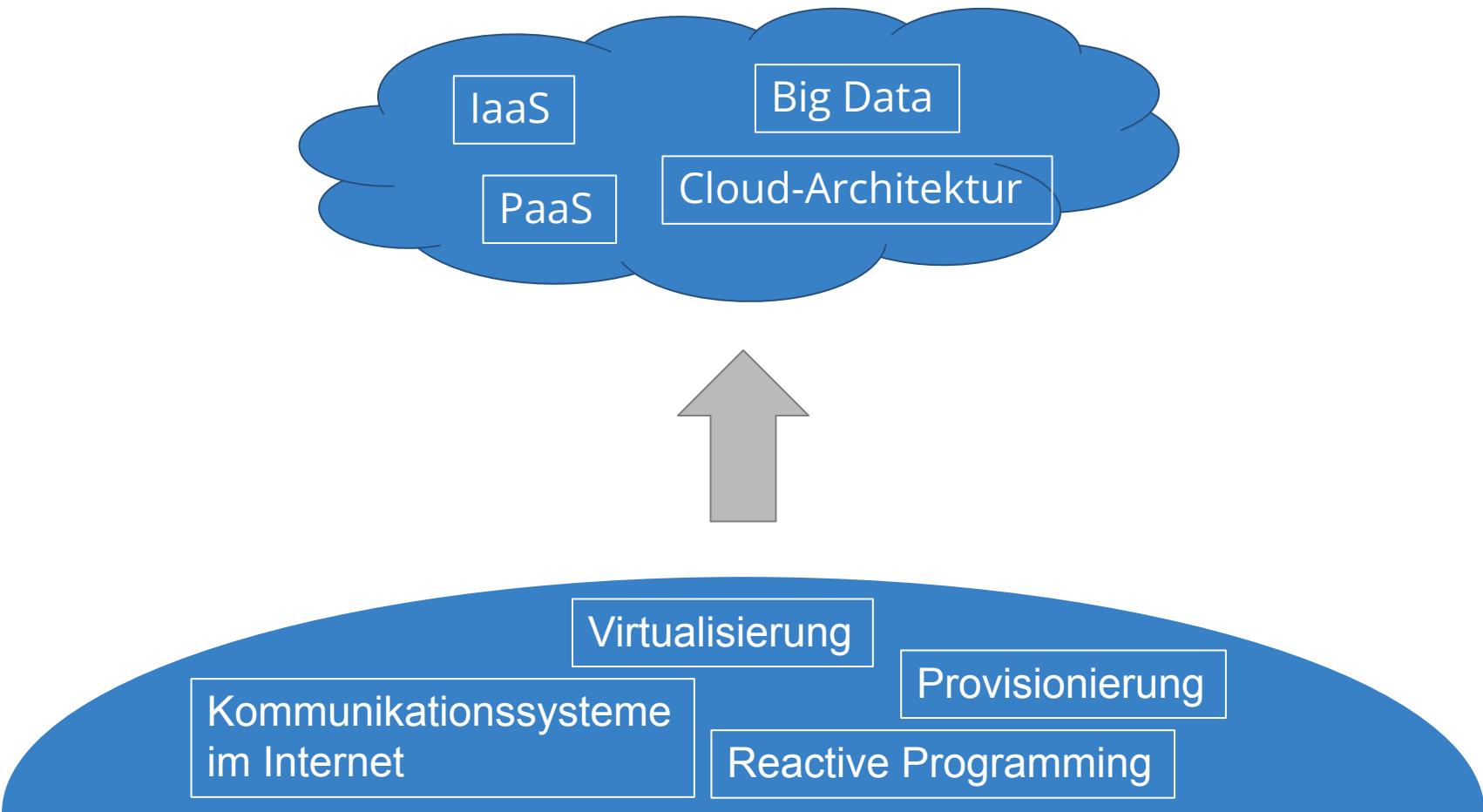


# Kapitel 7: Infrastructure-as-a-Service

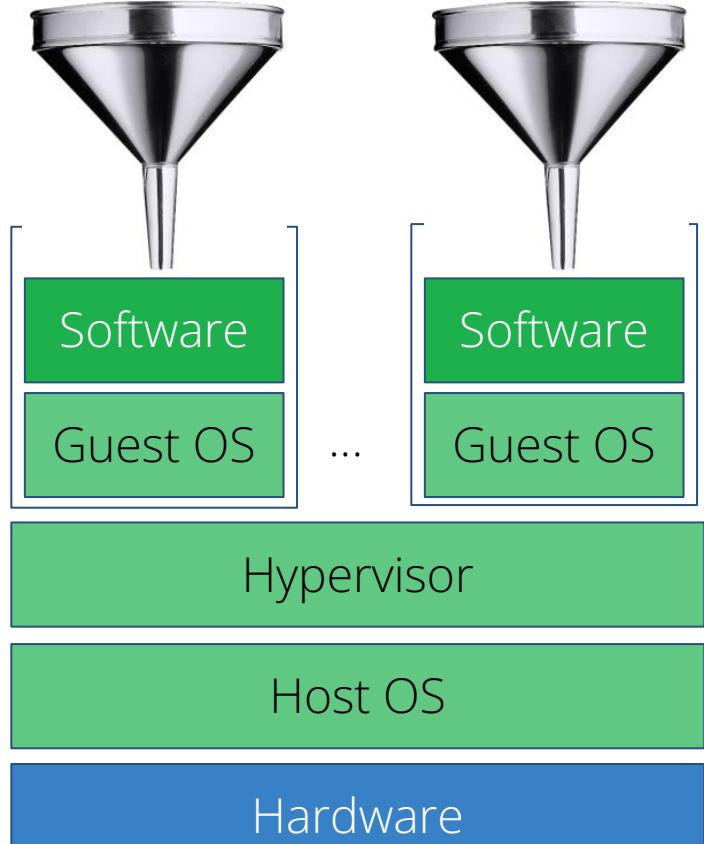
*vorlesung*

**CLOUD  
COMPUTING**

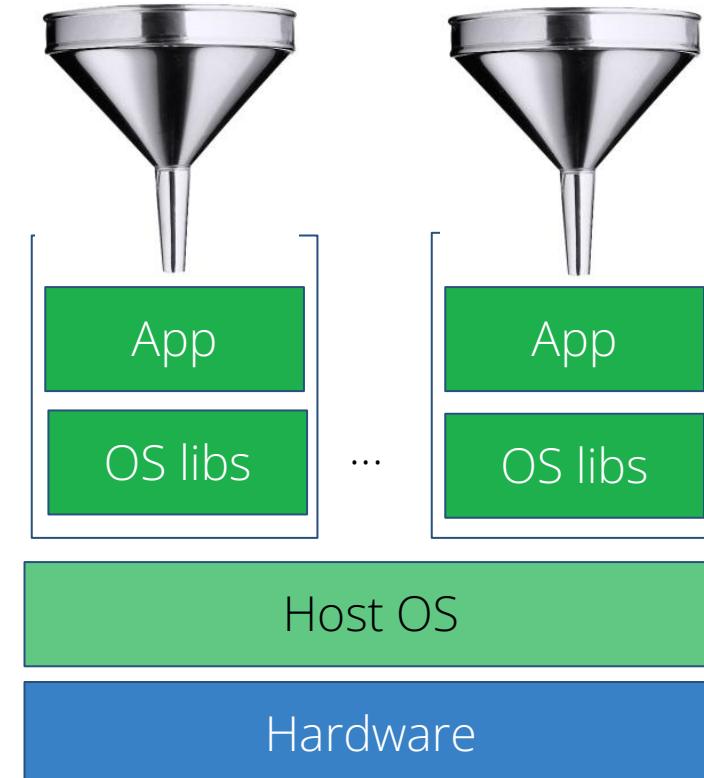
# Ab heute sind wir in der Cloud



# Die letzte Vorlesung: Wie kommt Software auf das Blech?

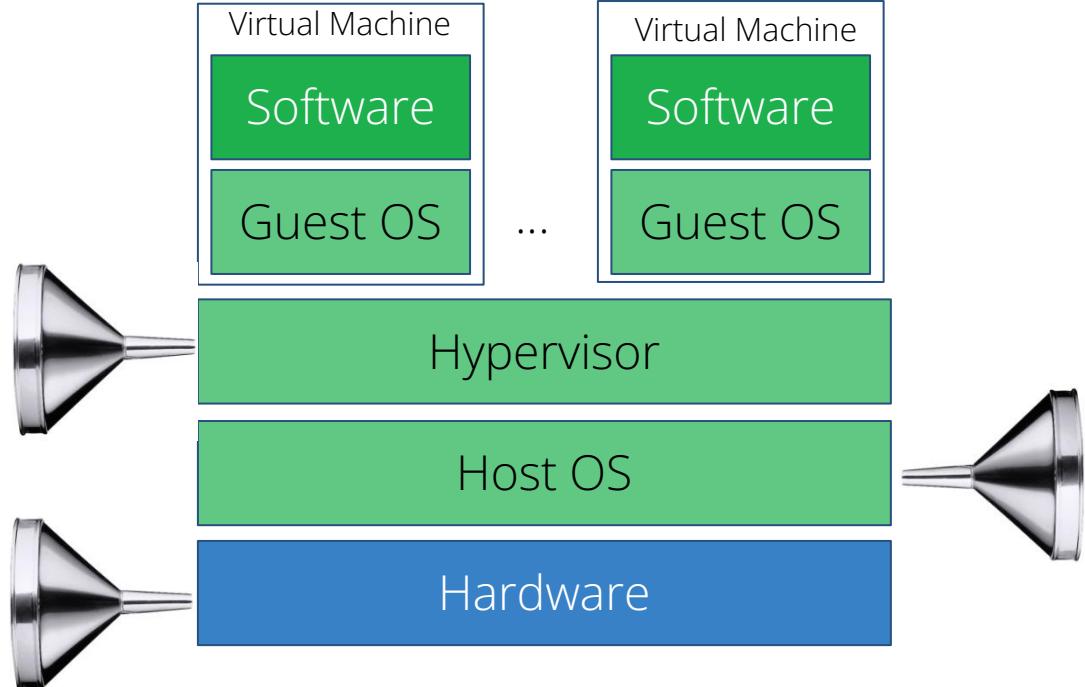


Hardware-Virtualisierung

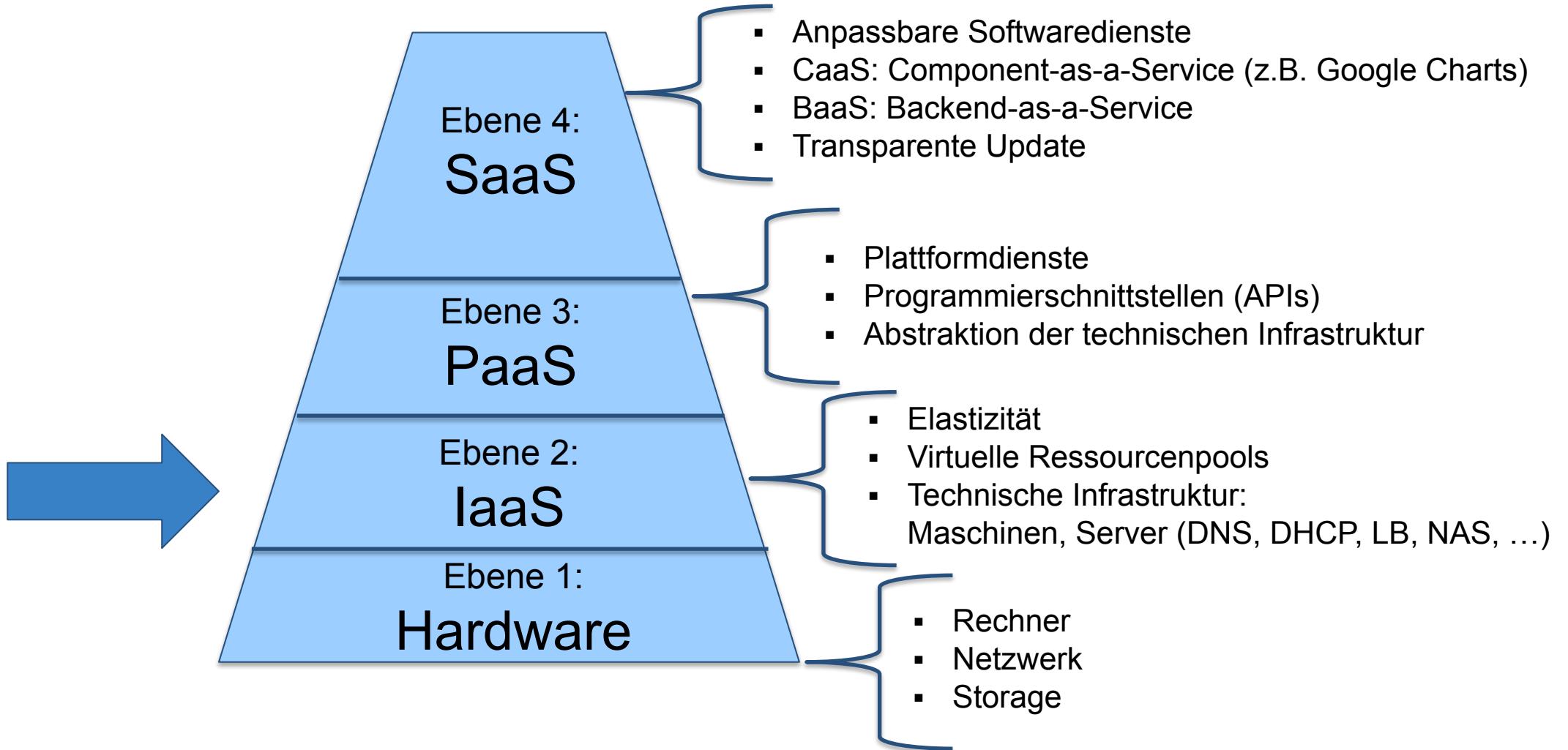


