

# CLOUD-NATIVE COMPUTING

*Unit 03:*

*Infrastructure as Code*

Stand: 27.03.23

# KAPITEL 7

## Infrastructure as Code



---

Nane Kratzke

### Cloud-native Computing

Software Engineering von Diensten und Applikationen  
für die Cloud

284 Seiten. E-Book inside

€ 59,99. ISBN 978-3-446-46228-1

Weitere Informationen unter: [www.hanser-fachbuch.de](http://www.hanser-fachbuch.de)

HANSER

## Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

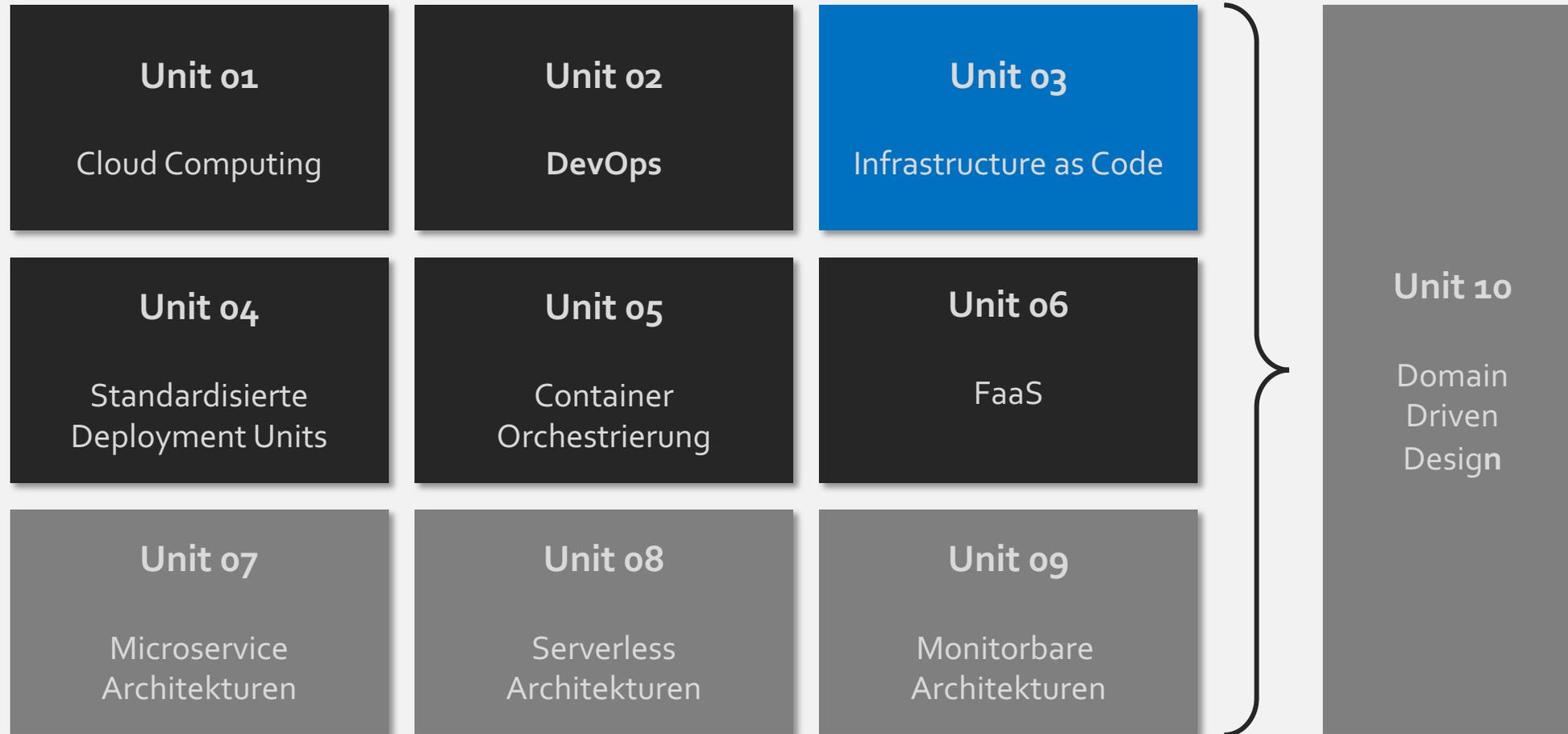
Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.



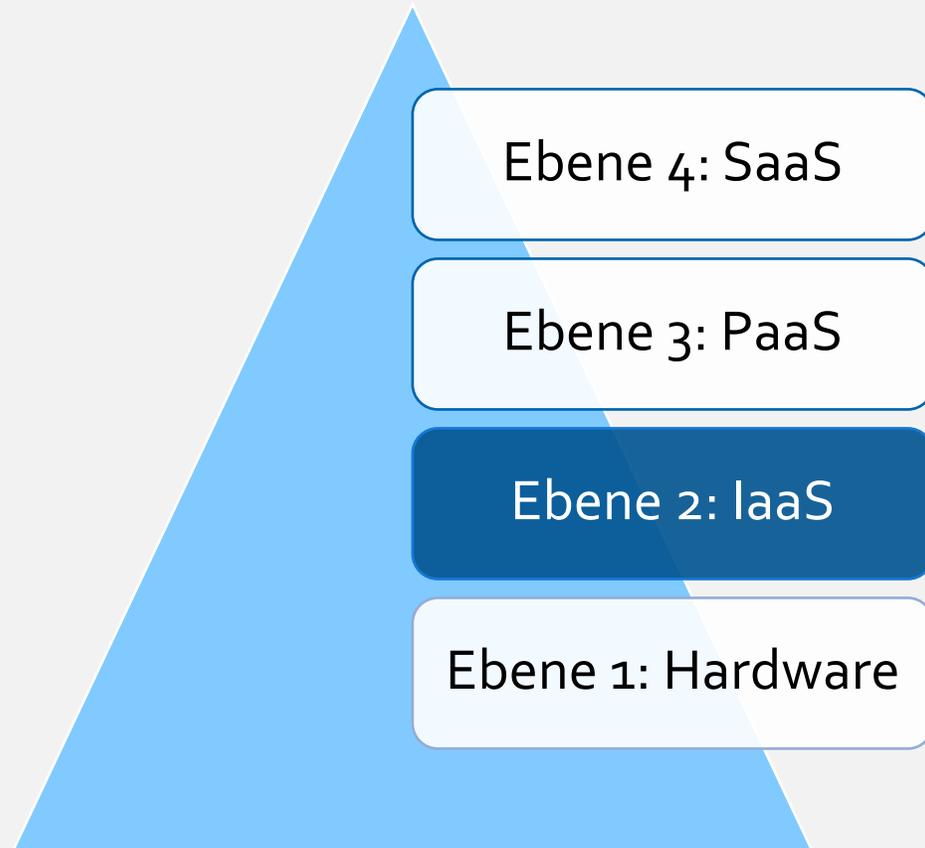
# INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



# DAS SCHICHTENMODELL DES CLOUD COMPUTINGS

Wo sind wir?



## Kunden, Endnutzer

- Anpassbare Software-Dienste
- XaaS (Everything as a Service)
- Transparente Updates

## Entwickler

- Programmierschnittstellen (APIs)
- Plattformdienste
- Abstraktion der technischen Infrastruktur

## Administratoren

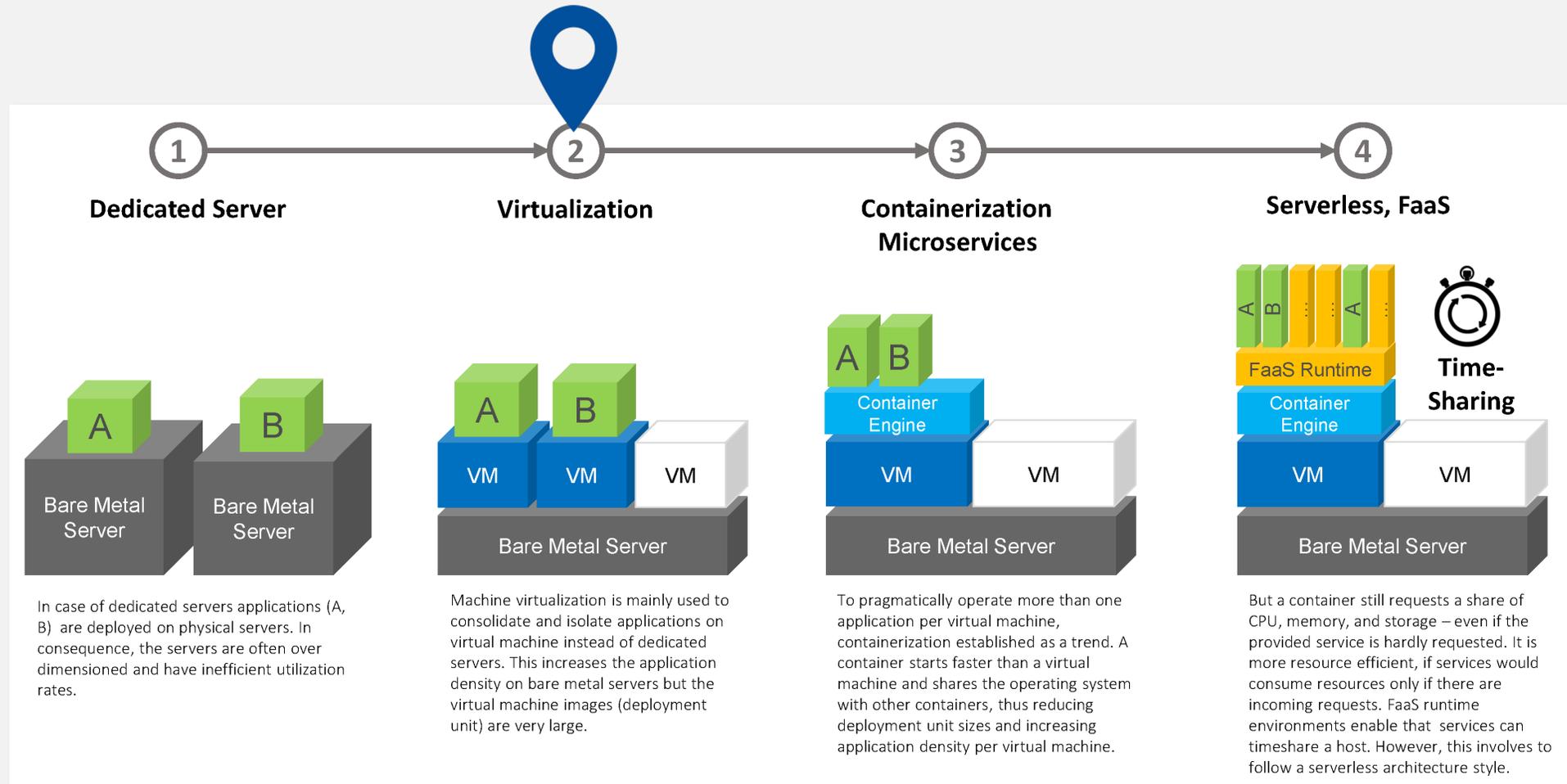
- **Elastizität**
- **Virtuelle Ressourcenpools**
- **Technische Infrastruktur (VM, Storage, Network)**

## Rechenzentrum

- Rechner
- Netzwerk
- Storage

# EINE KURZE GESCHICHTE DER CLOUD

Wo sind wir?



# INHALTE

## Virtualisierung

- Hardware-Virtualisierung (Typ-1, Typ-2)
- Betriebssystem-Virtualisierung
- Applikations-Virtualisierung
- Emulation

## Infrastructure as a Service

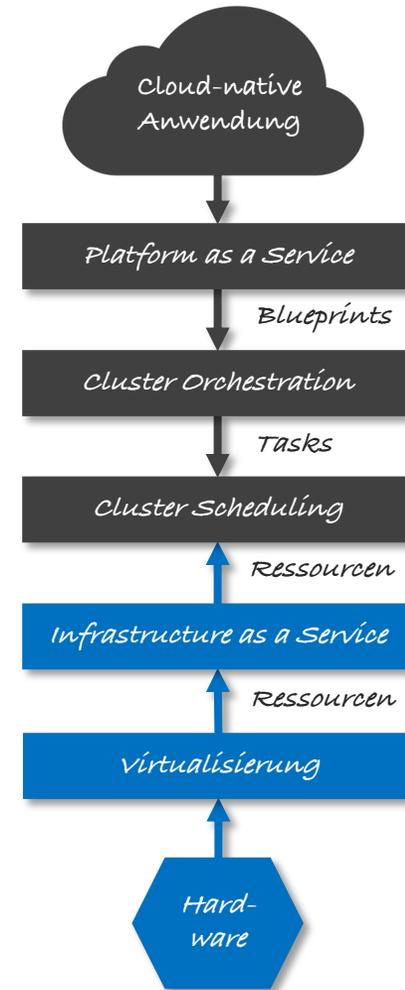
- Definition, Eigenschaften, Marktüberblick
- Private Cloud Infrastrukturen
- Public Cloud Infrastrukturen (am Bsp. des Typvertreterers AWS)

## Provisionierung in IaaS-basierte Infrastrukturen

- Historische Entwicklung und
- Ebenenmodell
- Immutable Infrastructures

## Infrastructure as Code

- Definition, Ansätze und Methoden
- Immutable Architecture
- Überblick gängiger Provisionierungswerkzeuge
- Single VM-Provisioning mittels Vagrant (Deklaratives Push-basiertes IaC)
- Multi VM-Provisioning mittels Terraform (Deklaratives Push-basiertes IaC)



# VIRTUALISIERUNG

Unter Virtualisierung versteht man die Erzeugung von virtuellen Realitäten und deren Abbildung auf die physikalische Realität.

