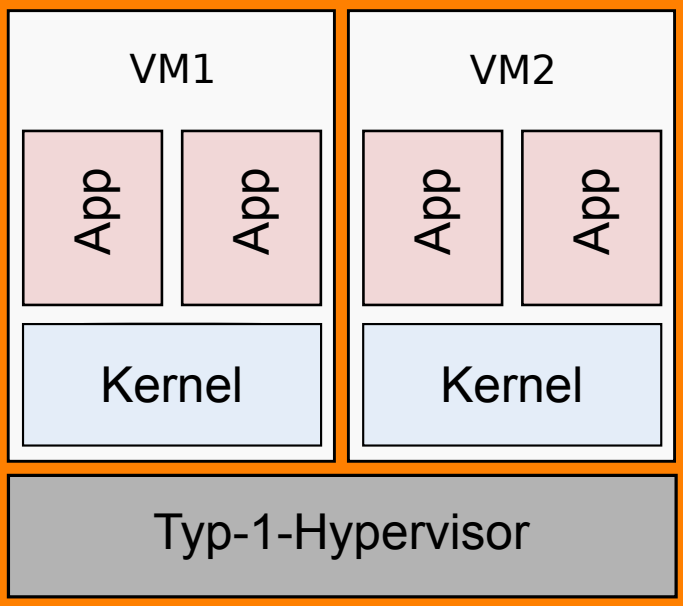
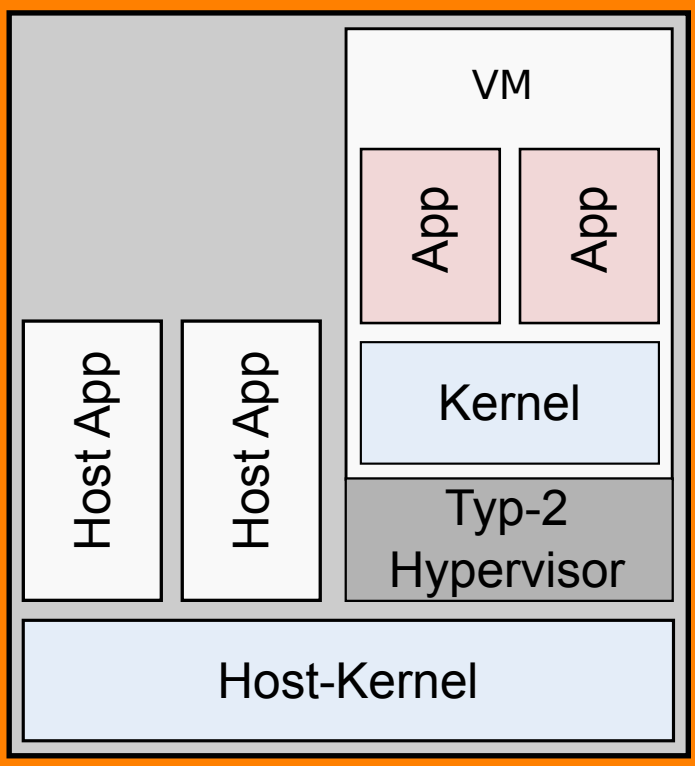
Open C3S
Open Competence Center for Cyber Security

UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik





Mikromodul 8001: Grundlagen von Virtualisierungstechnik und Cloud Computing

Autoren:
Prof. Dr. Hans P. Reiser
Noëlle Rakotondravony
Johannes Köstler

Universität Passau

Mikromodul 8001: Grundlagen von Virtualisierungstechnik und Cloud Computing

Autoren:

Prof. Dr. Hans P. Reiser

Noëlle Rakotondravony

Johannes Köstler

---

1. Auflage

Universität Passau

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

AUFSTIEG DURCH BILDUNG >>
OFFENE HOCHSCHULEN

Universität Passau
Fakultät für Informatik und Mathematik
Innstraße 43
94034 Passau

1. Auflage (4. Mai 2017)

Um die Lesbarkeit zu vereinfachen, wird auf die zusätzliche Formulierung der weiblichen Form bei Personenbezeichnungen verzichtet. Wir weisen deshalb darauf hin, dass die Verwendung der männlichen Form explizit als geschlechtsunabhängig verstanden werden soll.

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 16OH12025 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

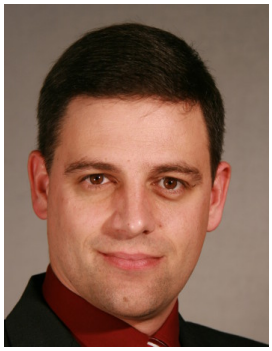
Inhaltsverzeichnis

Einleitung

I. Abkürzungen der Randsymbole und Farbkodierungen

Definition	D
Übung	Ü

II. Zu den Autoren



Hans P. Reiser ist Juniorprofessor für Sicherheit in Informationssystemen an der Universität Passau. Schwerpunkte seiner Arbeitsgruppe sind die Weiterentwicklung von Konzepten und Systemen aus dem Bereich der fehler- und einbruchstoleranten Replikation, die frühzeitliche und umfassende Erkennung von Sicherheitsproblemen und Sicherheitsvorfällen in Cloud-Umgebungen sowie die Erforschung neuartiger Sicherheitskonzepte auf Hypervisorebene.

Noëlle Rakotondravony ist seit September 2015 wissenschaftliche Mitarbeiterin in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Johannes Köstler ist seit Mai 2015 wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Prof. Hans P. Reiser an der Universität Passau.

Mikromodul: Grundlagen von Virtualisierungstechnik und Cloud Computing

1 Lernziele

Virtualisierungstechnik bildet die Grundlage von allen Cloud-Infrastrukturen. In diesem Mikromodul erhalten Sie zunächst einen Überblick über die geschichtlichen Hintergründe und die unterschiedlichen Arten von *Virtualisierungstechnik*. Darauf aufbauend entwickeln Sie ein vertieftes Verständnis für die Konzepte und Architekturen von *Rechnervirtualisierung*. Des Weiteren kennen Sie nach Abschluss dieses Mikromoduls die wichtigsten Aspekte der Entstehungsgeschichte von *Cloud Computing* und sind mit der Definition dieses Begriffes vertraut. Sie verfügen über ein vertieftes Verständnis über die wesentlichen Service- und Verarbeitungs-Modelle.

2 Virtualisierung: Geschichtliche Hintergründe

Die Ursprünge von Virtualisierungstechnik gehen zurück in die 60er-Jahre. Als erstes System, das es im Sinne von Rechnervirtualisierung ermöglichte, dass mehrere Benutzer auf einer physischen Hardware mehrere unabhängige Betriebssysteme ausführen, gilt das unter Leitung von Creasy entwickelte IBM CP-40 auf Basis eines Mainframe-Rechners IBM System/360 Model 40 mit modifizierter Speicherverwaltung (Creasy, 1981). Hierbei stellt ein *Control Program* (was man heute üblicherweise als Virtual Machine Monitor oder Hypervisor bezeichnen würde) mehrere Ausführungsumgebungen (*virtuelle Maschinen*) bereit, die sich jeweils wie ein isolierter physischer Rechner verhalten. Diese zum damaligen Zeitpunkt bahnbrechende Architektur beeinflusste maßgeblich nachfolgende Systeme, insbesondere CP-67 für IBM System/360 Model 67 und ab 1972 VM/370 für IBM System/370.