Betriebssystem-Virtualisierung

# Heute: Wie kommt Software an das Blech?



# Das Schichtenmodell des Cloud Computing: Vom Blech zur Anwendung



# Einführung: Infrastructure-as-a-Service

# Time-to-System im letzten Jahrhundert: > 1 Jahr



- ✓ Rechner kaufen
- ✓ Raum ausbauen
- ✓ Rechner installieren
- ✓ (Programme schreiben)
- ✓ Programme installieren

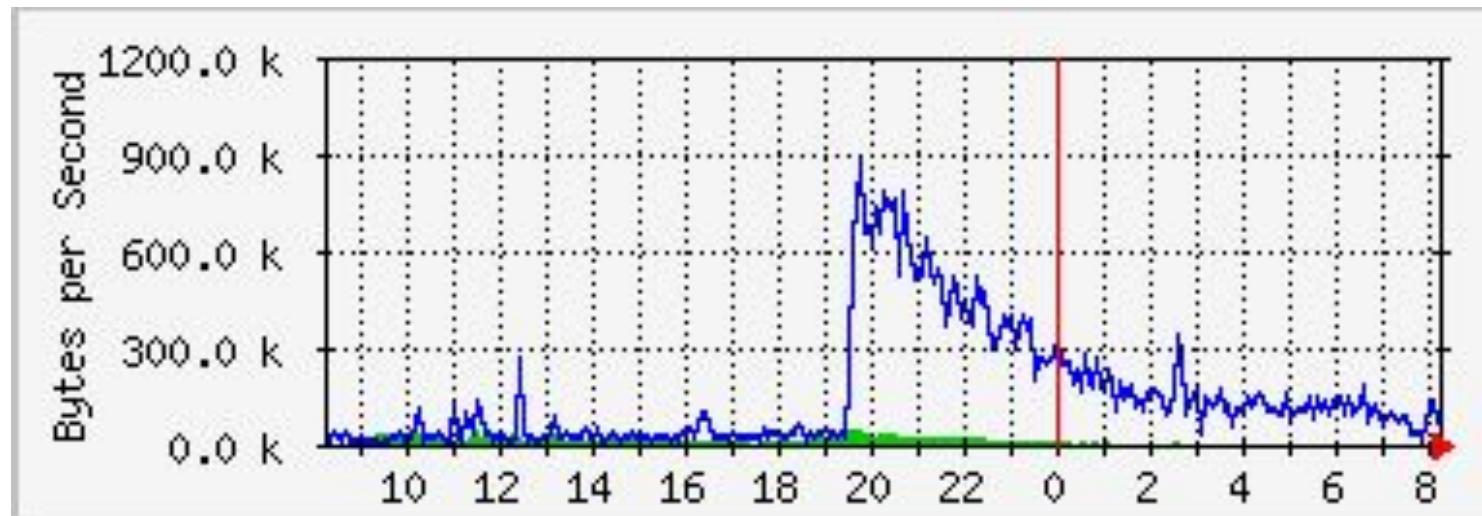
<https://de.wikipedia.org/wiki/Großrechner>

# Time-to-System in der Cloud-Ära: In Echtzeit

Als **Slashdot-Effekt** oder Slashdotting wird im englischsprachigen Netzjargon eine durch die Leser einer besonders populären Website oder eines populären Social-Media-Accounts verursachte Überlastung eines Web-Servers bezeichnet. Weitere Bezeichnungen im deutschsprachigen Raum sind **Heise-Effekt** oder **Twitter-Effekt**.