## Zweck

- **Multiplizität:** Erzeugung mehrerer virtueller Realitäten innerhalb einer physikalischen Realität (z.B. mehrere VMs auf einem physischen Server).
- **Entkopplung:** Bindung und Abhängigkeit zur Realität auflösen (z.B. Verschiebung einer VM von einem Virtualisierungshost zu einem anderen zur Laufzeit).
- **Isolation:** Physikalische Seiteneffekte zwischen virtuellen Realitäten vermeiden (z.B. beeinträchtigt eine Endlosschleife auf einer VM nicht die Applikation auf einer anderen VM).



*Wir verstehen in diesem Modul Virtualisierung immer als Virtualisierung von Hardware- oder Software-Ressourcen.*

# VIRTUALISIERUNGSARTEN

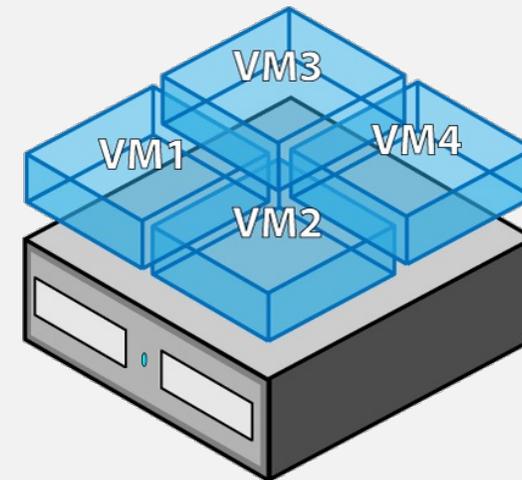
Unter Virtualisierung werden grundsätzlich verschiedene Konzepte und Technologien verstanden.

## Virtualisierung von Hardware-Infrastruktur

1. Emulation
2. Voll-Virtualisierung (Typ-2 Virtualisierung)
3. Para-Virtualisierung (Typ-1 Virtualisierung)

## Virtualisierung von Software-Infrastruktur

- Betriebssystem-Virtualisierung (Containerization)
- Anwendungs-Virtualisierung (Runtime, z.B. die Java Virtual Machine)



# VIRTUALISIERUNG

Im Cloud Computing

Entkopplung von der (physischen) Hardware (Blech) für mehr Flexibilität und Robustheit bei Ausfällen.

Normierung von Ressourcen-Kapazitäten auf heterogener und wechselnder Hardware („S/M/L/XL-Instanzen“)

Zentrale Steuerung und Bereitstellung von Rechenressourcen über die mit Virtualisierung bereitgestellten Software-Defined-Resources.

Beispiel von Instanztypen des Public Cloud Service Providers AWS  
Stand: 15.09.2020, <https://aws.amazon.com/de/ec2/instance-types/>

Instance	vCPU*	Arbeitsspeicher (GiB)	Speicher	Dedizierte EBS-Bandbreite (Mbit/s)	Netzwerkleistung
m4.large	2	8	Nur EBS	450	Mittel
m4.xlarge	4	16	Nur EBS	750	Hoch
m4.2xlarge	8	32	Nur EBS	1.000	Hoch
m4.4xlarge	16	64	Nur EBS	2.000	Hoch
m4.10xlarge	40	160	Nur EBS	4.000	10 Gigabit
m4.16xlarge	64	256	Nur EBS	10 000	25 Gigabit

# HARDWARE-VIRTUALISIERUNG

*Was wird virtualisiert?*

## Prozessor

- Virtuelle Rechenkerne (vCPU)
- Dispatching von Prozessor-Befehlen auf echte Rechnerkerne

## Hauptspeicher

- Virtuelle Hauptspeicher-Partition
- Management der realen Repräsentation (im RAM, Memory-Ballooning)

## Netzwerk

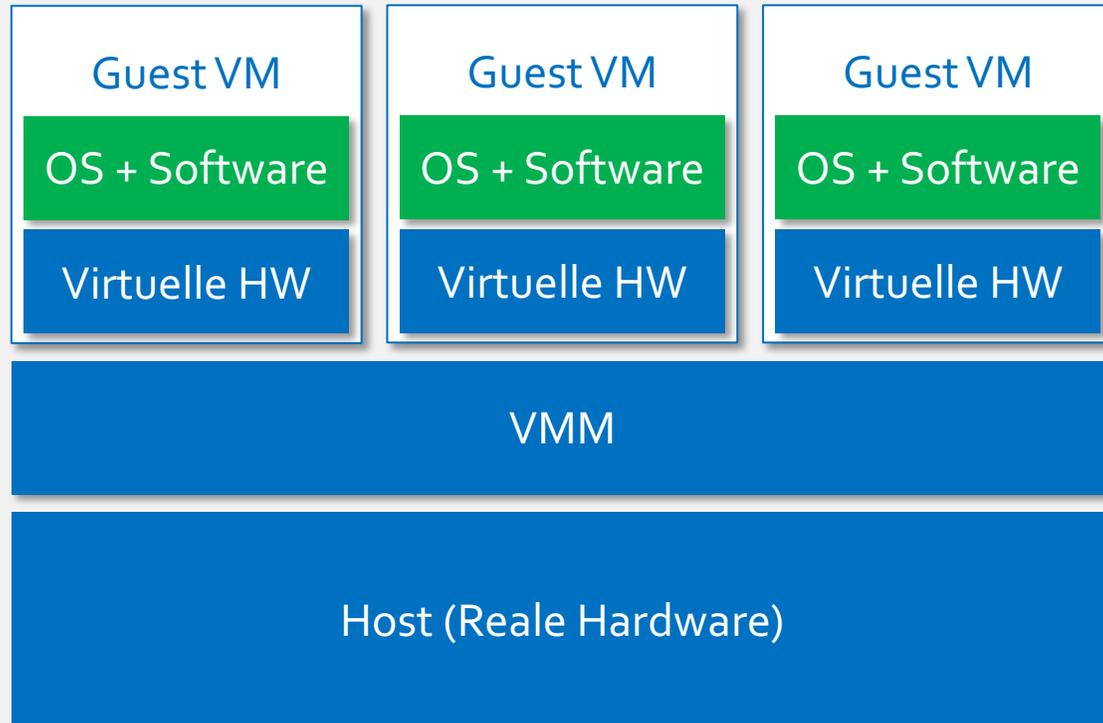
- Virtuelle Netzwerkschnittstellen und virtuelle Netzwerk-Infrastrukturen (VLAN)
- Brücken zwischen virtuellen und realen Netzen
- Firewallregeln

## Storage

- Virtuelle Festplattenlaufwerke (Blockstorage).
- Abbildung auf Dateien im realen Dateisystem.
- Virtuelle Storage Area Networks (Aufteilung der Daten eines virtuellen Laufwerks auf viele Storage Einheiten).

# HARDWARE-VIRTUALISIERUNG

Wie funktioniert HW-Virtualisierung?



**Host:** Der physikalische Rechner, der eine oder mehrere VM ausführt und entsprechende HW-Ressourcen zur Verfügung stellt.

**Guest:** Eine laufende (oder lauffähige) virtuelle Maschine (VM).

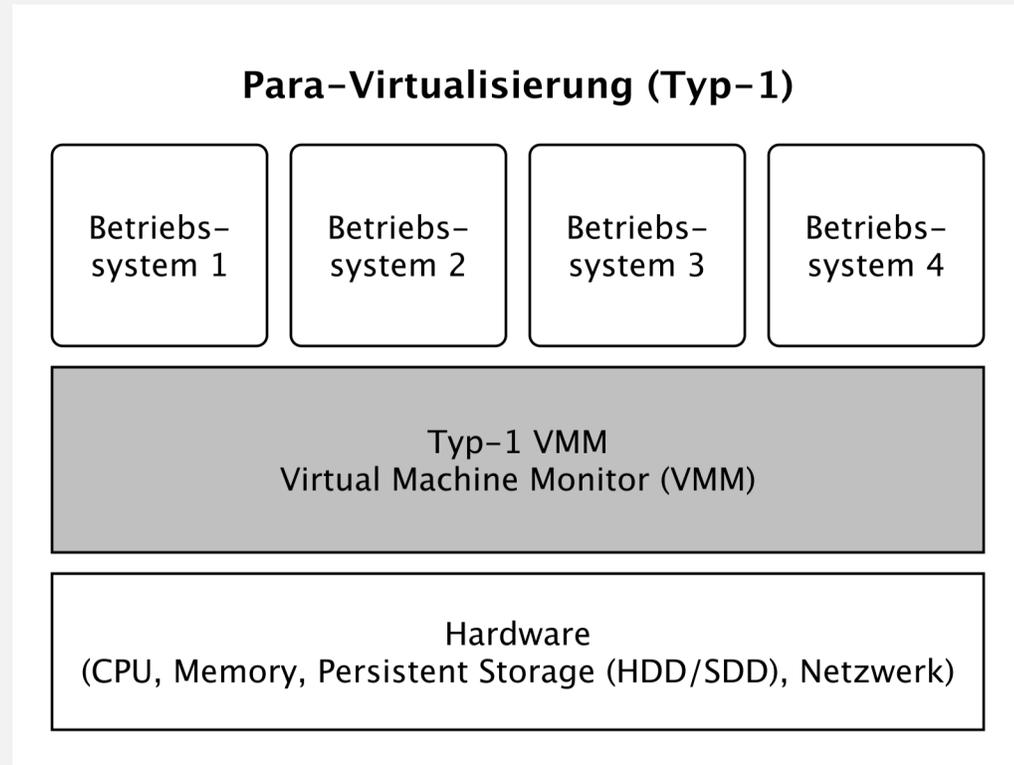
**VMM** (Virtual Machine Monitor): Die Steuerungssoftware zur Verwaltung von Guests und Host-Ressourcen.

- Durch Hardware-Virtualisierung werden die Ressourcen eines Rechnersystems aufgeteilt und von mehreren Betriebssystem-Instanzen genutzt.
- Anforderungen der Betriebssystem-Instanzen werden von der Virtualisierungssoftware (Virtual Machine Monitor, VMM) abgefangen und auf die reale vorhandene Hardware umgesetzt.

# HARDWARE-VIRTUALISIERUNG

## Para-Virtualisierung (Typ – 1)

- Der Hypervisor läuft direkt auf der verfügbaren Hardware. Er entspricht somit einem auf Virtualisierung spezialisiertem Betriebssystem.
- Das Gast-Betriebssystem muss um virtuelle Treiber ergänzt werden, um mit dem Hypervisor interagieren zu können.
- Wegen der erforderlichen Hypervisortreiber können nicht beliebige Guest OS auf Hypervisoren laufen.
- Der Hypervisor nutzt die Treiber eines Host-Betriebssystems, um auf die reale Hardware zuzugreifen. Damit braucht im Hypervisor nicht aufwändig eigene Treiber implementiert zu werden.



Leistungsverlust:  
Ca. 2-3%



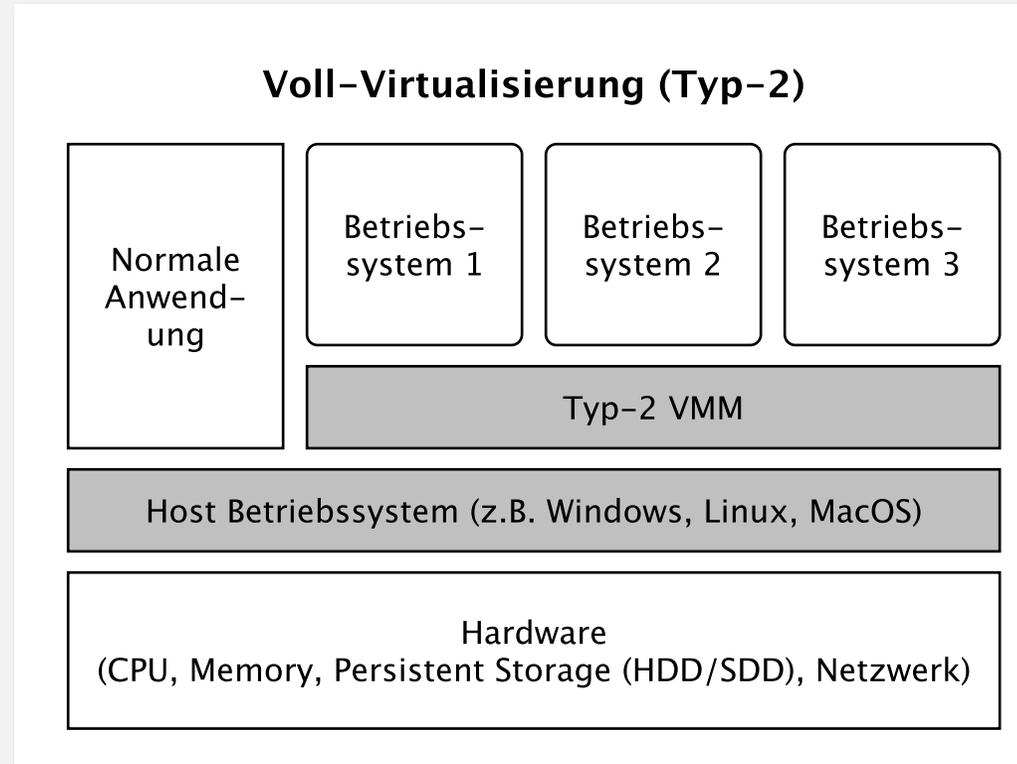
Microsoft  
Hyper-V



# HARDWARE-VIRTUALISIERUNG

## Voll-Virtualisierung (Typ – 2)

- Jedem Gastbetriebssystem steht ein eigener virtueller Rechner mit virtualisierten Ressourcen (CPU, RAM, Disks, Netzwerkkarten, etc.) zur Verfügung.
- Der VMM läuft hosted als Anwendung unter dem Host-Betriebssystem.
- Der VMM verteilt die Hardwareressourcen des Rechners an die VMs.
- Teilw. muss die VMM Hardware emulieren, die nicht für den gleichzeitigen Zugriff mehrere Betriebssysteme ausgelegt ist (z.B. Netzwerkkarten, Grafikkarten).