Ermöglicht wurde die Implementierung dieser ersten Virtualisierungsarchitektur vor allem durch zwei Komponenten: Zum einen entstanden ab den späten 50er-Jahren erstmals Konzepte von virtuellem Speicher. Getrieben durch die Motivation, teuren Hauptspeicher durch preiswerteren Trommelspeicher zu erweitern, kamen nun Architekturen auf, die eine Abbildung von virtuellen Adressen auf echte (physische) Speicheradressen in Hardware unterstützten. Zum anderen wurde im System/360 zwischen den CPU-Ausführungszuständen „*privileged*“ und „*problem*“ unterschieden. Im *problem*-Zustand sind nur solche Instruktionen verfügbar, die gewöhnliche Anwendungsprogramme normalerweise verwenden und die keine Nebenwirkungen auf andere virtuelle Maschinen haben. Im *privileged*-Zustand sind darüber hinaus *sensitive* Instruktionen verfügbar, die sich auf den gesamten Rechner auswirken und daher auch andere virtuelle Maschinen beeinflussen können oder Zustandsinformationen zu anderen virtuellen Maschinen lesen können. Wird eine solche privilegierte Operation im nichtprivilegierten Zustand ausgeführt, so wird die Ausführung unterbrochen und die Kontrolle an eine dedizierte Behandlungsroutine im *Control Program* übergeben. Diese kann dann die Wirkung dieser privilegierten Instruktion innerhalb der virtuellen Maschine simulieren (Creasy, 1981).

Bereits aus dieser kurzen Darstellung lässt sich erkennen, dass ein Rechner gewisse Anforderungen erfüllen muss, um die Ausführung mehrerer virtueller Maschinen zu ermöglichen. Es zeigte sich, dass eine derartige Virtualisierung auf manchen damals üblichen Rechnerarchitekturen nicht möglich war. Beispielsweise gab es bei einer DEC PDP-10 nichtprivilegierte Instruktionen, die aber dennoch sensitiv im Sinne von Auswirkungen auf den ganzen Rechner sind (Galley, 1973). Eine systematische, formalisierte Untersuchung dieser Voraussetzungen, welche ein

System erfüllen muss um Hardware virtualisieren zu können, wurde erstmals von Popek und Goldberg (1974) durchgeführt.

Nach Popek und Goldberg ist ein *Virtual Machine Monitor (VMM)* ein *Control Program*, das drei bestimmte Eigenschaften erfüllt (siehe Definition 1).

D

Definition 1: Eigenschaften von Virtualisierung nach Popek und Goldberg

- **Effizienz:** Alle unproblematischen Maschineninstruktionen werden direkt von der CPU ausgeführt, ohne Intervention durch das *Control Program*.
- **Ressourcenkontrolle:** Das *Control Program* hat stets die volle Kontrolle über Ressourcenzuteilungen, d. h., ein Programm in einer virtuellen Maschine darf nicht die Möglichkeit haben, die ihm zur Verfügung stehenden Ressourcen eigenmächtig zu verändern.
- **Äquivalenz:** Die Ausführung eines Programms in einer virtuellen Maschine ist – bis auf zwei Ausnahmen – nicht von der direkten Ausführung auf der Hardware (ohne *Control Program*) zu unterscheiden.

  Ausnahmen bei der Nichtunterscheidbarkeit sind beim Zeitverhalten und bei der Ressourcenverfügbarkeit erlaubt. Ist bei der Ausführung von problematischen Instruktionen die Intervention des *Control Programs* erforderlich, darf dies mehr Zeit in Anspruch nehmen als die direkte Ausführung auf der Hardware. Außerdem kann sich die Ausführung in einer virtuellen Maschine dadurch unterscheiden, dass sich diese die physischen Ressourcen mit dem *Control Program* und ggf. weiteren virtuellen Maschinen teilt und dadurch weniger Ressourcen zur Verfügung stehen.

Eine hinreichende Voraussetzung, um diese drei Bedingungen zu erfüllen, wurde von Popek und Goldberg (1974) mit folgendem Theorem formuliert:

D

Definition 2: Popek-Goldberg-Theorem zu Virtualisierbarkeit

Ein Computer lässt sich virtualisieren, wenn die Menge der sensitiven CPU-Instruktionen eine Teilmenge der privilegierten CPU-Instruktionen ist.

Es wird dabei von einer Architektur ausgegangen, bei der zwischen einem *privilegiertem* und einem *nichtprivilegierten* CPU-Modus unterschieden wird. Privilegierte Instruktionen sind solche, die i.d.R. nur im privilegierten Modus ausgeführt werden sollten und im nichtprivilegierten Modus einen *Trap* verursachen. Bei den sensitiven Instruktionen unterscheiden Popek und Goldberg (1974) zwischen *kontrollsensitiv* und *verhaltenssensitiv*. Erstere sind solche, die eine Kontrolle über Ressourcen ausüben, also z. B. die Zuteilung von Speicher oder den Betriebsmodus der CPU ändern. Zweitere sind solche, bei denen das Verhalten von äußeren Parametern (CPU-Modus, Abbildung der virtuellen auf physische Adressen etc.) abhängt. Für eine mathematische Formulierung dieses Modells sei an dieser Stelle an die Originalpublikation (Popek und Goldberg, 1974) verwiesen.

Während in den 60er- und 70er-Jahren Virtualisierungstechnik eine sehr große Bedeutung erlangte, konnte sich diese Entwicklung in der Folgezeit nicht in dem Maße fortsetzen. Zwar wurde Virtualisierung auf Mainframes auch weiterhin eingesetzt und weiterentwickelt, aber mit dem Aufkommen der *Personal Computer (PC)* ab den späten 70er-Jahren entstand eine preiswerte Alternative zu teuren Main-

frames, was zu einem großen Wandel in der IT-Landschaft führte. Infolgedessen verloren Mainframes und damit Virtualisierungstechnik an Bedeutung.

Die stark wachsende Verbreitung von x86-basierten PC-Systemen, insbesondere im Server-Bereich, führte aber auch zu neuen Herausforderungen. Oft waren PC-basierte IT-Systeme nur schlecht ausgelastet, was aus finanzieller Sicht auch unnötig hohe Kosten darstellte. Ende der 90er-Jahre kamen daher verstärkt Bestrebungen auf, auch PC-Systeme durch den Einsatz von Virtualisierungstechnik effizienter und damit kostengünstiger zu nutzen. Auf die Herausforderungen und Entwicklungen von Virtualisierung auf x86-basierten Systemen kommen wir in Abschnitt 4 zurück.