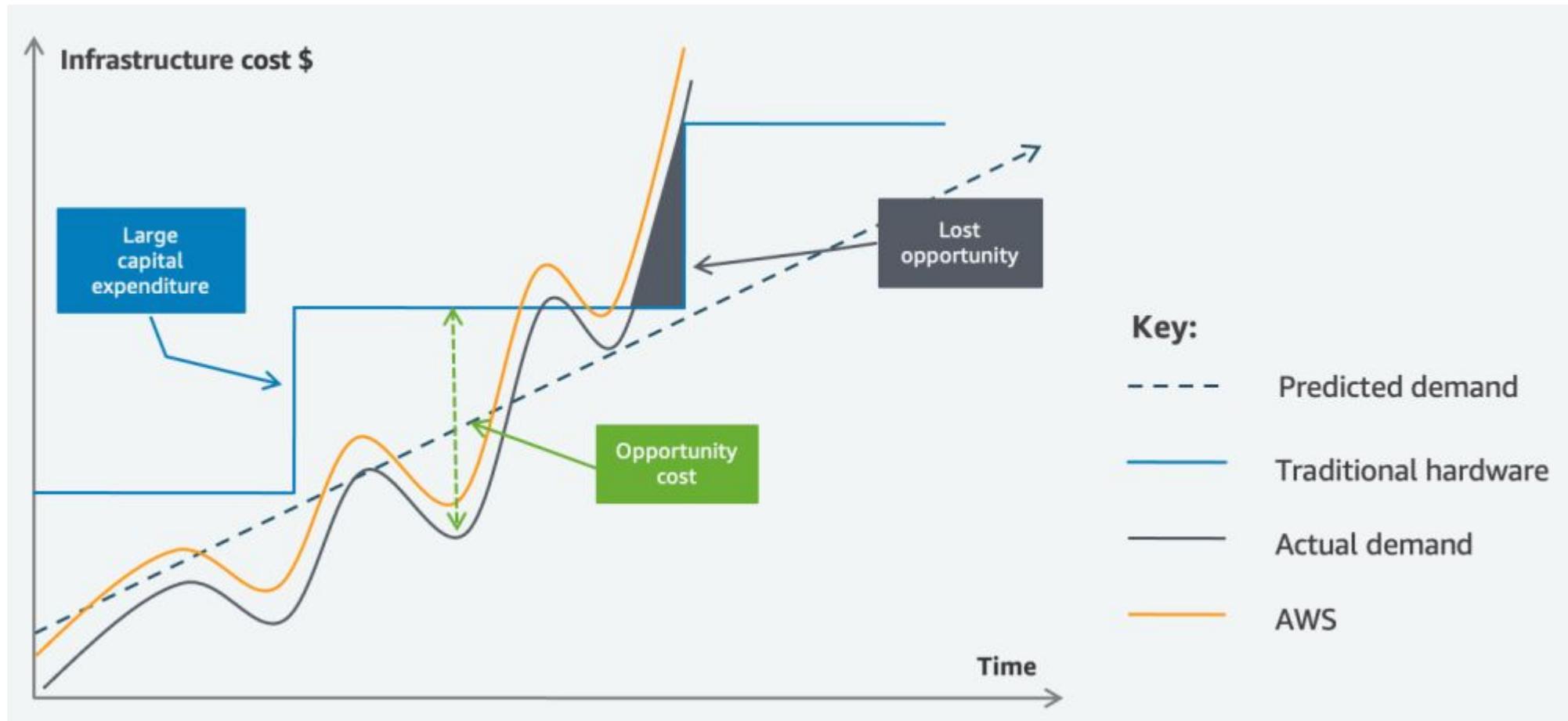
Der Slashdot-Effekt oder das Slashdotting tritt auf, wenn eine bisher wenig populäre Webseite von einem IT-Online-Magazin wie Slashdot oder heise aufgegriffen wird und so binnen Minuten ein erheblicher Benutzeransturm auf die Webseite beginnt. Dieser führt oft dazu, dass erheblicher Datenverkehr verursacht wird und der Server vorübergehend einzelne Anfragen nicht mehr oder nur noch sehr langsam beantworten kann. Die Seite ist dann „geslashdottet“ (englisch slashdotted).

Im deutschsprachigen Raum spricht man analog dazu auch vom Heise-Effekt („Server wurde geheist“) bei der Erwähnung einer Webseite bei heise online. Auch andere Nicht-IT-Online-Magazine (etwa tagesschau.de, Spiegel Online oder CNN) können ähnlichen Datenverkehr verursachen, wenn sie eine Webseite erwähnen, deren Server dem Ansturm nicht gewachsen ist.



<https://de.wikipedia.org/wiki/Slashdot-Effekt>

# Klassische Betriebsszenarien werden bei dynamischer Nachfrage teuer - Hohe Opportunitätskosten



Source: [Amazon Web Services](#)

# Definition IaaS

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

Eigenschaften einer IaaS-Cloud:

- **Ressourcen-Pools:** Verfügbarkeit von scheinbar unbegrenzten Ressourcen, die Anfragen verteilt verarbeiten.
- **Elastizität:** Dynamische Zuweisung von zusätzlichen Ressourcen bei Bedarf.
- **Pay-as-you-go Modell:** Abgerechnet werden nur verbrauchte Ressourcen.

Ressourcen-Typen in einer IaaS-Cloud:

- **Rechenleistung:** Rechner-Knoten mit CPU, RAM und HD-Speicher.
- **Speicher:** Storage-Kapazitäten als Dateisystem-Mounts oder Datenbanken.
- **Netzwerk:** Netzwerk und Netzwerk-Dienste wie DNS, DHCP, VPN, CDN und Load Balancer.