Leistungsverlust:  
Ca. 5-10%



VirtualBox

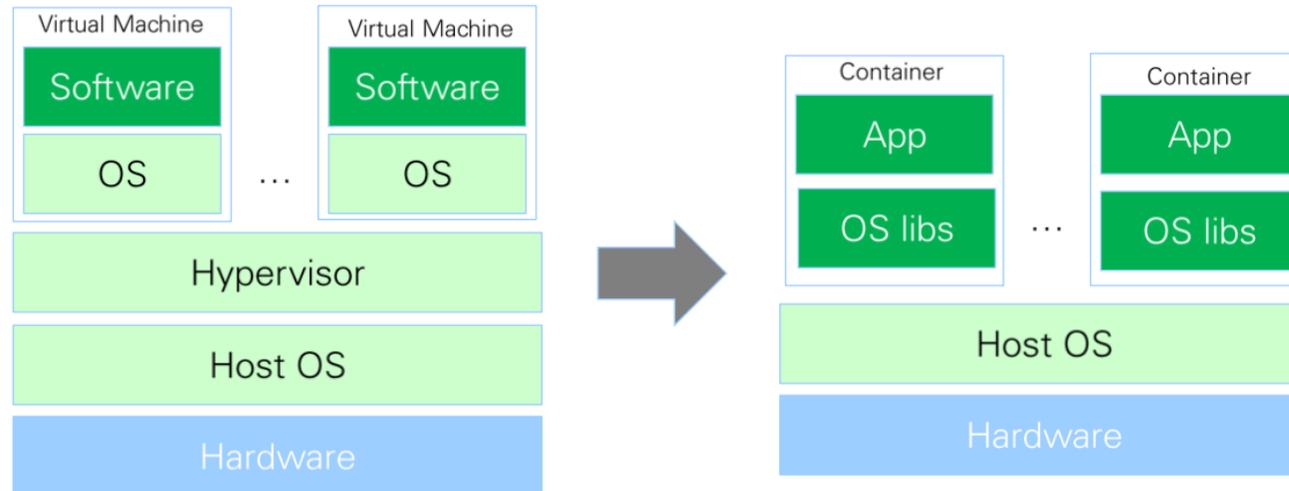


# BETRIEBSSYSTEM-VIRTUALISIERUNG

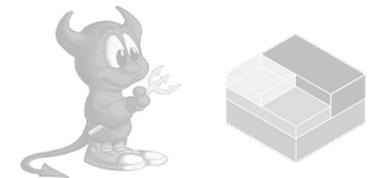
Leichtgewichtiger Virtualisierungsansatz mittels OS-Containern

Es gibt keinen Hypervisor. Jede Applikation läuft direkt als Prozess im Host-Betriebssystem.

Dieser Prozess ist jedoch mittels OS-Mechanismen isoliert.



Leistungsverlust:  
CPU-/RAM-Overhead in der Regel nicht messbar (~0%)



OpenVZ

SOLARIS

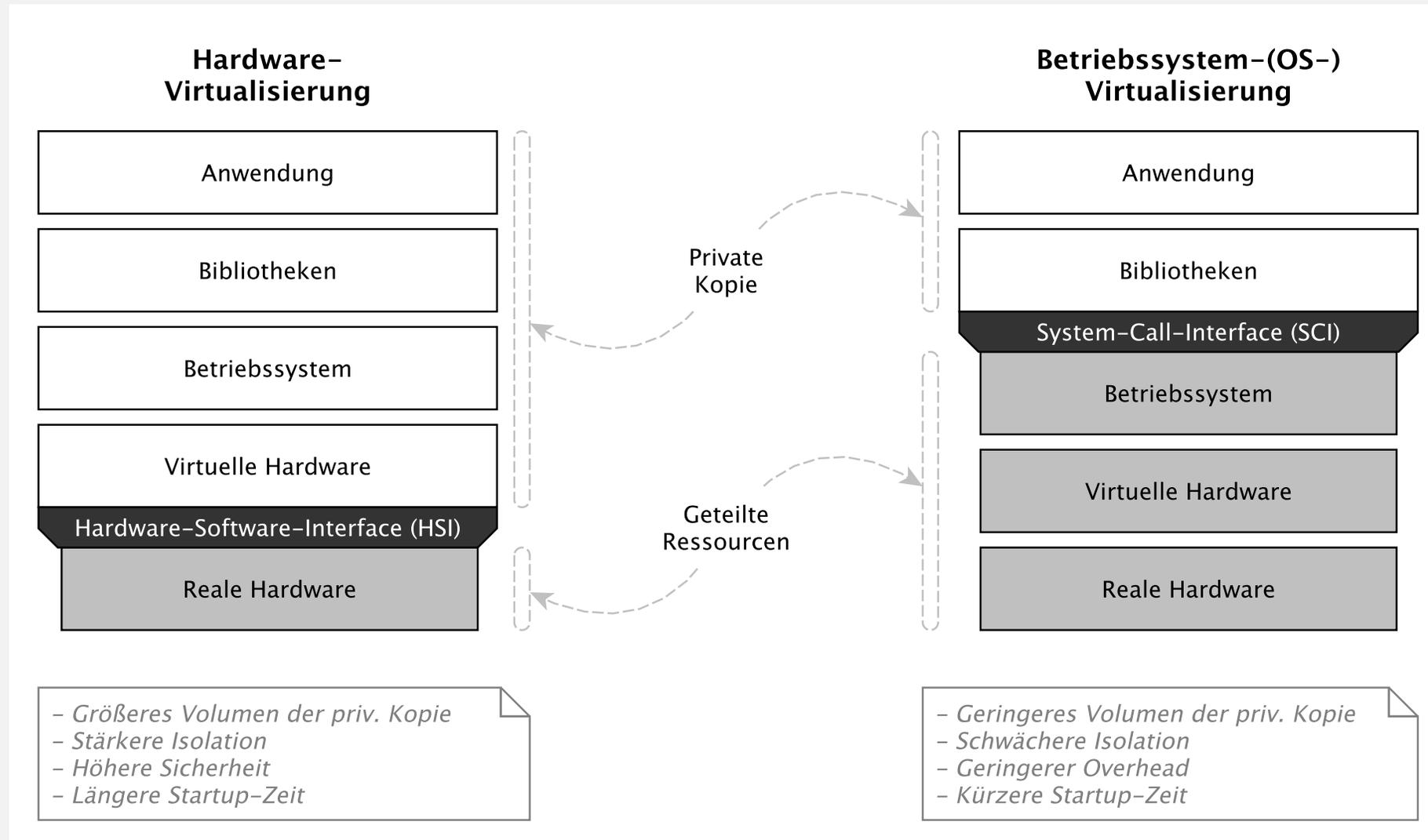
- Free BSD Jails (2000)
- Solaris Zones (2005)
- Linux OpenVZ (2005)
- Linux LXC (2008)
- und mehr ...

- Isolation des Prozesses durch Namespaces (bzgl. CPU, RAM, Disk I/O) und Containments
- Isoliertes Dateisystem
- Eigene Netzwerkschnittstelle
- Startup-Zeit = Startdauer für den ersten Prozess (kein Boot des OS erforderlich!)

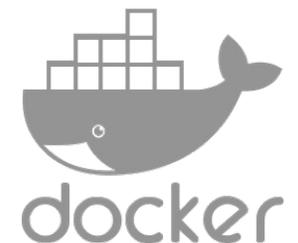
Docker v0.0.1  
release: 2013

# HARDWARE- VS. OS-VIRTUALISIERUNG

Trade-off zwischen Isolation (Sicherheit) und Ressourcen-Effizienz



Mit OS-Virtualisierung (Containern) werden wir uns noch intensiv in **Unit 04** und **Unit 05** befassen.



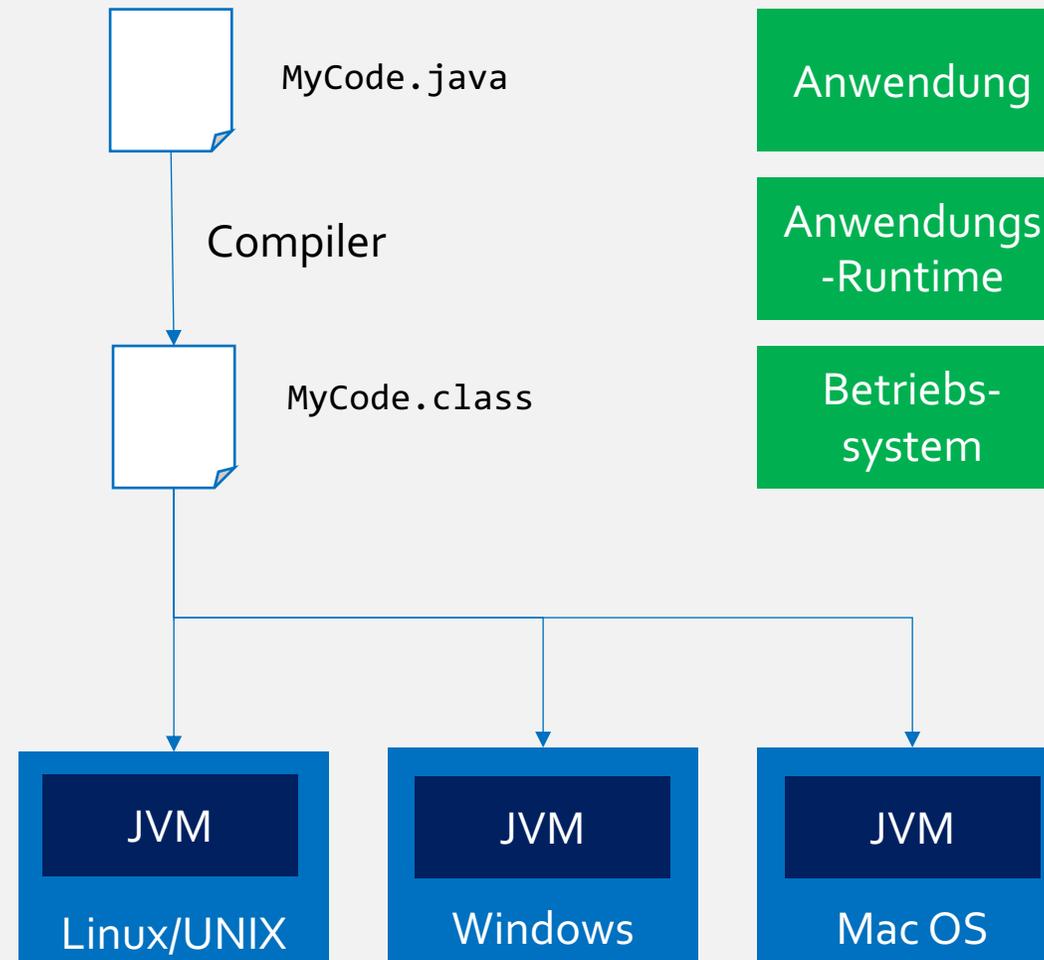
# EMULATION VS ANWENDUNGS-VIRTUALISIERUNG

## Anwendungs-Virtualisierung

Anwendungsvirtualisierung stellt Anwendungen eine Programmierschnittstelle und eine Laufzeitumgebung (Runtime) zur Verfügung, die (möglichst) komplett vom darunter liegenden Betriebssystem entkoppelt.

Zweck: OS-unabhängige Anwendungen.

Beispiel: Java Virtual Machine



Diese Art der Virtualisierung wird mittlerweile von vielen modernen Programmiersprachen genutzt.