3 Virtualisierungsarten

Betrachtet man neben diesem kurzen historischen Überblick die Verwendung des Begriffs *Virtualisierung* in aktuellen IT-Systemen, so wird man eine deutliche Diskrepanz feststellen. Während Popek und Goldberg (1974) von einem sehr eng umgrenzten Begriff ausgehen, findet sich heute der Begriff in sehr vielen unterschiedlichen Kontexten wie Speicher-Virtualiserung, Netz-Virtualisierung, Java-Virtual-Machines etc. Eine allgemeinere Definition des Begriffes Virtualisierung stammt von Tanenbaum und Steen (2006):

Definition 3: Virtualisierungs-Definition nach Tanenbaum und Steen (2006)

„In its essence, virtualization deals with extending or replacing an existing interface so as to mimic the behavior of another system.“

D

Nach dieser allgemein gehaltenen Definition geht es bei Virtualisierung also darum, das Verhalten eines Systems auf Grundlage einer vorhandenen Schnittstelle nachzubilden. Dies umfasst, abhängig von der Konkretisierung des Begriffs „System“, eine Vielzahl möglicher Varianten. Die wichtigsten Varianten davon sind folgende:

- *Rechner-Virtualisierung:* Virtualisierung eines physischen Rechners, d. h. beim nachgebildeten System handelt es sich um Hardwarekomponenten, insbesondere CPU und Hauptspeicher;
- *Betriebssystem-Virtualisiserung:* Bereitstellung einer Ausführungsumgebung, die der eines Betriebssystems entspricht;
- *Anwendungs-Virtualisierung:* Bereitstellung einer Ausführungsumgebung für eine bestimmte Anwendung;
- *Netz-Virtualisierung:* Bereitstellung eines Kommunikationsnetzes mit bestimmten Eigenschaften (Topologie, Latenzen etc.).

3.1 Rechner-Virtualisierung

Bei der Rechner-Virtualisierung entspricht die Schnittstelle, die nachgebildet wird, der Schnittstelle der physischen Hardware des Rechners, also insbesondere von CPU und Hauptspeicher, sowie ggf. von weiteren Peripheriegeräten. Manchmal wird diese Form von Virtualisierung entsprechend auch als Prozessor- und Speichervirtualisierung bezeichnet. Bei dem ursprünglichem Virtualisierungskonzept von Popek und Goldberg (1974) handelt es sich um Rechner-Virtualisierung.

In diesem Zusammenhang werden neben dem Begriff *Virtualisierung* oft auch die Begriffe *Emulation* und *Simulation* verwendet. Leider hat sich hierbei keine einheitliche Terminologie durchsetzen können, so dass die Begriffe in der Praxis oft unterschiedlich verwendet werden und die Wortwahl teilweise kontrovers diskutiert wird. Auch wenn sich eine gewisse Unschärfe nicht ganz vermeiden lässt, wollen wir im Folgenden dennoch diese Begriffe etwas genauer voneinander abgrenzen.

Abgrenzung zwischen Emulation und Virtualisierung

Für den Begriff der Emulation gibt es eine prägnante und treffende Definition im Duden:

D

Definition 4: Emulation

„Nachahmung der Funktionen eines anderen Computers“ (Bibliographisches Institut GmbH - Duden Verlag (Hrsg.), 2006)

Die Rechner-Emulation weist eine starke Ähnlichkeit zur Rechner-Virtualisierung auf: Aus Sicht von Betriebssystem und Anwendungen innerhalb der emulierten bzw. virtualisierten Umgebung stellt diese Umgebung konzeptionell einen Rechner bereit. Die Schnittstellen dieses nachgebildeten Rechners sind dabei den Schnittstellen von echter Rechnerhardware (Prozessor, Speicher, I/O-Geräte) nachempfunden. Von einigen Autoren werden beide Begriffe daher als Synonyme verwendet.

Üblich ist dennoch, zwischen beiden Begriffen zu differenzieren: Legt man den Begriff Virtualisierung eng nach der Definition von Popek und Goldberg (1974) aus, so versteht man darunter aufgrund der geforderten *Effizienz-Eigenschaft* nur solche Fälle, in denen die nachgebildete CPU-Architektur der physisch vorhandenen CPU-Architektur entspricht und unproblematische Instruktionen unverändert direkt ausgeführt werden können. Dagegen zielt die Emulation darauf, eine *andere* Architektur nachzubilden.

Eine derartige Differenzierung ist in der Praxis kritisch zu hinterfragen. Eine Nachbildung einer Architektur auf Grundlage einer anderen Architektur ist heute mit Hilfe moderner Techniken zur dynamischen Binärtransformation oft mit hoher Effizienz möglich, so dass dieses Unterscheidungsmerkmal in der Praxis vernachlässigbar ist. Auch die genaue Grenzziehung zwischen beiden Begriffen ist in der Praxis schwierig. Wird beispielsweise eine 32 bit-x86-CPU auf einer 64 bit-x86-CPU nachgebildet, so könnte man argumentieren, dass es sich wegen der Nachbildung einer anderen Architektur um Emulation handelt. Allerdings geschieht dies durch wesentliche Unterstützung durch die Hardware, so dass dennoch obengenannte Effizienz-Eigenschaft von Virtualisierung erfüllt ist. Trotz dieser unscharfen Abgrenzung werden wir in diesem Studienbrief versuchen, soweit sinnvoll möglich, die Begriffe *Virtualisierung* und *Emulation* differenziert jeweils für die Nachbildung einer identischen bzw. unterschiedlichen Hardware-Plattform zu verwenden.

Abgrenzung von Simulation

Unabhängig von einer Differenzierung zwischen Virtualisierung und Emulation lässt sich der Begriff Simulation (siehe Definition 5) betrachten. Während bei Emulation und Virtualisierung die Nachbildung eines Systems an einer Schnittstelle nach außen im Vordergrund steht, ist Simulation die nach innen gerichtete Nachbildung, bei der der Erkenntnisgewinn über das System selbst im Vordergrund steht.

Daher ist Simulation ein Begriff, der sich nicht als Synonym für Virtualisierung oder Emulation eignet.

Definition 5: Simulation nach VDI

„Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind.“ (VDI-Richtlinie 3633, 2013).

D

Das vorliegende Modul wird in diesem und in weiteren Studienbriefen vertieft Sicherheitsfragen und Probleme bei forensischen Untersuchungen betrachten, die sich beim Einsatz von Rechner-Virtualisierung ergeben können. Daher betrachten wir Rechner-Virtualisierung auf aktuell üblicher IT-Hardware vertieft in Abschnitt 4.