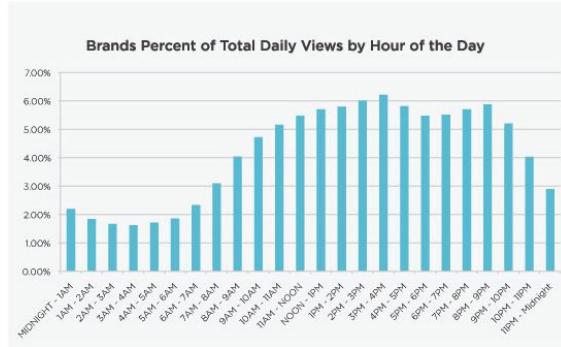
Infrastruktur-Dienste einer IaaS-Cloud:

- **Monitoring**
- **Ressourcen-Management**

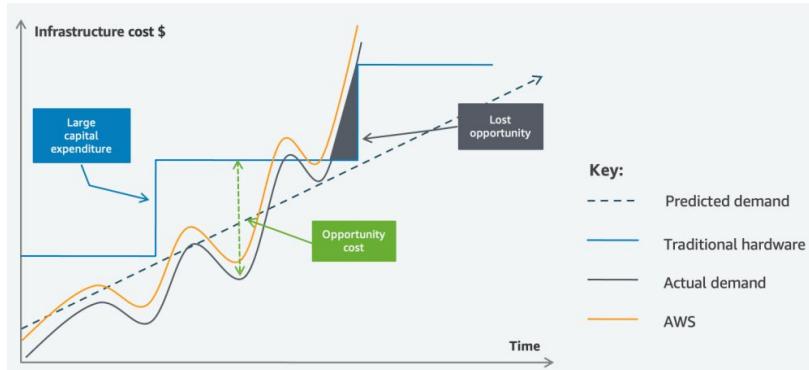
# Skalierbarkeit: Effekte

## ■ Tageszeitliche und saisonale Effekte:

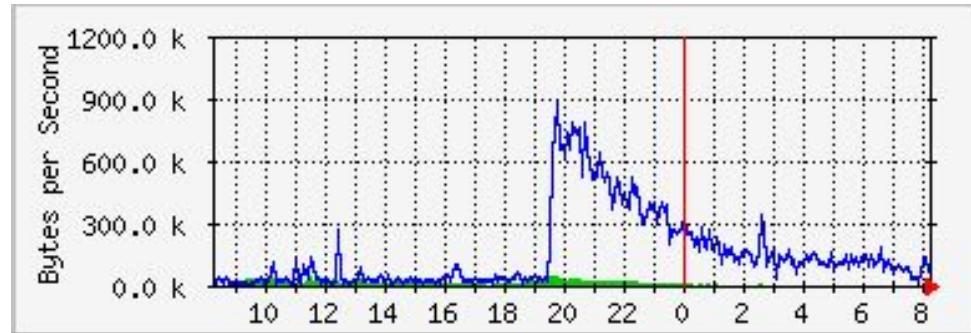
Mittags-Peak, Prime-Time-Peak, Wochenend-Peak,  
Weihnachten, Valentinstag, Muttertag, Black Friday  
(vorhersehbare Belastungsspitzen)