Die damit einhergehenden Leistungsverluste werden billiger in Kauf genommen (siehe Java).

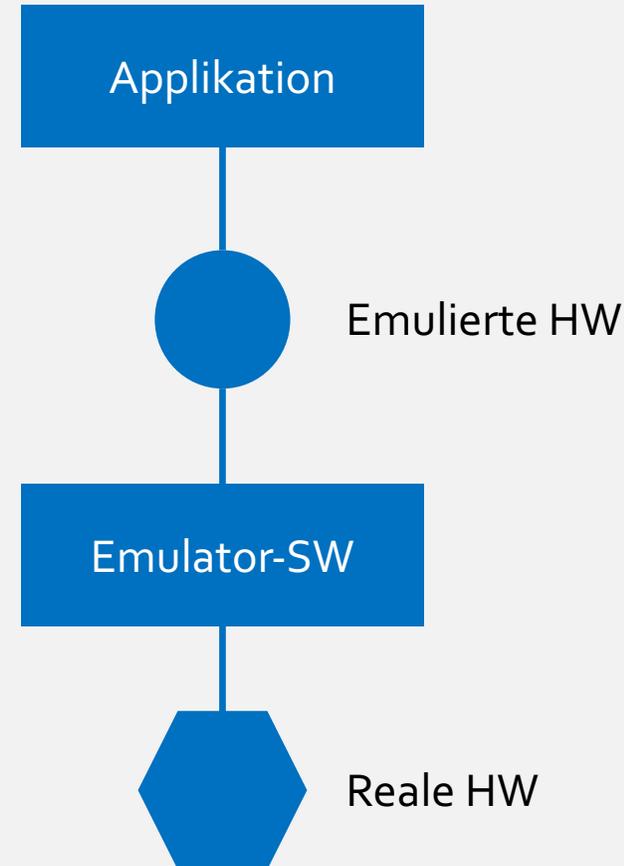
# EMULATION VS ANWENDUNGS-VIRTUALISIERUNG

## Emulation

Emulation bildet die Hardware eines nicht vorhandenen oder nicht kompatiblen Rechnersystems oder Teile eines entsprechenden Rechnersystems nach.

Hardware wird durch Software „simuliert“.

Z.B. Gameboy Emulator im Browser oder Smartphone.



Substantielle  
Leistungs-  
verluste.

Daher im  
Cloud  
Computing  
nicht wirklich  
gebräuchlich.

## Virtualisierung

- Hardware-Virtualisierung (Typ-1, Typ-2)
- Betriebssystem-Virtualisierung
- Applikations-Virtualisierung
- Emulation

## Infrastructure as a Service

- Definition, Eigenschaften, Marktüberblick
- Private Cloud Infrastrukturen
- Public Cloud Infrastrukturen (am Bsp. des Typvertreterers AWS)

## Provisionierung in IaaS-basierte Infrastrukturen

- Historische Entwicklung und
- Ebenenmodell
- Immutable Infrastructures

## Infrastructure as Code

- Definition, Ansätze und Methoden
- Immutable Architecture
- Überblick gängiger Provisionierungswerkzeuge
- Single VM-Provisioning mittels Vagrant (Deklaratives Push-basiertes IaC)
- Multi VM-Provisioning mittels Terraform (Deklaratives Push-basiertes IaC)

# INFRASTRUCTURE AS A SERVICE

## Definition und Eigenschaften

Unter IaaS versteht man ein Geschäftsmodell, das entgegen dem klassischen Kauf von Rechnerinfrastruktur vorsieht, diese je nach Bedarf anzumieten und freizugeben.

### Eigenschaften:

- **Ressourcen-Pools:** Verfügbarkeit von scheinbar unbegrenzten Ressourcen
- **Elastizität:** Dynamische Zuweisung von Ressourcen bei Bedarf
- **Pay-as-you-go Modell:** Abgerechnet werden nur bereitgestellte Ressourcen

### Ressourcen-Typen

- **Rechenleistung:** Rechenknoten mit CPU, RAM und HD
- **Speicher:** Storage Kapazitäten als Dateisystem-Mounts oder Datenbanken
- **Netzwerk(-dienste)** wie DNS, DHCP, VPN, CDN, Load Balancer

### Infrastruktur-Dienste

- **Monitoring:** Log-Konsolidierung, Performance, Speicherverbrauch
- **Ressourcen-Management:** Auto-Skalierung, Budgets, Ressourcenverbrauchs-Warnungen, konsolidierte Abrechnungen von mehreren Accounts

# INFRASTRUCTURE AS A SERVICE

## Elastizitätsarten

### Nachfrageelastizität:

Die allokierten Ressourcen steigen/sinken mit der Nachfrage

- **Pseudo-Elastizität:** Schnelle Anforderung, kurze Kündigungsfrist (Reduktion der Bindungsdauer)
- **Echtzeit-Elastizität:** Allokation und Freigabe von Ressourcen innerhalb von Minuten (ggf. Sekunden). Automatisierter Prozess mit manuellen Triggern oder nach Zeitplan.
- **Selbstadaptive Elastizität:** Automatische Allokation und Freigabe von Ressourcen (Auto-Scaling) in Echtzeit auf Basis von Regeln und Metriken.

*Eher Nutzersicht*

⇒ *Ressourcen-  
Effizienz*

### Angebotselastizität:

Die allokierten Ressourcen steigen/sinken mit Angebot und Nachfrage

- Batch-getriebenes Grid: Rechner werden allokiert, wenn Preisschwelle unterschritten wird.
- Es sind auch Varianten verfügbar, bei denen man für freie Ressourcen bieten kann.

*Eher IaaS-  
Providersicht*

⇒ *Kostenmodelle*  
⇒ *Auktionsmodelle  
(Kundensicht)*

### Einkommenselastizität:

Die allokierten Ressourcen steigen / sinken mit dem Einkommen / Budget / Umsatz.

*Eher P/SaaS-  
Providersicht*

⇒ *Wachstum*

# INFRASTRUCTURE AS A SERVICE

*Der momentane IaaS Markt*

**Table 1. Worldwide IaaS Public Cloud Services Market Share, 2020-2021 (Millions of U.S. Dollars)**

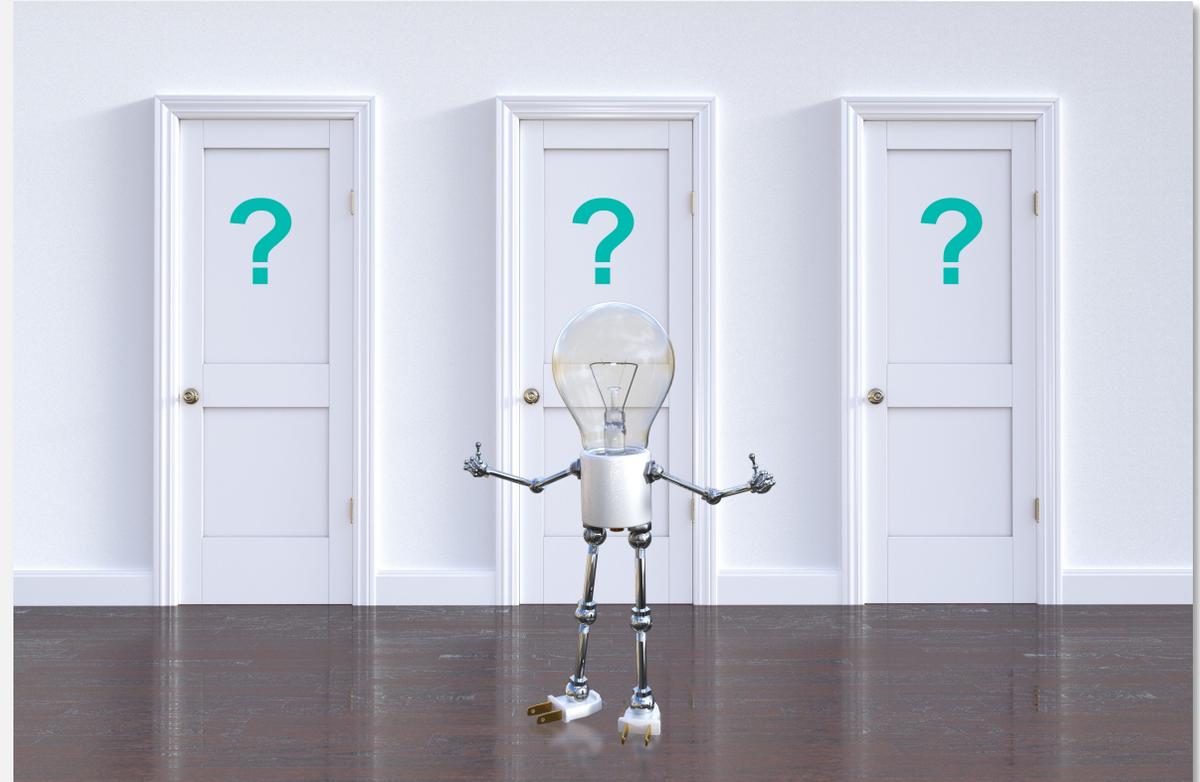
<b>Company</b>	<b>2021 Revenue</b>	<b>2021 Market Share (%)</b>	<b>2020 Revenue</b>	<b>2020 Market Share (%)</b>	<b>2020-2021 Growth (%)</b>
Amazon	35,380	38.9	26,201	40.8	35.0
Microsoft	19,153	21.1	12,659	19.7	51.3
Alibaba	8,679	9.5	6,117	9.5	41.9
Google	6,436	7.1	3,932	6.1	63.7
Huawei	4,190	4.6	2,681	4.2	56.3
Others	17,056	18.8	12,697	19.8	34.3
<b>Total</b>	<b>90,894</b>	<b>100.0</b>	<b>64,286</b>	<b>100.0</b>	<b>41.4</b>

Source: Gartner (June 2022)

# INFRASTRUCTURE AS A SERVICE

## Auswahlkriterien

- Unterstütze Cloud Variante  
(Private/Public/Community/Hybrid Cloud)
- Zuverlässigkeit / Verfügbarkeit
- Sicherheit und Datenschutz
- Gesetzliche Konformität (DSGVO!)
- Vorhersagbare und stabile Performance
- Preismodell: Fixe und flexible Kosten
- Skalierbarkeit: Grenzen, Automatismen und Reaktionszeiten
- Lock-In der Daten und Anwendungen:  
Offene APIs und Standards
- Haftung und Support



# INFRASTRUCTURE AS A SERVICE

## Service Level Agreements

Availability %	Downtime per Year	Downtime per Month	Downtime per Week
99.9% (three nines)	8.76 hours	43.2 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.6 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	.0605 seconds

Eine Service Level Agreement (SLA) ist ein Vertrag mit Zuverlässigkeitszusagen für Ressourcen und Dienste.

## Beispiel: AWS S3 (Object Storage)

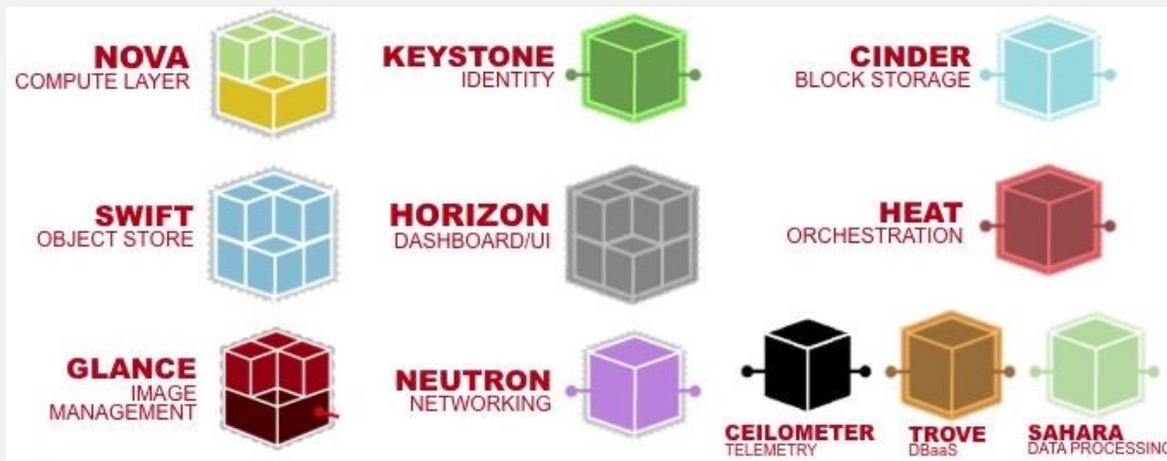
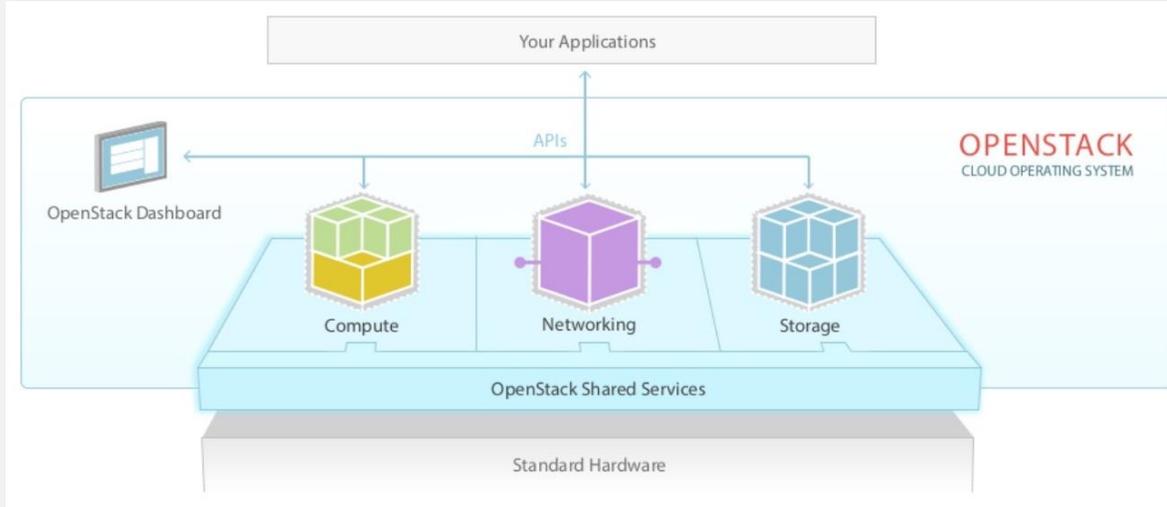
### Service Commitment

AWS will use commercially reasonable efforts to make Amazon S3 available with a Monthly Uptime Percentage (defined below) of at least 99.9% during any monthly billing cycle (the "Service Commitment"). In the event Amazon S3 does not meet the Service Commitment, you will be eligible to receive a Service Credit as described below.