Unterarten

Eine weiter untergliederte Klassifikation von Rechner-Virtualisierung geht zurück auf die Arbeit von Goldberg (1973), bei der zwischen einem *Typ-1-* und *Typ-2-*Hypervisor unterschieden wird. Dieser Unterschied ist in Abbildung 1 illustriert.

Ein Typ-1-Hypervisor wird direkt auf der physischen Hardware ausgeführt, er benötigt kein Betriebssystem als Basis. Eine plakativere und vom Verfasser des Studienbriefs daher bevorzugte Bezeichnung ist *Bare-Metal*-Hypervisor oder nativer Hypervisor. Die Herausforderung bei einer Typ-1-Architektur ist, dass der Hypervisor geeignete Treiber zum direkten Zugriff auf die Hardware enthalten muss.

Ein Typ-2-Hypervisor setzt dagegen auf einem Host-Betriebssystem auf und nutzt dessen Treiber zum Zugriff auf die Hardware. Entsprechend üblich ist auch die einprägsamere Bezeichnung als *Hosted*-Hypervisor. Aus Sicht des Host-Betriebssystems stellt der Hypervisor dabei nur einen Anwendungsprozess dar. Eine Herausforderung bei dieser Architektur ist es, zu vermeiden, dass die zusätzliche Schicht in der Architektur zu Performanceeinbußen führt.

In der Praxis ist diese klare Trennung in zwei Kategorien oft weniger eindeutig ausgeprägt. Beispielsweise handelt es sich bei Xen[1] um einen Typ-1-Hypervisor. Dennoch gibt es auch bei Xen (logisch „oberhalb“ des Hypervisors) eine privilegierte virtuelle Maschine (*Dom0*), deren Gerätetreiber vergleichbar zum Host-Betriebssystem bei einer Typ-2-Architektur für den Zugriff auf die Hardware verwendet werden. Die Emulation virtueller Geräte für die virtuelle Maschine erfolgt durch Anwendungsprozesse innerhalb von *Dom0*. VirtualBox[2] dagegen zählt als Beispiel einer Typ-2-Architektur. Doch auch in diesem Beispiel wird der Hypervisor nicht vollständig als Anwendungsprozess des Host-Betriebssystems ausgeführt, sondern benötigt auch Kernelmodule im Host-Betriebssystem, die direkt auf die Hardware zugreifen.

3.2 Betriebssystem-Virtualisierung

Anders als bei der Rechner-Virtualisierung, bei der jede virtuelle Maschine ein eigenes Betriebssystem ausführt, werden bei der Betriebssystem-Virtualisierung isolierte Benutzer-Instanzen auf Basis eines Betriebssystems bereitgestellt. Üblich

[1] http://www.xenproject.org [abgerufen am 2017-01-05]
[2] http://www.virtualbox.org [abgerufen am 2017-01-05]

Abb. 1: Typ-1-Hypervisor (links) vs. Typ-2-Hypervisor (rechts)

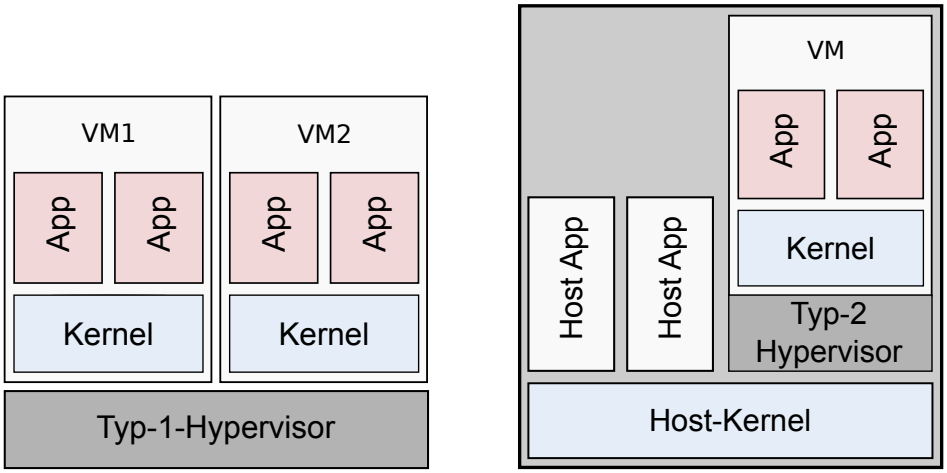
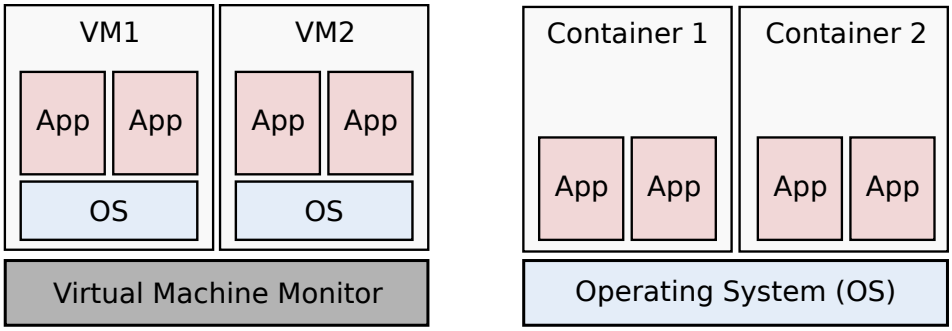


Abb. 2: Vergleich von Rechner-Virtualisierung (links) und Betriebssystem-Virtualisierung (rechts)



ist die Bezeichnung dieser Instanzen als *Container*. Daher ist auch der Begriff *Container-Virtualisierung* als Synonym weit verbreitet (siehe auch Abbildung 2).

Aus Anwendersicht sind die Unterschiede zur Rechner-Virtualisierung oft vernachlässigbar gering. In beiden Varianten erhält der Anwender eine isolierte Ausführungsebene, in der er selbst beliebige Anwendungen ausführen kann. Lediglich der Betriebssystemkern kann bei dieser Form der Virtualisierung nicht beliebig gewählt werden, sondern ist vom System außerhalb des Anwender-Containers vorgegeben.

Beispiele für aktuelle Produkte, die Betriebssystem-Virtualisierung bieten, sind unter anderem Linux-VServer[3], OpenVZ[4] und Docker[5].