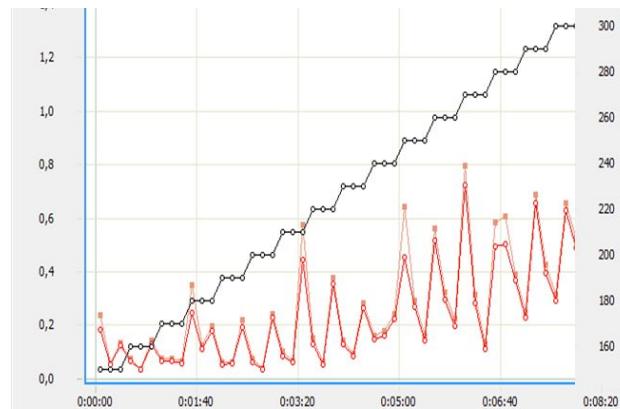
## ■ Kontinuierliches Wachstum



## ■ Sondereffekte: z.B. Slashdot-Effekt (unvorhersehbare Belastungsspitzen)



## ■ Temporäre Plattformen: Projekte, Tests, Batch...



# Elastizitätsarten

**Nachfrageelastizität:** Die allokierten Ressourcen steigen / sinken mit der Nachfrage.

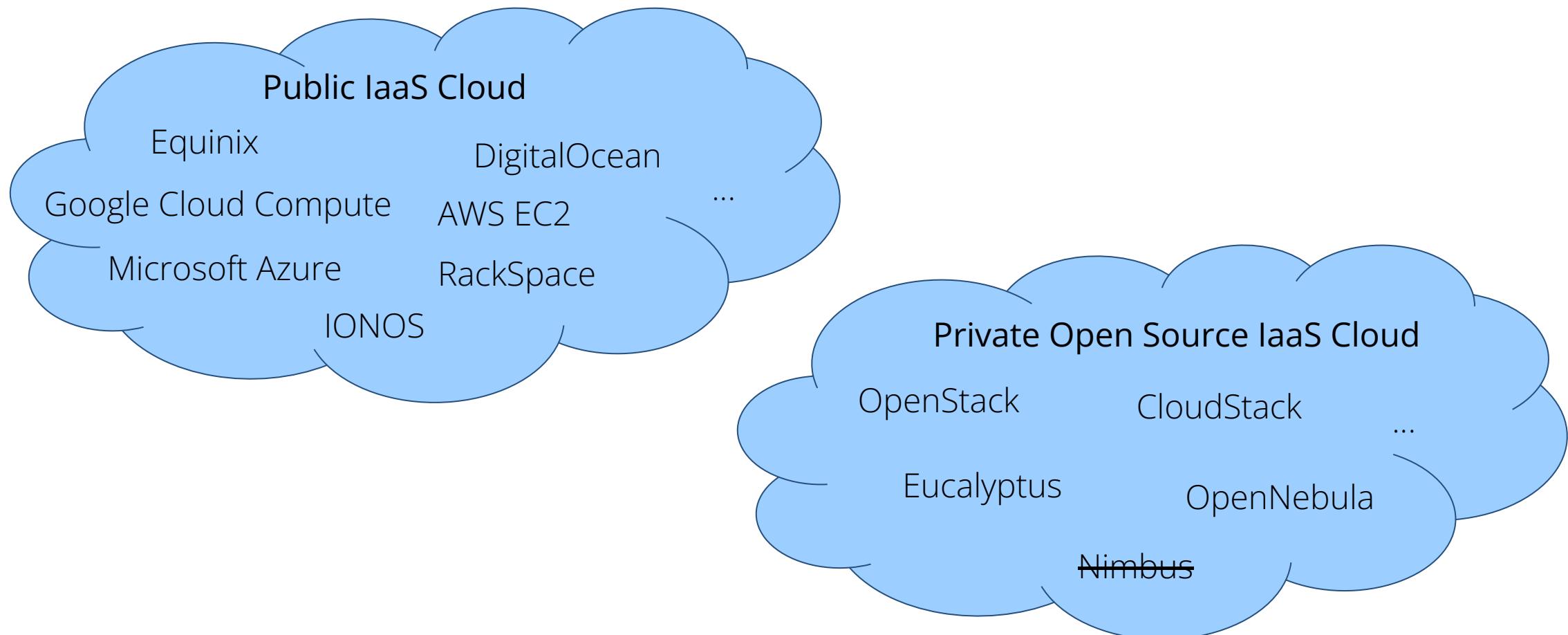
- Pseudo-Elastizität: Schneller Aufbau. Kurze Kündigungsfrist.
- Echtzeit-Elastizität: Allokation und Freigabe von Ressourcen innerhalb von Sekunden.  
Automatisierter Prozess mit manuellen Triggern oder nach Zeitplan.
- Selbstdadaptive Elastizität: Automatische Allokation und Freigabe von Ressourcen in Echtzeit auf Basis von Regeln und Metriken.

**Angebotselastizität:** Die allokierten Ressourcen steigen / sinken mit dem Angebot.

- Dies ist das typische Verhalten eines Grids: Alle verfügbaren Rechner werden allokiert.
- Es sind auch Varianten verfügbar, bei denen man für freie Ressourcen bieten kann.

**Einkommenselastizität:** Die allokierten Ressourcen steigen / sinken mit dem Einkommen bzw. dem Budget.

# Es gibt vielerlei Anbieter für Public und Private IaaS Clouds



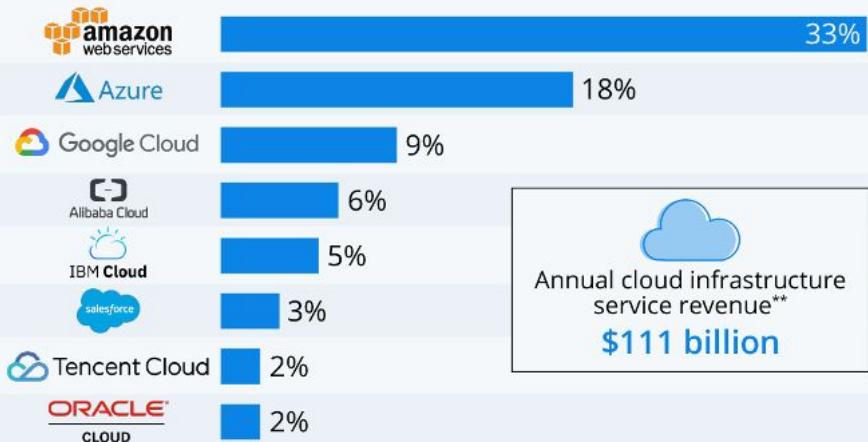
# Der IaaS Markt 2020

Figure 1. Magic Quadrant for Cloud Infrastructure and Platform Services



## Amazon Leads \$100 Billion Cloud Market

Worldwide market share of leading cloud infrastructure service providers in Q2 2020\*



\* includes platform as a service (PaaS) and infrastructure as a service (IaaS) as well as hosted private cloud services