Monthly Uptime Percentage	Service Credit Percentage
Equal to or greater than 99% but less than 99.9%	10%
less than 99%	25%

# OPENSTACK (PRIVATE IAAS)

*De-facto Standard für Open-Source Private IaaS Clouds*



Wurde maßgeblich von RackSpace und der NASA initiiert.

Erstes Release Oktober 2010.

Apache Lizenz

Vielzahl der „klassischen“ IT-Player sind Teil der OpenStack-Community (SAP, IBM, vmWare, HP, Oracle, Cisco, etc.)

Sehr aktives Projekt mit > 400 aktiven Committern.

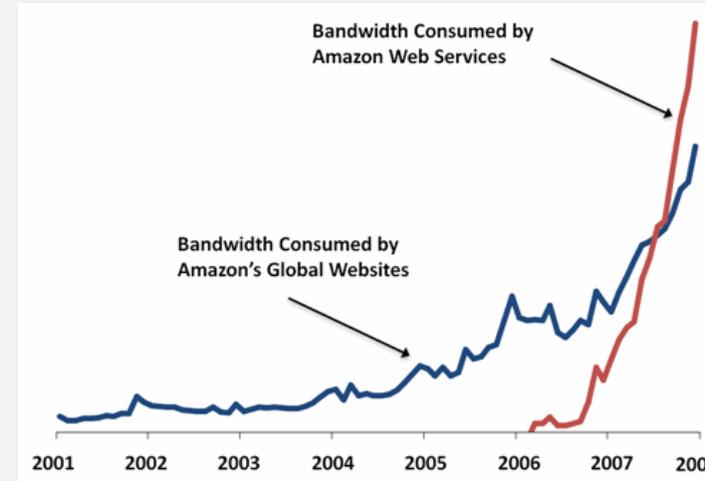
Ausgelegt eher als Framework denn als fertiges System für IaaS-Clouds.

Extrem hohe Installations- und Konfigurationskomplexität.

# AMAZON EC<sub>2</sub> (PUBLIC IAAS)

*EC2 ist der Kerndienst von AWS (AWS hostet seine eigenen Dienste mit EC2)*

- Start innerhalb von Amazon im Jahr 2001
- Öffentliche Beta ab August 2006
- Ab Mitte 2008 mehr Bandbreite durch Dritte in der Cloud konsumiert, als durch Amazon Websites (Shop)
- Produktionsreife ab Oktober 2008
- 2005 bis 2012 ca. 12 Mrd. \$ Investment in die Infrastruktur
- 2015: 1,5 bis 2 Mio. Server in 10 globalen Rechenzentren.
- On-Demand-, Reserved-, und Spot-Instanzen in verschiedenen Größen



*Wir nehmen AWS als TYP-  
Repräsentant, um zu zeigen, wie  
alle Public Cloud Provider  
üblicherweise ihr Geschäft zum  
Kunden hin organisieren.*

# AMAZON EC2

## Globale Verteilung

### # Region und Anzahl der Availability Zones

<b>USA Ost</b> Nord-Virginia (6), Ohio (3)	<b>China</b> Peking (2)
<b>USA West</b> Nordkalifornien (3), Oregon (3)	<b>Europa</b> Frankfurt (3), Irland (3), London (2)
<b>Asien-Pazifik</b> Mumbai (2), Seoul (2), Singapur (2), Sydney (3), Tokio (3)	<b>Südamerika</b> São Paulo (3)
<b>Kanada</b> Zentral (2)	<b>AWS GovCloud (US- West)</b> (2)

### ○ Neue Region (in Kürze verfügbar)

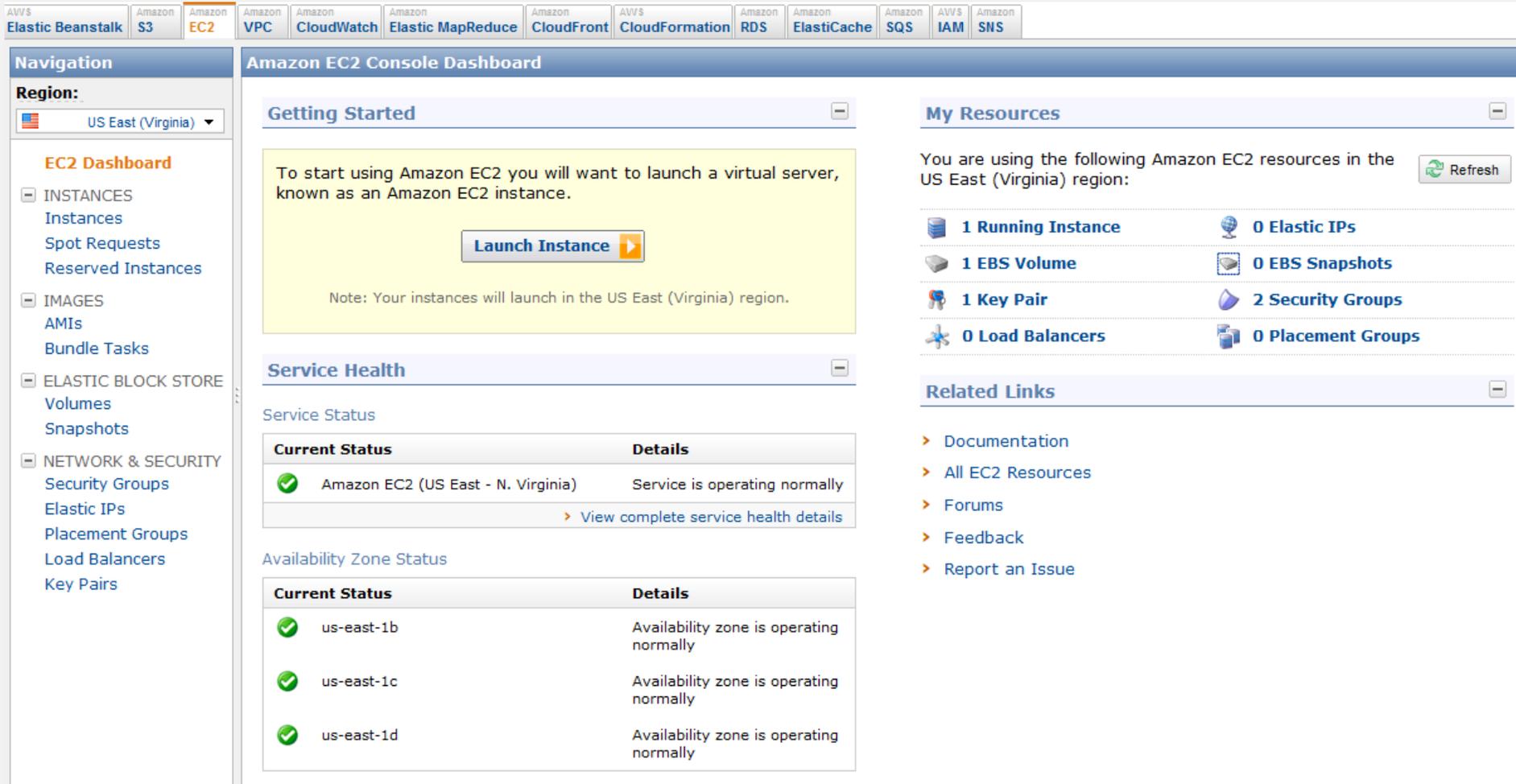
Bahrain  
China  
Frankreich  
Hongkong  
Schweden  
AWS GovCloud (US-  
East)



Darstellung kann veraltet sein.

# AMAZON EC2

## Web-basierte Management Console



The screenshot shows the Amazon EC2 Console Dashboard for the US East (Virginia) region. The navigation pane on the left includes sections for EC2 Dashboard (Instances, Spot Requests, Reserved Instances), IMAGES (AMIs, Bundle Tasks), ELASTIC BLOCK STORE (Volumes, Snapshots), and NETWORK & SECURITY (Security Groups, Elastic IPs, Placement Groups, Load Balancers, Key Pairs). The main content area is divided into three sections: Getting Started, Service Health, and My Resources. The Getting Started section contains a yellow box with instructions to launch a virtual server and a 'Launch Instance' button. The Service Health section shows the current status of Amazon EC2 (US East - N. Virginia) as 'Service is operating normally' and lists the status of three availability zones (us-east-1b, us-east-1c, us-east-1d), all of which are operating normally. The My Resources section displays a summary of resources: 1 Running Instance, 0 Elastic IPs, 1 EBS Volume, 0 EBS Snapshots, 1 Key Pair, 2 Security Groups, 0 Load Balancers, and 0 Placement Groups. A 'Refresh' button is provided for the resource summary. Below the resource summary, there are links for Documentation, All EC2 Resources, Forums, Feedback, and Report an Issue.

Navigation

Region: US East (Virginia)

EC2 Dashboard

- INSTANCES
  - Instances
  - Spot Requests
  - Reserved Instances
- IMAGES
  - AMIs
  - Bundle Tasks
- ELASTIC BLOCK STORE
  - Volumes
  - Snapshots
- NETWORK & SECURITY
  - Security Groups
  - Elastic IPs
  - Placement Groups
  - Load Balancers
  - Key Pairs

Amazon EC2 Console Dashboard

### Getting Started

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

[Launch Instance](#)

Note: Your instances will launch in the US East (Virginia) region.

### Service Health

Service Status

Current Status	Details
 Amazon EC2 (US East - N. Virginia)	Service is operating normally <a href="#">View complete service health details</a>

Availability Zone Status

Current Status	Details
 us-east-1b	Availability zone is operating normally
 us-east-1c	Availability zone is operating normally
 us-east-1d	Availability zone is operating normally

### My Resources

You are using the following Amazon EC2 resources in the US East (Virginia) region: [Refresh](#)

- 1 Running Instance
- 0 Elastic IPs
- 1 EBS Volume
- 0 EBS Snapshots
- 1 Key Pair
- 2 Security Groups
- 0 Load Balancers
- 0 Placement Groups

### Related Links

- > Documentation
- > All EC2 Resources
- > Forums
- > Feedback
- > Report an Issue

Darstellung kann veraltet sein.

# AMAZON EC2

## Sicherheitsaspekte

- Zertifiziert nach ISO 27001 (Empfehlung BSI). In europäischen Datacentern den EU-Datenschutzrichtlinien unterworfen. Amazon ist ebenso global dem US Patriot Act unterworfen.
- AWS unterstützt diverse Sicherheitsstandards und Compliance-Zertifizierungen, darunter PCI-DSS (Payment Cards), HIPAA/HITECH (Health Insurance), GDPR (DSGVO), FIPS 140-2 (Federal Information Processing) und NIST 800-171 (Unclassified Information in Nonfederal Systems).
- Daten und Instanzen können global auf alle Rechenzentren verteilt werden. Jedes dieser Rechenzentren besteht aus mehreren Verfügbarkeitszonen, die ein in sich geschlossenes Rechen-Cluster darstellen.
- Jede EC2-Instanz muss einer Security Group zugeordnet sein. Eine Security Group ist die Konfiguration einer Inbound-Firewall für Instanzen.
- Der Zugriff auf die EC2-Administrationsfunktionen können über den zentralen IAM-Service gesteuert werden (Benutzermanagement und API-Zugriff per Schlüssel und Zertifikaten).
- Zugriff auf Linux Instanzen erfolgt per default mittels Schlüssel-basiertem SSH (kein Passwort).