Im direkten Vergleich mit Rechner-Virtualisierung gibt es insbesondere folgende Unterschiede:

Vorteile:

- Geringerer Ressourcenbedarf: Betriebssystem-Virtualisierung erfordert weniger Ressourcen. Es gibt insgesamt nur ein Betriebssystem, daher werden dessen Ressourcen (insbesondere der Hauptspeicher) nur einmal benötigt.
- Effizientere Ausführung: Es gibt praktisch keinen Unterschied zwischen einer Anwendung auf einem nativen Betriebssystem und in einem Betriebssystem-Container. In beiden Fällen interagiert die Anwendung mit dem gleichen System-Interface. Bei der Rechnervirtualisierung entsteht dagegen zusätzlicher Overhead durch eine Indirektionsschicht: Das

[3] http://linux-vserver.org/ [abgerufen am 2017-01-05]
[4] http://openvz.org/ [abgerufen am 2017-01-05]
[5] http://docker.com/ [abgerufen am 2017-01-05]

Gast-Betriebssystem muss mit dem Host-Betriebssystem bzw. dem Virtual Machine Monitor interagieren, und die dafür notwendigen Kontextwechsel erfordern zusätzlichen Aufwand.

- Schnellerer Start: Es muss kein neues Betriebssystem gestartet werden, ein virtueller Container kann wesentlich schneller erzeugt werden als eine virtuelle Maschine.

Nachteile:

- Flexibilität: Für verschiedene Gäste können keine unterschiedlichen Betriebssysteme bzw. Betriebssystemkerne verwendet werden.

In Bezug auf Fragen der Sicherheit gibt es vor allem zwei relevante Unterschiede:

- Während bei Rechner-Virtualisierung der Virtual Machine Monitor für die Trennung zwischen den virtuellen Maschinen verantwortlich ist, ist es bei Betriebssystem-Virtualisierung das Betriebssystem. Hier kann argumentiert werden, dass ein (monolithisches) Betriebssystem aufgrund der größeren Funktionalität und der damit einhergehenden größeren Codebasis möglicherweise mehr Schwachstellen enthalten kann als ein schlankerer Virtual Machine Monitor, dessen einzige Aufgabe es ist, die virtuellen Maschinen zu verwalten.
- Die Verantwortung für die Wartung des Betriebssystems (insbesondere Sicherheits-Patches) liegt bei Hardware-Virtualisierung beim Anwender selbst, während sie bei Betriebssystem-Virtualisierung beim Betreiber der Infrastruktur liegt.

3.3 Anwendungs-Virtualisierung

Im Gegensatz zu den bisher betrachteten Varianten, bei denen jeweils die komplette Umgebung eines Betriebssystems zur Verfügung gestellt wird, ist die Anwendungs-Virtualisierung auf einer höheren Schicht angesiedelt. Oberhalb des Betriebssystems wird gezielt für eine *Anwendung* eine spezifische Ausführungsumgebung bereitgestellt, die oft auch als *Sandbox (Sandkasten)* bezeichnet wird. Anwendungsvirtualisierung schafft also eine zusätzliche Zwischenschicht zwischen Anwendung und Betriebssystem.

In der Praxis finden sich zwei Varianten von Anwendungs-Virtualisierung, die sich darin unterscheiden, wie die Schnittstelle zur Anwendung gestaltet ist.

Eine bekannte Variante besteht darin, für die Anwendung eine Ausführungsumgebung mit einer speziellen Schnittstelle (völlig unabhängig von der Betriebssystem-Schnittstelle) bereitzustellen. Verbreitete Beispiele hierzu sind das .Net-Framework von Microsoft mit der CLR (Common Language Runtime) und Java mit der JVM (Java Virtual Machine). Diese Variante bietet insbesondere den Vorteil der Portabilität, da die Anwendung auf unterschiedlichen Betriebssystemen verwendet werden kann, solange dort jeweils die Ausführungsumgebung vorhanden ist. Darüber hinaus lässt sich der Zugriff auf Ressourcen des Systems einschränken.

Die zweite Variante besteht darin, eine existierende Anwendung, welche die normale Schnittstelle eines Betriebssystems nutzt, durch eine Virtualisierungsschicht zu kapseln. Hier können sowohl Sicherheitsüberlegungen eine Rolle spielen als auch die Vereinfachung der Installation bzw. des Deployments. Eines von vielen

Beispielen in dieser Kategorie ist VMware ThinApp[6]. Ein wichtiger Anwendungszweck dieser Form der Virtualisierung ist es, eine komplette Anwendung mit all ihren Abhängigkeiten in eine in sich geschlossene ausführbare Datei zu packen, die dann ohne zusätzlichen Installationsaufwand auf einem System ausgeführt werden kann. Auch lassen sich damit einfach verschiedene Versionen einer Anwendung, die sich oft wegen Konflikten bei Abhängigkeiten nicht zeitgleich direkt auf einem System installieren lassen, auf einem System verwenden. Auch die bereits beschriebene Containervirtualisierung lässt sich zur Anwendungsvirtualisierung nutzen.

3.4 Netz-Virtualisierung

Eine weitere Variante der Virtualisierung ist die Netz-Virtualisierung. Diese findet hier der Vollständigkeit halber Erwähnung, wird aber in diesem Modul keine signifikante Rolle spielen. Während es bei allen zuvor diskutierten Varianten um Virtualisierung auf unterschiedlichen Ebenen innerhalb des Rechners ging, betrachtet die Netz-Virtualisierung die an der Kommunikation beteiligten Komponenten und zielt darauf, eine Trennung zwischen den physischen Komponenten und den Funktionen wie Router, Switch und Port zu erreichen.

In einfacher Form hat sich Netz-Virtualisierung schon seit längerem etabliert. Das bekannteste klassische Beispiel stellen *Tagged VLANs* (Virtual Local Area Networks) nach IEEE Std 802.1Q (2014) dar. Dabei wird ein einzelnes physisches Netz in mehrere logische Netze unterteilt. Ein physischer Switch kann dabei in mehrere logische Switches aufgeteilt werden. Die Zuordnung einzelner Geräte zu virtuellen Netzen erfolgt dabei durch „Tags“ in Software. Eine Umkonfiguration ist möglich, ohne die physischen Verbindungen ändern zu müssen.

Eine neuere Entwicklung im Bereich der Netz-Virtualisierung ist *Software Defined Networking (SDN)*. Dabei ist es das Ziel, das Verhalten von Netz-Geräten durch Software flexibler programmierbar und durch eine zentrale Instanz kontrollierbar zu machen. Einer der ersten und bekanntesten Standards für SDN ist OpenFlow (McKeown et al., 2008).

4 Vertiefung zu Rechner-Virtualisierung

4.1 Analyse von Robin und Irvine (2000)

Zu Beginn dieses Mikromoduls haben wir bereits die grundlegenden Merkmale gesehen, die im Mainframe-Bereich erstmals Rechner-Virtualisierung ermöglicht haben. Von zentraler Bedeutung waren dabei zum einen die Existenz eines privilegierten und eines nichtprivilegierten CPU-Modus, sowie eine Unterstützung von Speichervirtualisierung.