\*\* 12 months ended June 30, 2020

Source: Synergy Research Group



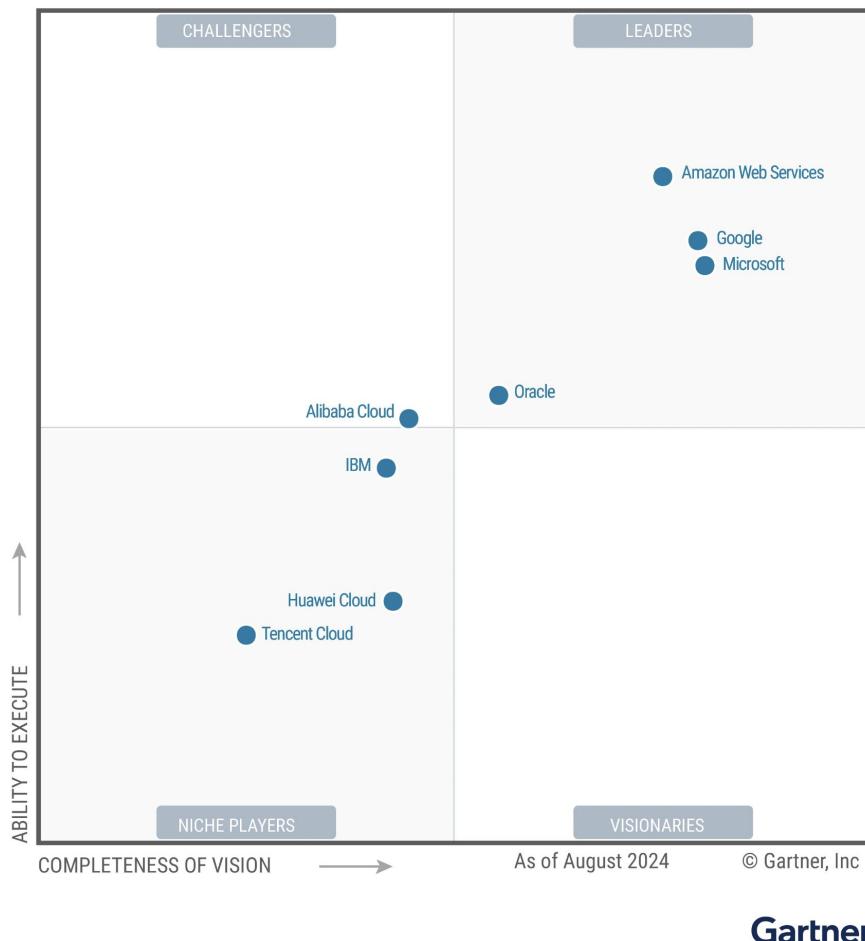
statista

2020 Magic Quadrant for Cloud Infrastructure & Platform Services

<https://azure.microsoft.com/en-us/blog/microsoft-named-a-leader-in-2024-gartner-magic-quadrant-for-strategic-cloud-platform-services/>

# Der IaaS Markt 2024

Figure 1: Magic Quadrant for Strategic Cloud Platform Services

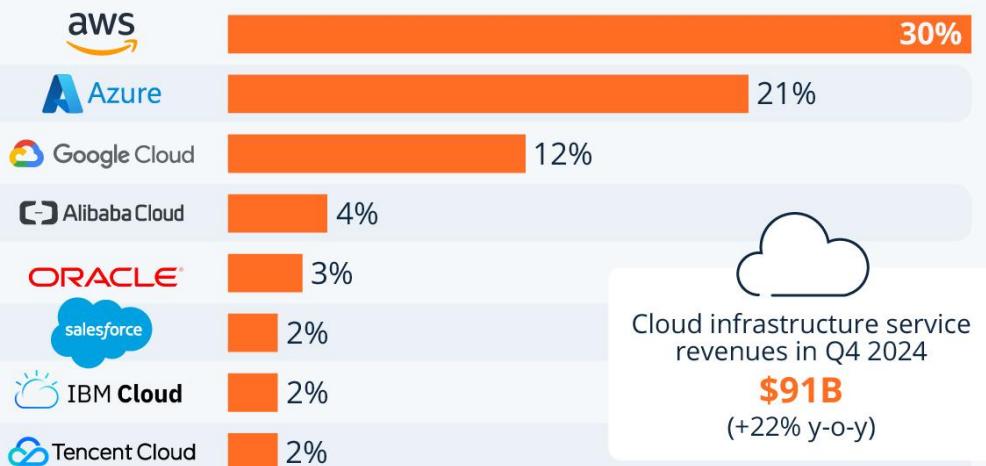


2024 Magic Quadrant for Cloud Infrastructure & Platform Services

<https://azure.microsoft.com/en-us/blog/microsoft-named-a-leader-in-2024-gartner-magic-quadrant-for-strategic-cloud-platform-services/>

## Amazon and Microsoft Stay Ahead in Global Cloud Market

Worldwide market share of leading cloud infrastructure service providers in Q4 2024\*



\* Includes platform as a service (PaaS) and infrastructure as a service (IaaS) as well as hosted private cloud services

Source: Synergy Research Group



statista

# Kriterien bei der Auswahl einer passenden IaaS-Cloud

- Unterstützte Cloud-Varianten (Private Cloud, Public Cloud, Hybrid Cloud, ...)
- Zuverlässigkeit / Verfügbarkeit
- Sicherheit und Datenschutz
- Vorhersagbare und stabile Performance
- Preismodell: Fixe und flexible Kosten
- Skalierbarkeit: Grenzen, Automatismen und Reaktionszeiten
- Lock-In der Daten und Anwendungen: Offene APIs
- Haftung
- Support

# Service Level Agreement

## Service Level Objective

Zielwert einer messbaren Metrik, die über die Qualität oder Verfügbarkeit eines Dienstes eine Aussage trifft.

## Service Level Agreement

Vertrag über die Bereitstellung von Ressourcen und Dienste mit Zuverlässigkeitzzusagen (SLOs). Häufig verbunden mit Konsequenzen bei Nichterfüllung der SLO, wie z.B. finanziellen Strafen.

## Verfügbarkeitsklassen:

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

## Beispiel: Amazon S3 (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.

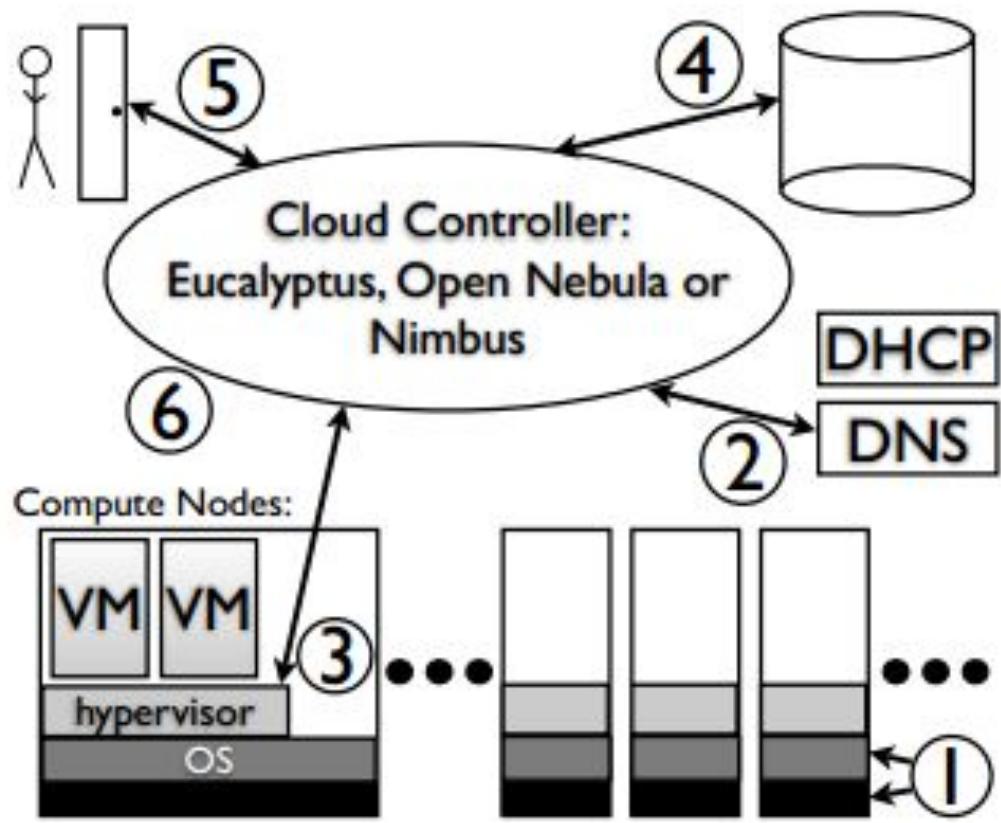
Monthly Uptime Percentage	Service Credit Percentage
Equal to or greater than 99% but less than 99.9%	10%
less than 99%	25%