# AMAZON WEB SERVICES

Auf Basis des EC2 Service bietet AWS ein kontinuierlich wachsendes Cloud Service Angebot

### Entdecken Sie unsere Produkte

**Analysen**

**Unternehmensanwendungen**

**Anwendungsintegration**

**Datenverarbeitung**

**AR und VR**

**Container**

**AWS-Kostenmanagement**

**Kundeneinbindung**

**Blockchain**

**Datenbank**

**Amazon EC2**  
Virtuelle Server in der Cloud

**Amazon EC2 Auto Scaling**  
Rechenkapazität skalieren, um Anforderungen zu erfüllen

**Amazon Lightsail**  
Virtual Private Server starten und verwalten

**AWS Batch**  
Ausführen von Batch-Jobs beliebiger Größe

**AWS Elastic Beanstalk**  
Web-Apps ausführen und verwalten

**AWS Lambda**  
Code ausführen, ohne an Server zu denken

**AWS Outposts**  
AWS-Infrastruktur lokal ausführen

**AWS Serverless Application Repository**  
Erkunden, Bereitstellen und Veröffentlichung von serverlosen Anwendungen

**AWS Snow-Familie**  
Physische Geräte zum Aggregieren und Verarbeiten von Daten an Edge-Standorten und zum anschließenden Übertragen in AWS

**AWS Wavelength**  
Bereitstellen von Anwendungen mit extrem niedriger Latenz für 5G-Geräte

**VMware Cloud on AWS**  
Erstellen einer Hybrid-Cloud ohne angepasste Hardware

**Entwickler-Tools**

**Endbenutzer-Datenverarbeitung**

**Game Tech**

**Internet of Things**

**Machine Learning**

**Verwaltung und Governance**

**Medianservices**

**Migration und Übertragung**

**Mobil**

**Netzwerk und Inhaltsbereitstellung**

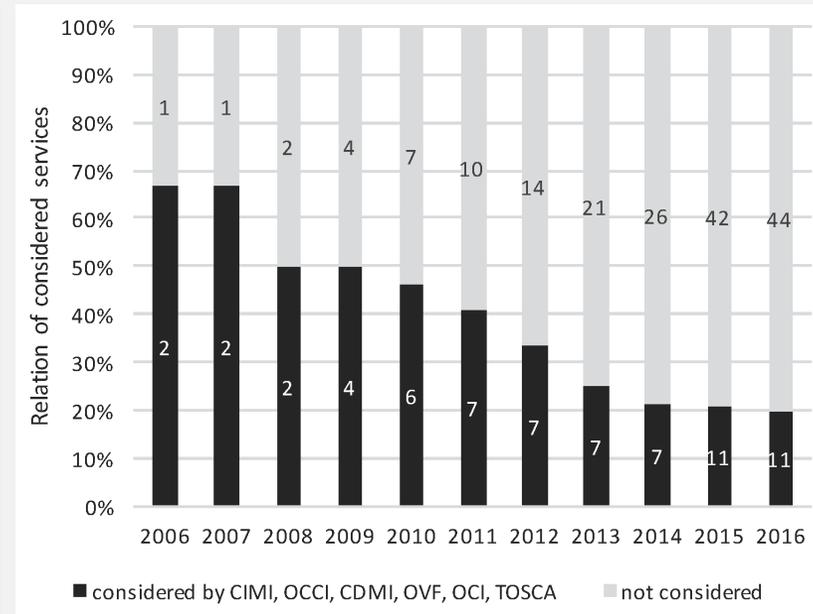
**Quantentechnologien**

**Robotik**

**Satellit**

**Sicherheit, Identität und Compliance**

**Storage**



## Analyse im Jahr 2017 am Beispiel von AWS:

Verhältnis von Services denen Standards zugeordnet werden können zu Services die in Standards nicht einmal auftauchen

**Quelle:** Kratzke, N. A Brief History of Cloud Application Architectures. Appl. Sci. 2018, 8, 1368.

Quelle: AWS, Darstellung kann veraltet sein.

## Virtualisierung

- Hardware-Virtualisierung (Typ-1, Typ-2)
- Betriebssystem-Virtualisierung
- Applikations-Virtualisierung
- Emulation

## Infrastructure as a Service

- Definition, Eigenschaften, Marktüberblick
- Private Cloud Infrastrukturen
- Public Cloud Infrastrukturen (am Bsp. des Typvertreterers AWS)

## Provisionierung in IaaS-basierte Infrastrukturen

- Historische Entwicklung und
- Ebenenmodell
- Immutable Infrastructures

## Infrastructure as Code

- Definition, Ansätze und Methoden
- Überblick gängiger Provisionierungswerkzeuge
- Single VM-Provisioning mittels Vagrant (Deklaratives Push-basiertes IaC)
- Multi VM-Provisioning mittels Terraform (Deklaratives Push-basiertes IaC)

# PROVISIONIERUNG

*Automatisierte Bereitstellung von IT-Ressourcen im Verlaufe der letzten Jahrzehnte*

## **Ohne Virtualisierung (vor 2000):**

- Manuelles Installieren von Betriebssystem auf dedizierter Hardware
- Manuelle Installation von Infrastruktur-Software
- Manuelle / teilautomatisierte / automatisierte Installation der Anwendungssoftware per Installer, Skript, proprietäre Lösungen

## **Virtualisierung einzelner Maschinen (2000 – heute)**

- Manuelles Installieren von virtuellen Maschinen
- Manuelle Installation von Infrastruktur-Software
- Manuelle / teilautomatisierte / automatisierte Installation der Anwendungssoftware per Installer, Skript, proprietäre Lösungen

## **Virtualisierung in der Cloud (seit 2010)**

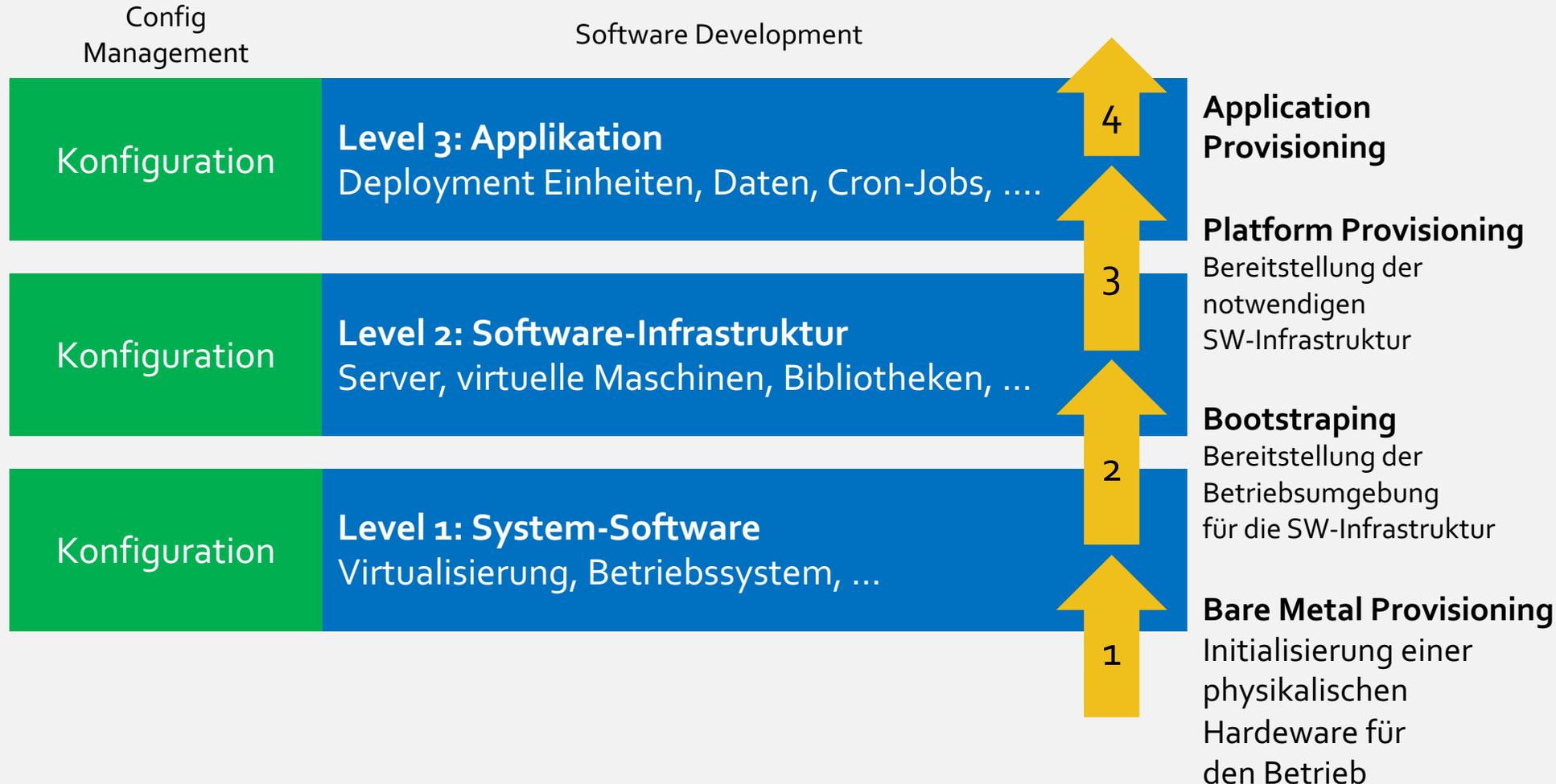
- Automatisches Bereitstellen von vorgefertigten virtuellen Maschinen und Containern
- Manuelle Installation der Infrastruktur-Software nur 1x Clone-Master-Image
- Bereitstellung einer definierten Umgebung auf Knopfdruck

## **Infrastructure as Code (2010 – heute)**

- Programmieren der Provisionierung und weiterer Betriebsprozeduren

# PROVISIONIERUNG

## Ebenenmodell

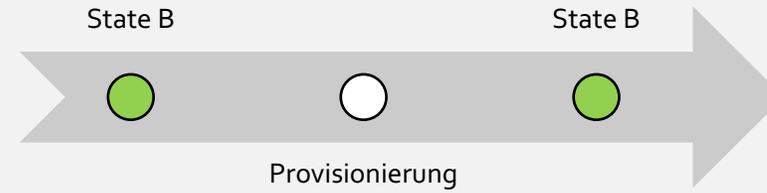


# PROVISIONIERUNG

## Konzeptionelle Überlegungen

**Systemzustand** := Gesamtheit der Software, Daten und Konfigurationen auf einem System

**Provisionierung** := Überführung eines System-Ist-Zustands in einen Ziel-Zustand



### Provisionierungsmechanismus:

1. Ausgangszustand feststellen
2. Vorbedingungen prüfen
3. Zustandsändernde Aktionen ermitteln
4. Zustandsändernde Aktionen durchführen
5. Nachbedingungen prüfen (ggf. Zustand zurücksetzen)

**Idempotenz:** Die Fähigkeit eine Aktion durchzuführen und sie dasselbe Ergebnis erzeugt, egal ob sie einmal oder mehrfach ausgeführt wird (Wiederholungen ändern den Zustand nicht).

**Konsistenz:** Nach Ausführung der Aktionen herrscht ein ein konsistenter Systemzustand. Egal ob einzelne, mehrere oder alle Aktionen gescheitert sind.

# IMMUTABLE ARCHITECTURES

## *Pets vs. Cattle*

An immutable infrastructure is another infrastructure paradigm in which servers are **never modified** after they're deployed. If something needs to be updated, fixed, or modified in any way, **new servers built from a common image with the appropriate changes are provisioned** to replace the old ones. After they're validated, they're put into use and the old ones are decommissioned.

The benefits of an immutable infrastructure include **more consistency and reliability** in your infrastructure and a **simpler, more predictable deployment process**. It mitigates or entirely **prevents** issues that are common in mutable infrastructures, like **configuration drift**.



