeben lassen oder `jq` dafür einsetzen, Ihnen zumindest den ersten Eintrag von `activeTargets` in einem JSON-Dokument vollständig auszuwerfen:

```
~$ curl -s localhost:9090/api/v1/targets | jq .data.activeTargets[0]
{
  "discoveredLabels": {
    "__address__": "172.31.36.137:80",
    "__meta_ec2_availability_zone": "eu-central-1b",
    "__meta_ec2_instance_id": "i-00b1717ed1e92da21",
    "__meta_ec2_instance_state": "running",
    "__meta_ec2_instance_type": "t2.micro",
    "__meta_ec2_owner_id": "130587313947",
    "__meta_ec2_primary_subnet_id": "subnet-d4a883ae",
```

```
    "__meta_ec2_private_dns_name": "ip-172-31-36-137.eu-central-
                                          1.compute.internal",
    "__meta_ec2_private_ip": "172.31.36.137",
    "__meta_ec2_public_dns_name": "ec2-3-121-233-104.eu-central-
                                          1.compute.amazonaws.com",
    "__meta_ec2_public_ip": "3.121.233.104",
    "__meta_ec2_subnet_id": ",subnet-d4a883ae,",
    "__meta_ec2_tag_aws_cloudformation_logical_id": "node1",
    "__meta_ec2_tag_aws_cloudformation_stack_id": "arn:aws:cloudformation:
            eu-central-1:130587313947:stack/prometheustest/
            be381940-406a-11ea-ab42-02012ecdd3ee",
    "__meta_ec2_tag_aws_cloudformation_stack_name": "prometheustest",
    "__meta_ec2_vpc_id": "vpc-d05177b8",
    "__metrics_path__": "/metrics",
    "__scheme__": "http",
    "job": "ec2"
  },
  "labels": {
    "instance": "node1",
    "job": "ec2"
  },
  "scrapeUrl": "http://3.121.233.104:9100/metrics",
  "lastError": "",
  "lastScrape": "2024-01-26T20:42:34.871171978Z",
  "health": "up"
}
```

Sie finden unter `discoveredLabels` eine ganze Reihe von einzelnen Informationen über diese EC2-Instanz, darunter sogar auch Werte, die anzeigen, dass sie Bestandteil des CloudFormation-Stacks *prometheustest* ist.

Die `logical_id` dieser Instanz `node1` finden Sie als Ergebnis der zweiten Austauschoperation in der Konfigurationsdatei als Wert für das Label `instance` wieder. Hier würde ohne diesen Austausch der Wert `3.121.233.104:9100` stehen, der hinterher zu unbequem für Abfragen aus der Datenbank ist.

Namen der Metriken

Es kann vorkommen, dass sich die Namen von Metriken des Node Exporters im Laufe der Zeit verändern. Beispielsweise wurden bei dem Update des prometheus-node-exporter auf Version 0.16.0 einige Namen angepasst. Die Metrik `node_network_receive_bytes_total` hatte vorher den Namen `node_network_receive_bytes`.

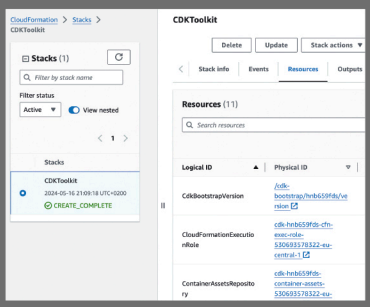
Der Name von Metriken stammt immer vom Exporter selbst, deshalb sollten Sie vor allem in produktiven Setups vorher prüfen, was sich bei einem Update verändert.

Cloud-Infrastrukturen

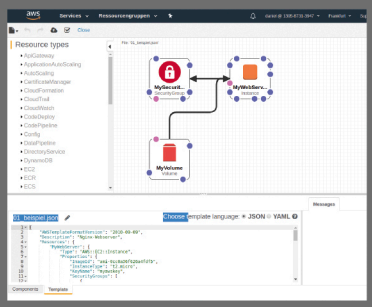
»There is no cloud, it's just someone else's computer.«

Welcome to the Cloud!

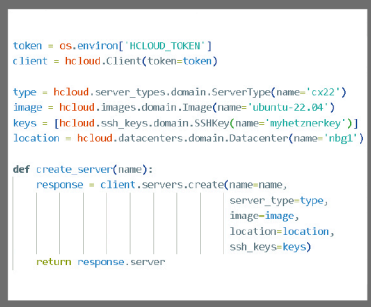
Der Weg zu modernen, zuverlässigen und skalierbaren IT-Infrastrukturen führt in die Cloud – und dieses Buch begleitet Sie dorthin. Sie lernen die wichtigsten Anbieter, Tools und Arbeitstechniken in der Praxis kennen und erfahren, wie Sie Ihr eigenes virtualisiertes Rechenzentrum einrichten.



Infrastruktur einrichten



Systeme planen



```
token = os.environ['HCLOUD_TOKEN']
client = hcloud.Client(token=token)

type = hcloud.server_types.domain.ServerType(name='cx22')
image = hcloud.images.domain.Image(name='ubuntu-22.04')
keys = [hcloud.ssh_keys.domain.SSHKey(name='myhetznerkey')]
location = hcloud.datacenters.domain.Datacenter(name='nbg1')

def create_server(name):
    response = client.servers.create(name=name,
                                     server_type=type,
                                     image=image,
                                     location=location,
                                     ssh_keys=keys)
    return response.server
```

Ausführliche Codebeispiele

Infrastructure as a Service

Die Vorteile von IaaS kennen Sie: flexibler Zuschnitt, Kostenersparnis, Ressourcen nach Bedarf. Dem stehen Sicherheitsprobleme, komplexe Konfigurationen und das Vendor-Lock-in gegenüber. Wägen Sie Chancen und Risiken ab und finden Sie die Lösung, die für Sie passt.

Tools und Skills

Cloud Engineers brauchen zeitgemäße Werkzeuge: Nutzen Sie Python, Go, Ansible, Terraform, Prometheus und Co., um Systeme aufzubauen und fernzusteuern. So richten Sie sich Ihre ideale Arbeitsumgebung ein und konfigurieren Ihre Systeme effizient.

Automatisieren und überwachen

Die Cloud spielt ihre Vorteile dann aus, wenn Sie Ihre Workflows automatisieren und verteilt orchestrieren. Erfahren Sie hier, wie Sie immer den Überblick behalten und die Cloud-Ressourcen für sich arbeiten lassen.

Alle Codebeispiele und Projektdateien zum Download

Kevin Welter ist Geschäftsführer der HumanITy GmbH, die Großkonzerne und KMUs zu Cloud-Systemen berät. Er hat das bewährte Praxisbuch von Daniel Stender überarbeitet und zeigt Ihnen, wie Sie Ressourcen in der Cloud administrieren.

Aus dem Inhalt

Rheinwerk Computing

Gedruckt in Deutschland
Mineralölfreie Druckfarben
Zertifiziertes Papier

€ 49,90 [D] € 51,30 [A]

Netzwerke & Datenbanken
ISBN 978-3-367-10426-0