Auf der Intel Pentium-Architektur finden sich all diese grundlegenden Dinge. Es gibt in der Architektur vier Ausführungsmodi (*CPL – current privilege level*), die auch als Ringe bezeichnet werden. Ring 0 ist dabei der privilegierteste Modus, Ring 3 der am wenigsten privilegierte. Nur die wenigsten Betriebssysteme nutzen alle 4 Modi aus. Weit verbreitet ist die Nutzung von nur zwei Modi: Ring 0 als privilegierter Modus, der vom Betriebssystem verwendet wird, und Ring 3 als nichtprivilegierter Modus, in dem Anwendungsprogramme ausgeführt werden.

Allerdings spielte bei der Entwicklung der Intel-Pentium-Architekur Virtualisierung keine nennenswerte Rolle, und daher wurden keine dedizierten Maßnahmen zur Unterstützung von Virtualisierung getroffen. Die Frage, ob Virtualisierung

[6] http://www.vmware.com/products/thinapp.html [abgerufen am 2017-01-05]

dennoch mit der Pentium-Architektur möglich ist, wurde von Robin und Irvine (2000) systematisch untersucht. Diese Untersuchung zielt darauf, ob es *sensitive unprivilegierte CPU-Instruktionen* gibt. Falls dies der Fall ist, sind die Anforderungen an ein virtualisierbares System nach Popek und Goldberg (siehe Definition 2) nicht erfüllt.

Die Autoren kommen zu dem Ergebnis, dass es 17 Instruktionen[7] gibt, die problematisch sind, da sie auf sensitive CPU-Register zugreifen (SGDT, SIDT, SLDT, SMSW, PUSHF, POPF) oder weil sie Informationen über den aktuellen Zugriffsschutz der Speichervirtualisierung liefern (LAR, LSL, VERR, VERW, POP, PUSH, CALL, JMP, INT n, RET/IRET, STR, MOVE).

Die Anforderungen nach Popek und Goldberg sind demnach nicht erfüllt. Die Aussage von Popek und Goldberg besagt aber lediglich, dass in deren Modell Virtualisierung möglich ist, wenn die Anforderungen erfüllt sind. Das heißt im Umkehrschluss aber nicht zwingend, dass Virtualisierung unmöglich ist, wenn die Anforderungen nicht erfüllt sind!

4.2 Lösungsansätze

Eine naheliegende Möglichkeit, das Problem der sensitiven, nichtprivilegierten CPU-Instruktionen zu umgehen, besteht darin, ein Gastsystem nicht direkt auf der CPU auszuführen, sondern in einem *Emulator*. Diese Variante zählt nicht zur Virtualisierung nach der engen Definition von Popek und Goldberg (siehe Definition 1), da die *Effizienz*-Eigenschaft verletzt wird. Optimierungen wie „on-the-fly“-Übersetzung zu nativen Code innerhalb des Emulators erlauben es, die Geschwindigkeitseinbußen gegenüber einem nicht virtualisierten System klein zu halten. Dennoch ist Emulation weniger effizient als andere Varianten der Virtualisierung. Beispiele für Emulatoren von einer x86-CPU sind Bochs[8] und JSLinux[9].

Eine effiziente Variante der Emulation ist das *Dynamic Rewriting*, das erstmals von VMware ESX Server auf x86-Hardware verwendet wurde (Devine et al., 1998). Dabei werden die Teile des Gast-Betriebssystems dynamisch so modifiziert, dass problematische Instruktionen im Gast-Betriebsystem durch geeignete Behandlungsroutinen ersetzt werden. Dieses Vorgehen weist eine hohe Komplexität auf und einen nicht zu vernachlässigen Aufwand zur Analyse und Modifikation des Gast-Betriebssystems zur Laufzeit. Dieser Aufwand fällt während der Ausführung allerdings nur einmal nach dem Start an.

Eine weitere Möglichkeit ist *CPU-Paravirtualisierung*. Bei dieser Möglichkeit wird auf die *Äquivalenz*-Eigenschaft nach Popek und Goldberg verzichtet. Das Gastsystem wird statisch so modifiziert, dass es keine problematischen CPU-Instruktionen (sensitiv, nichtprivilegiert) mehr besitzt. Diese Instruktionen werden durch spezielle *Hypercalls* zum Hypervisor ersetzt, der dann das gewünschte Verhalten der ersetzten Instruktionen bereitstellt. Erstmals populär wurde Paravirtualisierung durch den *Xen* Virtual Machine Monitor (Barham et al., 2003). Nachteil von Paravirtualisierung ist, dass das Gastsystem (insbesondere das in der virtuellen Maschine ausgeführte Betriessystem) modifiziert werden muss. Dies ist bei einem Closed-Source-Betriebssystem meist nicht möglich.

[7] Der Artikel listet explizit insgesamt 19 problematische Instruktionen auf. Eine davon, SMSW, ist nur für heute nicht mehr verwendete x86-real-mode-Anwendungen relevant. Davon abgesehen scheint die Abweichung zwischen der von Robin und Irvine (2000) genannten Zahl 17 und der Zahl der aufgelisteten Instruktionen ein Fehler der Originalpublikation zu sein.

[8] http://bochs.sourceforge.net [abgerufen am 2017-01-05]

[9] http://jslinux.org [abgerufen am 2017-01-05]

Im Jahr 2006 wurden sowohl von AMD (als AMD-V) als auch von Intel (als VT-x) Hardware-Erweiterungen für x86-CPUs eingeführt, mit deren Hilfe transparente *Hardware-gestützte Virtualisierung* möglich wurde. Unabhängig von dem bereits beschriebenen Ring-Modell gibt es auf x86-CPUs mit Virtualisierungsunterstützung zwei Ausführungsmodi: „root mode“ für den Hypervisor und „non-root mode“ für das Gastsystem. Damit wurde es nun erstmals möglich, ohne statische oder dynamische Veränderung des Gastsystems Virtualisierung auf der x86-Architektur zu verwenden.

Dennoch haben durch die Einführung von AMD-V und Intel VT-x die anderen Varianten der Virtualisierung ihre Daseinsberechtigung nicht ganz verloren. Vor allem aus Performance-Sicht ist bei Hardware-gestützter Virtualisierung zu bedenken, dass alle problematischen Instruktionen des Gast-Systems einen Kontextwechsel (Trap) zum Hypervisor erfordern, der einen großen Aufwand bedeutet (je nach CPU-Architektur etwa $1000 - 2000$ CPU-Taktzyklen). Sowohl bei Paravirtualisierung als auch bei entsprechend optimiertem Dynamic Rewriting lassen sich komplexe Operationen des Gast-Systems, die mehrere privilegierte Operationen erfordern, durch einen einzigen Aufruf zum Hypervisor ersetzen.

5 Geschichtliche Entwicklung von Cloud Computing

Der Begriff Cloud Computing ist heutzutage in vielen Bereichen der Informationstechnik omnipräsent. Während sich im ersten Jahrzehnt des Jahrhunderts Cloud-Computing zum großen Hype-Thema entwickelte und zum Teil mit unrealistischen Erwartungen verbunden war, haben sich inzwischen die Vorstellungen zur Begrifflichkeit und zu den Vor- und Nachteilen etwas konsolidiert.