*Server (virtuelle Maschinen)  
sind Wegwerf-Ware!*

# IMMUTABLE ARCHITECTURES

*Reduzieren zu berücksichtigende Zustände im Provisionierungsprozess*

Beliebiger  
Zustand



1. Ausgangszustand feststellen
2. Vorbedingungen prüfen
3. Zustandsändernde Aktionen ermitteln
4. Zustandsändernde Aktionen durchführen
5. Nachbedingungen prüfen (ggf. Zustand zurücksetzen)



Ziel-Zustand

Basis-  
Zustand



- ~~1. Ausgangszustand feststellen~~
- ~~2. Vorbedingungen prüfen~~
- ~~3. Zustandsändernde Aktionen ermitteln~~
4. Zustandsändernde Aktionen durchführen
5. Nachbedingungen prüfen (ggf. Zustand zurücksetzen)



Ziel-  
Zustand

# WERKZEUGE

Übersicht gängiger Provisionierungswerkzeuge

## Imperativ

Shell Scripting

Shell Abstraktion

## Deskriptiv

Zustandsautomaten

Bash  
PowerShell

Ruby  
Perl

Capistrano  
Dockerfile  
Vagrant

CFEngine

Chef  
Saltstack  
Ansible  
Otter

Puppet  
Terraform

## Virtualisierung

- Hardware-Virtualisierung (Typ-1, Typ-2)
- Betriebssystem-Virtualisierung
- Applikations-Virtualisierung
- Emulation

## Infrastructure as a Service

- Definition, Eigenschaften, Marktüberblick
- Private Cloud Infrastrukturen
- Public Cloud Infrastrukturen (am Bsp. des Typvertreeters AWS)

## Provisionierung in IaaS-basierte Infrastrukturen

- Historische Entwicklung und
- Ebenenmodell
- Immutable Infrastructures

## Infrastructure as Code

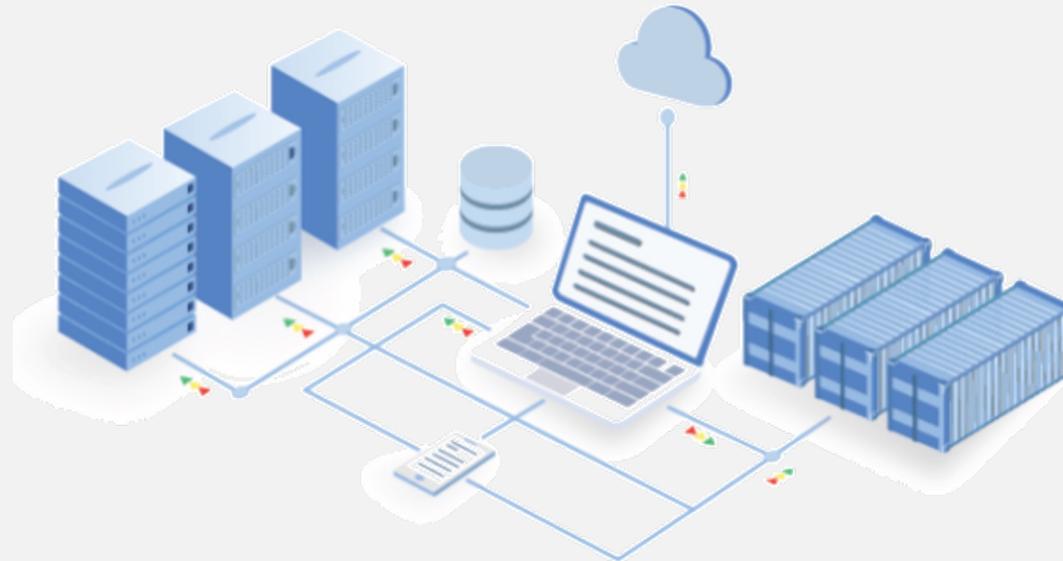
- Definition, Ansätze und Methoden
- Überblick gängiger Provisionierungswerkzeuge
- Single VM-Provisioning mittels Vagrant (Deklaratives Push-basierte IaC)
- Multi VM-Provisioning mittels Terraform (Deklaratives Push-basiertes IaC)

# INFRASTRUCTURE AS CODE

## *Definition*

Unter Infrastruktur als Code (IaC) versteht man die Verwaltung und Bereitstellung von IT-Ressourcen mittels maschinenlesbarer Definitionsdateien anstelle von physischer Hardwarekonfiguration oder interaktiven Konfigurationstools.

Mittels IaC verwaltete IT-Infrastruktur kann sowohl physische Geräte wie Bare-Metal-Server als auch virtuelle Maschinen und zugehörige Konfigurationsressourcen umfassen.



# INFRASTRUCTURE AS CODE

## *Ansätze und Methoden*

### **Deklarativer Ansatz**

- Der deklarative Provisionierungsansatz definiert den gewünschten Zustand.
- Das System führt aus, was geschehen muss, um diesen gewünschten Zustand zu erreichen.

### **Pull Methode**

Bei der Pull-Methode zieht die zu konfigurierende Maschine die Konfiguration vom steuernden Provisionierungsmechanismus.

### **Imperativer Ansatz**

- Der imperative Provisionierungsansatz definiert, wie die Infrastruktur geändert werden soll.
- Hierzu werden bestimmte Befehle definiert, die in der richtigen Reihenfolge ausgeführt werden müssen, um im gewünschten Zustand zu enden.

### **Push Methode**

Bei der Push Methode injiziert der Provisionierungsmechanismus die anzuwendene Konfiguration auf die zu provisionierende Maschine.

# INFRASTRUCTURE AS CODE

## Tool-Überblick

Tool	Released by	Method	Approach	Written in	Comments
<a href="#">Chef</a>	Chef (2009)	Pull	Declarative and imperative	Ruby	-
<a href="#">Otter</a>	Inedo	Push	Declarative and imperative	-	Windows oriented
<a href="#">Puppet</a>	Puppet (2005)	Pull	Declarative	C++ & Clojure since 4.0, Ruby	-
<a href="#">SaltStack</a>	SaltStack	Push and Pull	Declarative and imperative	Python	-
<a href="#">CFEngine</a>	Northern.tech	Pull	Declarative	C	-
<a href="#">Terraform</a>	HashiCorp (2014)	Push	Declarative	Go	-
<a href="#">Ansible</a>	Red Hat (2012)	Push	Declarative and imperative	Python	-

Provider-spezifische Ansätze wie AWS CloudFormation sind hier nicht aufgenommen.

# HARDWARE-VIRTUALISIERUNG

*Immutable Architecture am Beispiel von Vagrant und VirtualBox*



Open Source Voll-Virtualisierung  
(Typ-2 ) für Windows, Linux,  
MacOS (x86) und Solaris

**Download:**

<https://www.virtualbox.org>



Shell-basierte Automationssoftware  
für virtuelle Umgebungen auf einem  
Rechner

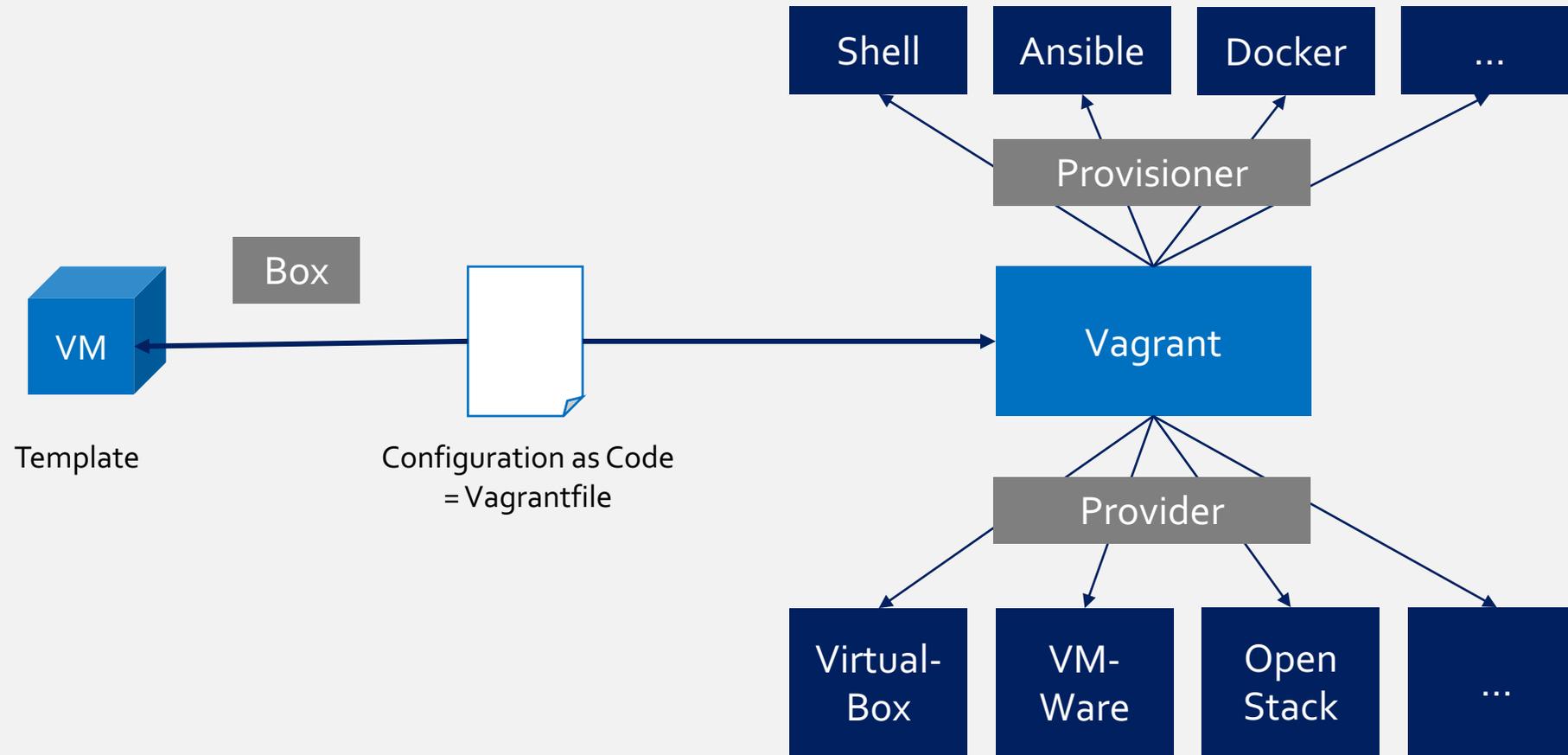
**Download:**

<https://www.vagrantup.com/downloads>

*Diese  
Kombination  
eignet sich gut  
für lokale  
Entwicklungs-  
maschinen.*

# VAGRANT

## Konzepte



# VAGRANTFILE

## Beschreibung einer zu erstellenden Virtuellen Maschine

```
# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # My base box
  config.vm.box = "chef/ubuntu-14.04"

  # Define shell provisioning
  config.vm.provision :shell, path: "bootstrap.sh"

  # Define docker provisioning
  config.vm.provision "docker" do |d|
    d.run "nginx1", image: "dockerfile/nginx", args: "-p 8080:80", daemonize: true
    d.run "nginx2", image: "dockerfile/nginx", args: "-p 9080:80", daemonize: true
    d.run "haproxy", image: "dockerfile/haproxy", args: "-p 80:80 --link nginx1:nginx1 --link nginx2:nginx2 -v /vagrant:/haproxy-override"
  end

  # Configure VirtualBox
  config.vm.provider "virtualbox" do |v|
    v.memory = 1024
    v.cpus = 4
  end

  # Forward ports
  config.vm.network :forwarded_port, host: 80, guest: 80
  config.vm.network :forwarded_port, host: 8080, guest: 8080
  config.vm.network :forwarded_port, host: 9080, guest: 9080
end
```

Definition der Basis-Box

Konfiguration der Provisionierung

Konfiguration des Virtualisierungs-Providers

Konfiguration des Netzwerks

# VAGRANT

## Typischer Arbeitsablauf

#	Befehle	Bedeutung
1	<code>md &lt;box-dir&gt;</code> <code>cd &lt;box-dir&gt;</code>	Verzeichnis für Vagrant Umgebung erstellen und dorthin wechseln
2	<code>vagrant init [&lt;box-name&gt;] [&lt;boxurl&gt;]</code>	Vagrant Umgebung initialisierung. Es wird eine Datei <i>Vagrantfile</i> erstellt und initial mit dem Namen und URL der Box falls angegeben initialisiert.
3	<code>code Vagrantfile</code>	<i>Vagrantfile</i> nach Bedarf anpassen (z.B. IP vergeben, Port-Mapping/Verzeichnis-Share-Mapping zwischen Host und Guest, ...)
4	<code>vagrant up</code>	Startet die Virtuelle Maschine aus dem Box-Template und konfiguriert diese entsprechend dem <i>Vagrantfile</i>
5	<code>vagrant ssh</code>	Per SSH auf die VM einloggen (vom Host)
6	<code>exit</code> (in SSH Shell des Guest)	Die SSH Shell in der VM verlassen (zurück zum Host)
7	<code>vagrant halt</code>	Die VM stoppen.