Der Begriff Cloud Computing wird im Allgemeinen mit drei zentralen Merkmalen in Verbindung gebracht:

1. Auslagerung von Datenspeicherung und/oder –verarbeitung an Dritte:
   Ressourcen (Hardware, Speicherplatz, Dienste etc.) stehen nicht lokal zur Verfügung, sondern werden von externen Dritten zur Verfügung gestellt.
2. Bedarfsgerechte Skalierbarkeit:
   Ressourcen oder Dienste stehen in dem Umfang und der Leistungsfähigkeit bereit, in der sie benötigt werden. Eine Anpassung an geänderten Bedarf ist jederzeit einfach und schnell möglich („rapid elasticity“).
3. Verbrauchsgenaue Abrechnung („pay-as-you-go“):
   Nicht nur die Nutzung erfolgt variabel und bedarfsgerecht, auch die Abrechnung erfolgt entsprechend nach Nutzung.

An dieser Stelle sei zunächst darauf hingewiesen, dass vieles, was heutzutage unter den Begriff Cloud Computing fällt und von manchen als moderne innovative Entwicklung angesehen wird, in Wirklichkeit identisch oder zumindest sehr ähnlich zu vielen deutlich älteren Konzepten ist. Eindrucksvoll drückte dies beispielsweise Larry Ellison (CEO, Oracle) am 26.09.2008 im Wall Street Journal aus:

> „The interesting thing about cloud computing is that we've redefined cloud computing to include everything that we already do. [...] The computer industry is the only industry that is more fashion-driven than women's fashion [...] I don't understand what we would do differently in the light of cloud computing other than change the wording of some of our ads.“

In der Tat lassen sich einige Konzepte, die oft als innovative Charakteristika von Cloud Computing angesehen werden, bereits weit zurück in der Vergangenheit finden. So entwickelte sich bereits in den früher 60er-Jahren die Vision von „Utility Computing“, wobei Rechenleistung ähnlich wie Strom oder Wasser an Haushalte geliefert und dort nach Bedarf genutzt werden kann.

In eine ganz ähnliche Richtung zielte in den 80er-Jahren das Konzept des „Grid Computing“, wobei der Begriff eine Anspielung auf das Energienetz (electric power grid) ist: Rechenleistung soll also wie Elektrizität an Nutzer verteilt werden. Offensichtlich waren beide Konzepte jeweils ihrer Zeit voraus. Zwar gab es jeweils umfangreiche Forschungsarbeiten in diesen Gebieten, eine Verbreitung in der Praxis außerhalb von Forschungsprototypen blieb aber aus. Dafür mag es viele Gründe geben; an erster Stelle ist sicher neben der im Vergleich zu heute geringen Rechenleistung von Rechenanlagen vor allem die fehlende Verfügbarkeit von Daten-Kommunikationsnetzen zu nennen. Auch mangelte es an tragfähigen Abrechnungsmodellen.

Anfang des 21. Jahrhunderts lag eine stark veränderte Ausgangssituation vor. Schnelle Kommunikationsnetze haben inzwischen eine hohe Verbreitung gefunden und stehen für viele Unternehmen wie auch Privathaushalte preiswert zur Verfügung. Gleichzeitig platze Anfang 2000 die „Internet-Blase“. Viele Unternehmen, die in leistungsfähige Datencenter investiert hatten, standen nun mit teuren, aber sehr schlecht ausgelasteten IT-Ressourcen da. Hier entstand nun wachsendes Interesse, diese Ressourcen durch Vermietung an Dritte besser auszulasten.

6 Definition und Modelle von Cloud Computing

Heutzutage hat sich Cloud Computing zu einem in der Praxis etablierten Konzept entwickelt. Nach einer Hype-Phase gegen Ende des vergangenen Jahrzehnts, in der der Begriff „Cloud“ inflationär gebraucht wurde, hat sich auch die Begriffsbildung weiterentwickelt, und die Definition des National Institute of Standards and Technology (NIST) von 2011 ist heute allgemein akzeptiert.

6.1 NIST-Definition von Cloud Computing

Definition 6: Cloud Computing nach NIST

„Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.“ (Mell und Grance, 2011)

D

Bei Cloud Computing geht es also darum, Computer-Ressourcen von überall, bequem und nach Bedarf über Kommunikationsnetze nutzen zu können. Als Ressourcen kommen hierbei eine Vielzahl von Möglichkeiten in Frage: Netze, Server, Datenspeicher, Anwendungen und Dienste. Wesentlich dabei ist auch, dass die Provisionierung und Rückgabe der Ressourcen mit minimalen Aufwand für den Provider möglich ist.

In der NIST-Definition werden weiterhin fünf essentielle Eigenschaften, die den Kern von Cloud Computing ausmachen, näher definiert. Diese sind die folgenden (Mell und Grance, 2011):

- „On-demand self service“: Ein Nutzer (bzw. Kunde eines Cloud-Anbieters) kann selbst (ohne manuelle Intervention des Anbieters) die Bereitstellung von Ressourcen entsprechend seines Bedarfs veranlassen.
- „Broad network access“: Ressourcen werden über das Netz angeboten und können durch unterschiedlichste Client-Plattformen (wie Smartphones, Tablets, Notebooks oder Workstations) genutzt werden.
- „Resource pooling“: Vorhandene physische Ressourcen des Cloud-Anbieters werden durch mehrere Nutzer verwendet, wobei virtuelle Ressourcen dynamisch auf physische Ressourcen abgebildet werden und der Nutzer keine direkte Kontrolle darüber hat, welche physischen Ressourcen er nutzt.
- „Rapid elasticity“: Ressourcen können dynamisch bereitgestellt und zurückgegeben werden, um eine rasche Skalierung entsprechend dem Bedarf zu ermöglichen. Aus Sicht des Nutzers scheinen die Ressourcen in unbeschränktem Umfang bereitzustehen.
- „Measured Service“: Die Nutzung von Ressourcen wird auf geeignete Art gemessen, und damit stehen Informationen über die Ressourcennutzung sowohl dem Cloud-Anbieter als auch dem Nutzer transparent zur Verfügung.

6.2 Service-Modelle

Das NIST-Modell von Cloud Computing unterscheidet drei verschiedene Service-Modelle, die sich insbesondere darin unterscheiden, wie viel Kontrolle der Cloud-Nutzer über die ihm zur Verfügung gestellte Infrastruktur erhält.