Ggf. hilfreich:

**reload:** Startet eine VM neu und aktualisiert die Konfiguration nach angepasstem *Vagrantfile*

**package:** Erstellt aus einer VM wieder eine Box (Template)

**destroy:** Löscht eine VM

# VAGRANT

*Nur Versuch macht kluch ...*



## Klonen Sie dieses Repository:

```
git clone https://git.mylab.th-luebeck.de/cloud-native/lab-local-vm-provisioning.git
```



### NGINX

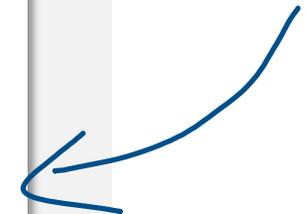
```
cp Vagrantfile.nginx Vagrantfile  
vagrant up
```



### Kubernetes (MicroK8S)

```
cp Vagrantfile.microk8s Vagrantfile  
vagrant up
```

*Diese  
Provisionierung  
kann Ihnen in  
diesem Modul noch  
viel Arbeit  
ersparen.*



# INFRASTRUCTURE AS CODE

## Terraform

- Entwickelt von HashiCorp (wie auch Vagrant)
- Open Source, in Go geschrieben
- Kommandozeilenwerkzeug
- Deklarativ und Push-basierter Ansatz
- Direkte Anbindung vieler Public und Private Cloud Infrastrukturen und Plattformen (u.a. AWS, GCE, Azure, vSphere, OpenStack, Kubernetes, uvm.)



```
Terraform will perform the following actions:

# kubernetes_pod.example will be updated in-place
~ resource "kubernetes_pod" "test" {
  id = "default/terraform-test"

  metadata {
    generation = 0
    labels     = {
      "app" = "MyApp"
    }
  }
  name           = "terraform-test"
  namespace      = "default"
  resource_version = "650"
  self_link      = "/api/v1/namespaces/default/pods/terraform-test"
  uid            = "5130ef35-7c09-11e9-be7c-080027f59de6"
}

~ spec {
```

*vagrant ist gut für Single Node/VM Installationen (Städte).*

*Terraform „formiert ganze Planeten“ (also Multi-Node Installationen).*

# TERRAFORM

*Deklarative Programmierung von Infrastrukturen*

## Write:

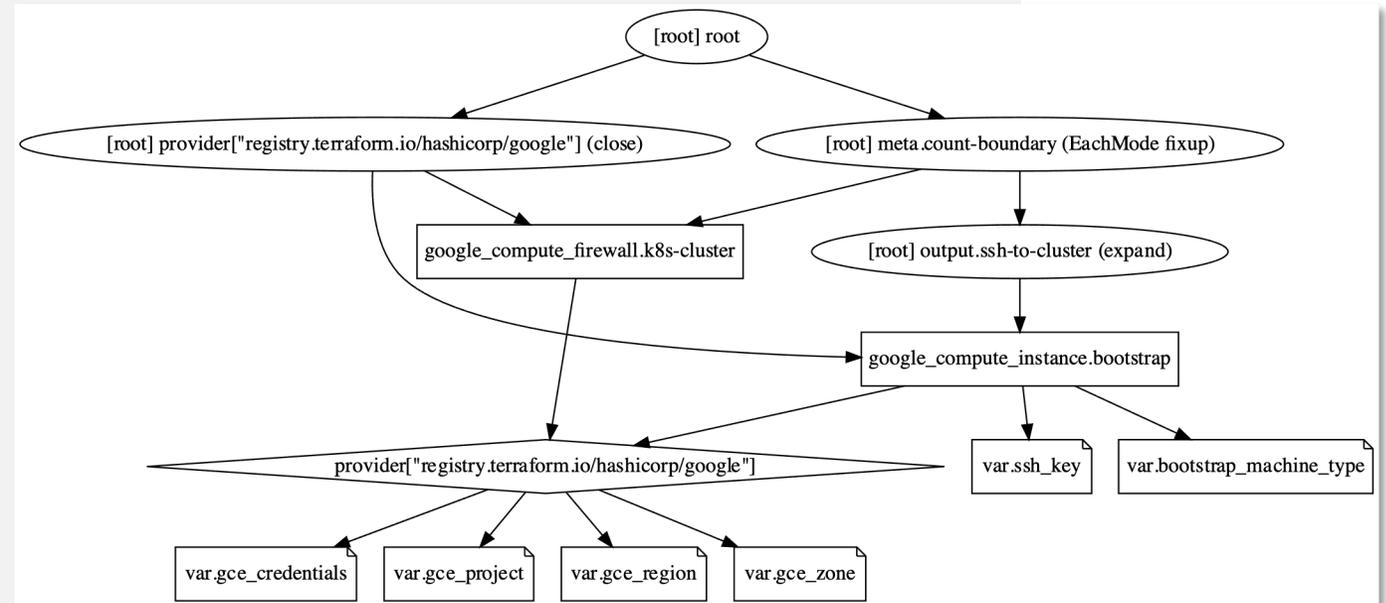
Beschreibung eines Zielzustands über eine domänenspezifische Sprache HCL (HashiCorp Configuration Language)

## Plan (terraform plan):

Ist-Zustand ermitteln. Notwendige Änderungen planen (entsprechend Abhängigkeiten geordnet und parallelisiert)

## Apply (terraform apply):

Idempotente Herstellung des Zielzustands. Der Zustand (.tfstate) Datei wird dabei lokal oder einem Remote Store (S3, HTTP, Postgres DB, ...) gespeichert.



Beispiel eines Terraform Plans (Directed Acyclic Graph, DAG) zur Ausbringung eines Kubernetes Nodes in GCE

# TERRAFORM

## Die Kern-Entitäten in Terraform Skripten

### Provider:

Schnittstelle zum Infrastruktur-Provider.  
Hier werden üblicherweise die Access Credentials, Zugriffsprotokolle, sowie die Data Centers (Zonen) hinterlegt.

```
provider "google" {  
  credentials = file(var.gce_credentials)  
  project = var.gce_project  
  region = var.gce_region  
  zone = var.gce_zone  
}
```

### Data Sources:

Zugriff/bzw. Bekanntmachung von Informationen, die nicht durch Terraform selber in der Infrastruktur angelegt werden, sondern extern und global definiert werden/wurden (und nicht anpassbar sind).

- Netzwerknamen
- Datacenter Namen

```
data vsphere_network "private" {  
  name          = "ei-vm-clients"  
  datacenter_id = data.vsphere_datacenter.this.id  
}  
  
data vsphere_network "public" {  
  name          = "dmz_ei_vm2"  
  datacenter_id = data.vsphere_datacenter.this.id  
}
```

*Diese Entität wird meist in privaten Infrastrukturen benötigt, weniger in Public Cloud Infrastrukturen.*

# TERRAFORM

## Die Kern-Entitäten in Terraform Skripten

### Ressource:

Anzulegende Ressourcen in der Infrastruktur

```
resource "google_compute_instance" "bootstrap" {  
  name = "k8s-bootstrap"  
  machine_type = var.bootstrap_machine_type  
  
  metadata = {  
    ssh-keys = "ubuntu:${file(var.ssh_key)}"  
  }  
  
  metadata_startup_script = file("resources/install.sh")  
  
  boot_disk {  
    initialize_params {  
      image = "ubuntu-os-cloud/ubuntu-2004-lts"  
    }  
  }  
}
```

### Provisioner:

Ausführung von Änderungen auf Ressourcen

```
connection {  
  type      = "ssh"  
  user      = "ubuntu"  
  private_key = file(var.priv_ssh_key)  
  host      = var.bastion.ip  
}  
  
provisioner "remote-exec" {  
  inline = [  
    "sudo apt-get install fail2ban -y",  
    "sudo ufw allow http",  
    "sudo ufw allow https",  
    "sudo ufw allow 16443"  
  ]  
}
```

# TERRAFORM

## Strukturierung von Terraform Skripten

*Basisdefinitionen, die für alle Environments gelten*

```
terraform/  
├── base/  
│   ├── vpc.tf  
│   ├── network.tf  
│   ├── variables.tf  
│   └── terraform.tfvars  
├── qa/  
│   ├── ec2.tf  
│   ├── cloudwatch.tf  
│   ├── route53.tf  
│   ├── variables.tf  
│   └── terraform.tfvars  
└── prod/  
    ├── ec2.tf  
    ├── cloudwatch.tf  
    ├── route53.tf  
    ├── variables.tf  
    └── terraform.tfvars
```

*\*.tf: Terraform Deklarationen*

*QA (Test/Staging) Environment*

*Production Environment (nur die sieht der Kunde, Live-System)*

*\*.tfvars: Variablenwerte (ggf. Environment-spezifisch)*

# TERRAFORM

*Nur Versuch macht kluch ...*



## Klonen Sie dieses Repository:

```
git clone https://git.mylab.th-luebeck.de/cloud-native/lab-iaas-iac.git
```



## Kubernetes (MicroK8S)

```
> cd cluster
> terraform init
> terraform apply
```

Ergänzen Sie anschließend dies zur  
`config.auto.tfvars`

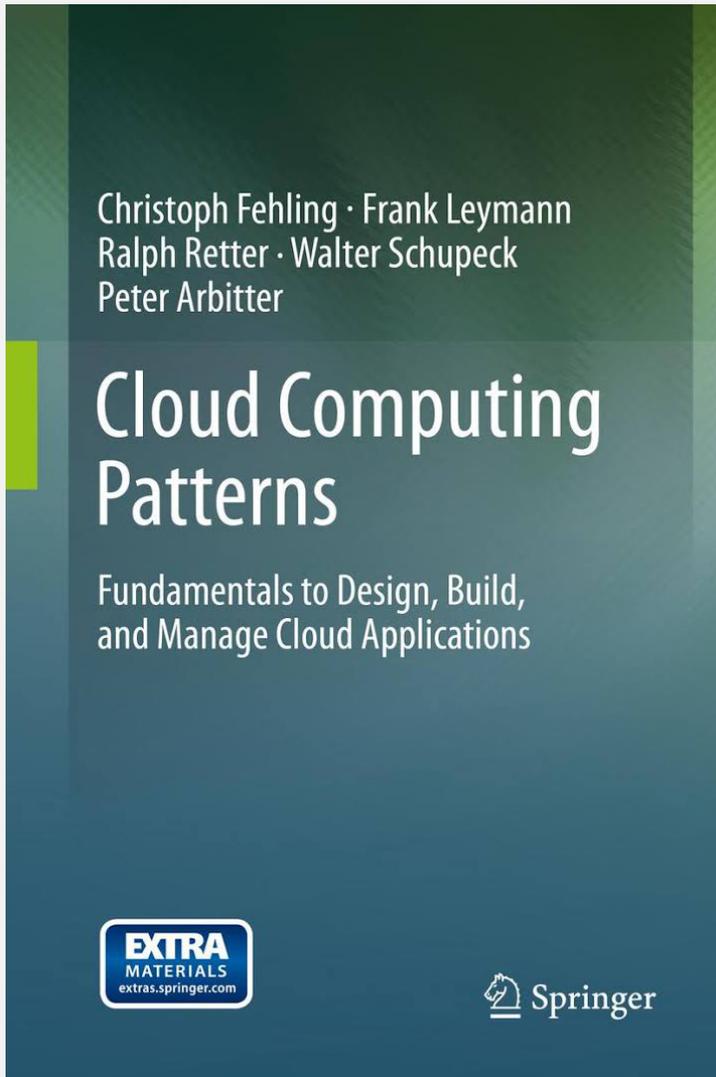
```
small_workers=2
xl_workers=2
```

um Ihren Cluster von einem Knoten auf fünf  
Knoten hochzuskalieren.

```
> terraform apply
```

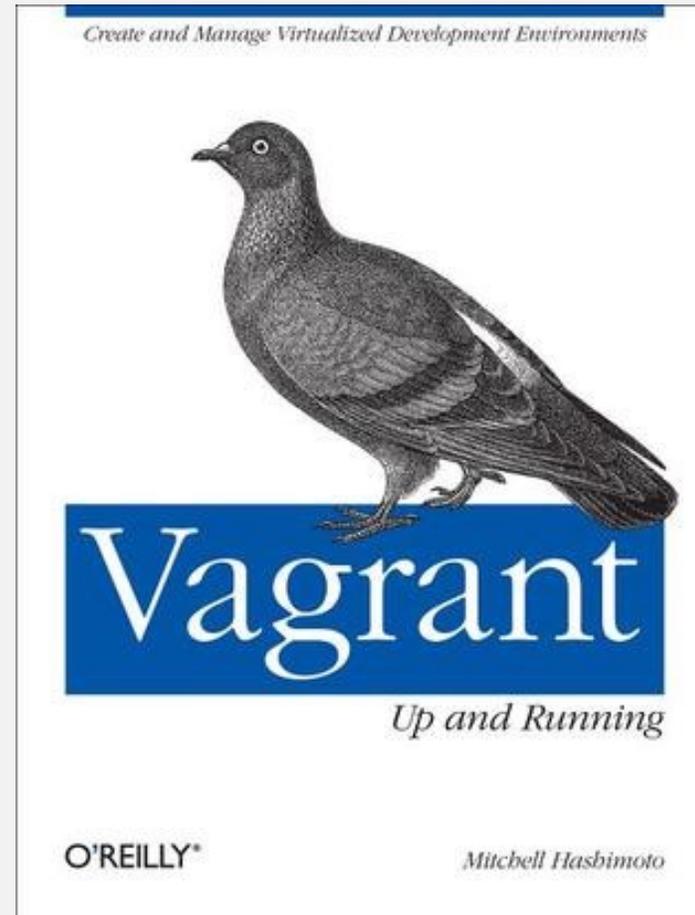
*Provisioning  
und  
Skalierung von  
Kubernetes  
Cluster in GCE.*

# ZUM NACHLESEN



- 2.3**
  - 2.3.1 Infrastructure as a Service
  
- 3.3.**
  - 3.3.1 Elastic Infrastructure
  
- 3.4**
  - 3.4.1 Hypervisor
  
- 3.5**
  - 3.5.1 Block Storage
  - 3.5.2 Blob Storage
  
- 3.6**
  - 3.6.1 Virtual Networking

# ZUM NACHLESEN



# KONTAKT

*Disclaimer*

**Nane Kratzke**

📞 +49 451 300-5549

✉ nane.kratzke@th-luebeck.de

🌐 kratzke.mylab.th-luebeck.de