- Bei *Infrastructure as a Service (IaaS)* werden dem Nutzer virtuelle Ressourcen (CPU, Speicher, Netz) in der Cloud-Infrastruktur direkt zur Verfügung gestellt. Der Nutzer hat keinen Zugriff auf die zugrunde liegende Infrastruktur. Er kann die bereitgestellten virtuellen Ressourcen aber frei nutzen, um darauf beliebige Software (einschließlich eines eigenen Betriebssystems) auszuführen. In diesem Modell trägt der Nutzer die größte Verantwortung.
- Bei *Platform as a Service (PaaS)* erhält der Nutzer die Möglichkeit, eigene Anwendungen auf einer vom Cloud-Provider bereitgestellten Plattform zu betreiben. Diese bereitgestellte Plattform umfasst das Betriebssystem und weitere Komponenten wie Bibliotheken, Middleware-Plattformen, Dienste und Werkzeuge. Vom Cloud-Nutzer direkt verwaltet werden nur die auf Basis dieser Plattform ausgeführten Anwendungen.
- Bei *Software as a Service (SaaS)* werden dem Nutzer fertig konfigurierte Anwendungen vom Cloud-Anbieter bereitgestellt, welche dieser über das Netz mittels eines anwendungsspezifischen Interface nutzen kann. Die Gestaltungsmöglichkeiten des Cloud-Nutzers beschränkten sich auf ggf. vorgesehene anwendungsspezifische Konfigurationsmöglichkeiten.

6.3 Deployment-Modelle

Darüber hinaus unterscheidet das NIST-Modell auch vier Deployment-Modelle. Hierbei wird differenziert, welche Nutzer jeweils auf eine Cloud-Infrastruktur Zugriff haben.

- Bei einer *Private Cloud* steht die physische Infrastruktur exklusiv für eine einzelne Organisation bereit. Dabei ist es unerheblich, ob die Infrastruktur von der Organisation intern selbst betrieben wird oder der Betrieb örtlich oder organisatorisch extern ausgelagert ist.

- Bei einer *Community Cloud* wird die physische Infrastruktur exklusiv für eine Gruppe von Nutzern („Community“) betrieben. Dieses Cloud-Modell bietet sich an, wenn mehrere Nutzer gemeinsame, spezifische Anforderungen z. B. hinsichtlich Sicherheitsmechanismen oder gesetzlichen Vorgaben haben.
- Bei einer *Public Cloud* steht die Infrastruktur öffentlich zur Nutzung zur Verfügung.
- Von einer *hybriden Cloud* spricht man, wenn die verwendete Cloud-Infrastruktur aus einer Kombination von mindestens zwei der zuvor genannten Modelle zusammengesetzt ist. Beispielsweise könnte eine Private Cloud mit einer Public Cloud, welche in Lastspitzen für zusätzliche Ressourcen genutzt wird, kombiniert werden.

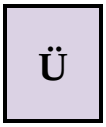
7 Übungsaufgaben

Übung 1: Vorteile durch Rechner-Virtualisierung

Diskutieren Sie, welche Vorteile durch die Verwendung von Rechner-Virtualisierung gegenüber nicht virtualisierten Systemen erzielt werden können. Welche Nachteile kann dies mit sich bringen?

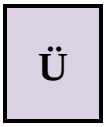
Übung 2: Rechner-Virtualisierung

Bei der Rechner-Virtualisierung ist die Virtualisierung der CPU ein zentraler Bestandteil. Warum kann dies bei manchen CPU-Architekturen ein Problem darstellen?



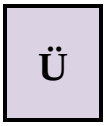
Übung 3: Virtualisierung, Emulation und Simulation

Erläutern Sie die Unterschiede zwischen Virtualisierung, Emulation und Simulation.



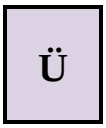
Übung 4: Bare-Metal-Virtualisierung vs. Hosted-Virtualisierung

Beschreiben Sie den Unterscheid zwischen Bare-Metal-Virtualisierung und Hosted-Virtualsierung und erläutern Sie jeweils mehrere praxisrelevante Beispielsysteme.



Übung 5: Rechner-, Betriebssystem- und Anwendungsvirtualisierung

Erläutern Sie die Unterschiede zwischen Rechner-, Betriebssystem- und Anwendungsvirtualisierung.



Ü

Übung 6: Varianten von CPU-Virtualisierung

Erläutern Sie bei CPU-Virtualisierung die Unterschiede zwischen Paravirtualisierung, Virtualisierung durch Dynamic Rewriting und hardwaregestützter Virtualisierung und diskutieren Sie den Einfluss dieser Varianten auf die Performance (Ausführungsgeschwindigkeit der Anwendungen in einer virtuellen Maschine).

Ü

Übung 7: Cloud Computing

Welche zentralen Eigenschaften sind charakteristisch für Cloud Computing gemäß der Definition des NIST?

Ü

Übung 8: Cloud-Computing-Modelle

Welche unterschiedlichen Ausprägungen von Cloud Computing kennen Sie hinsichtlich Dienstmodellen und Nutzergruppen?

Verzeichnisse

I. Abbildungen

II. Definitionen

III. Literatur

Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, und Andrew Warfield. Xen and the Art of Virtualization. *SIGOPS Oper. Syst. Rev.*, 37(5):164–177, Oktober 2003.

Bibliographisches Institut GmbH - Duden Verlag (Hrsg.). *Duden – Deutsches Universalwörterbuch*. Bibliographisches Institut, Mannheim, 6 Aufl., Oktober 2006. ISBN 3411055065.

Robert Jay Creasy. The Origin of the VM/370 Time-sharing System. *IBM J. Res. Dev.*, 25(5):483–490, September 1981.

Scott W. Devine, Edouard Bugnion, und Mendel Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture, 1998. US Patent 6,397,242.

Stuart W. Galley. PDP-10 Virtual Machines. In *Proceedings of the Workshop on Virtual Computer Systems*, S. 30–34, New York, NY, USA, 1973. ACM.

Robert P. Goldberg. Architecture of Virtual Machines. In *Proceedings of the Workshop on Virtual Computer Systems*, S. 74–112, New York, NY, USA, 1973. ACM.

IEEE Std 802.1Q. IEEE Standard for local and metropolitan area networks – bridges and bridged networks. Online verfügbar unter http://standards.ieee.org/getieee802/download/802-1Q-2014.pdf, [abgerufen am 2017-03-01], 2014.

Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, und Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, März 2008. ISSN 0146-4833.

Peter M. Mell und Timothy Grance. SP 800-145. The NIST Definition of Cloud Computing. Technical report, Gaithersburg, MD, United States, 2011.

Gerald J. Popek und Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, 17(7):412–421, Juli 1974.

John Scott Robin und Cynthia E. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. In *Proceedings of the 9th Conference on USENIX Security Symposium - Volume 9*, SSYM'00, S. 10–10, Berkeley, CA, USA, 2000. USENIX Association.

Andrew S. Tanenbaum und Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 0132392275.

VDI-Richtlinie 3633. *Simulation von Logistik-, Materialfluss- und Produktionssystemen - Grundlagen*. VDI Verlag, 2013.