# Aspekte der Sicherheit in einer IaaS-Cloud

- Vertraulichkeit der Daten und Datenkommunikation: Datenverschlüsselung, VPNs
- Nachvollziehbarkeit der Daten: Einhaltung nationaler Gesetze (z.B. EU-Datenschutzbestimmung, US Patriot Act) durch geographische Datenhaltung
- Firewalls und starke Authentifizierungsverfahren
- Backup der VMs, Storages und Datenbanken
- Zertifizierungen: ISO 27001, TÜV IT

# Architektur einer IaaS-Cloud

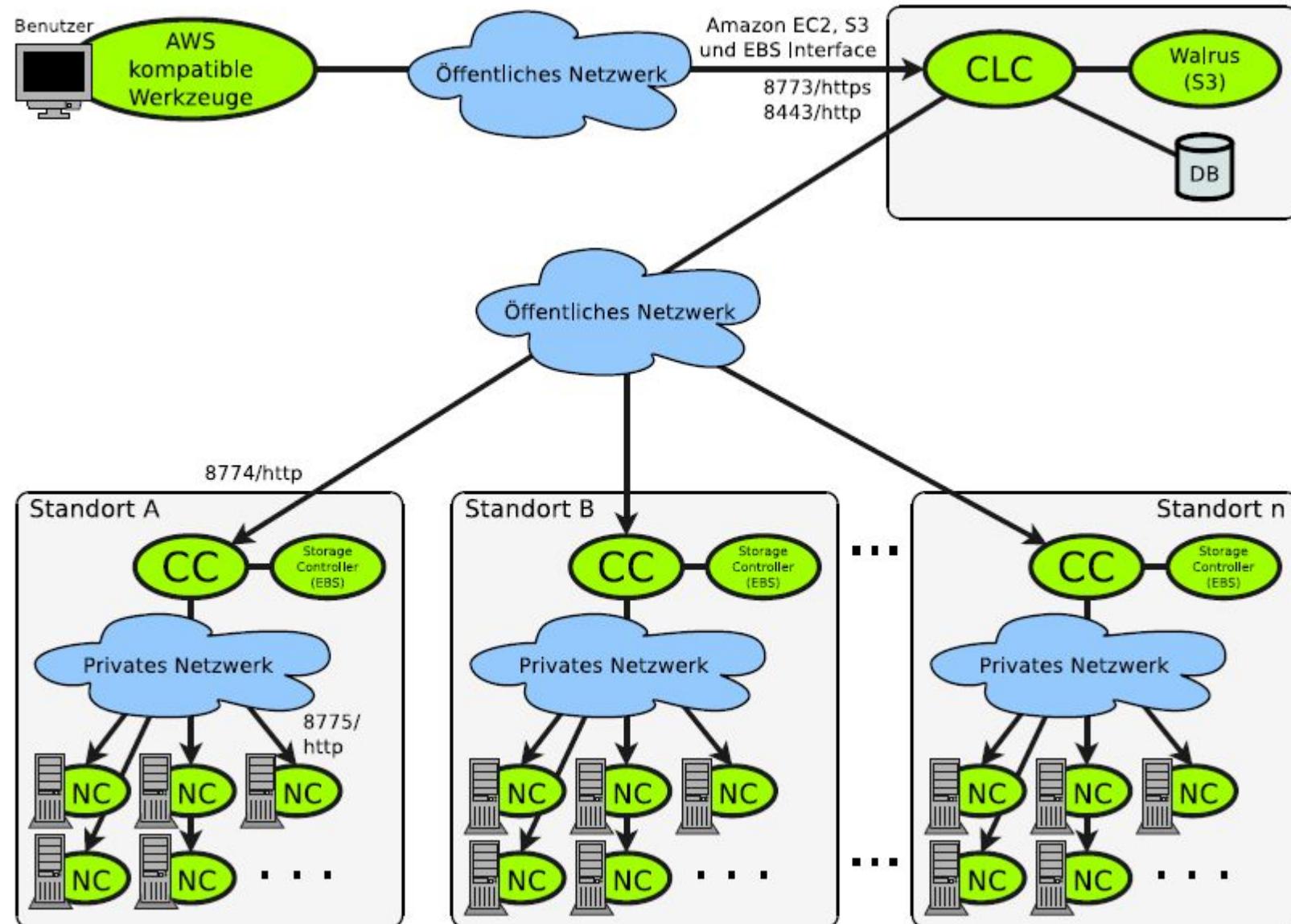
# Eine IaaS-Referenzarchitektur



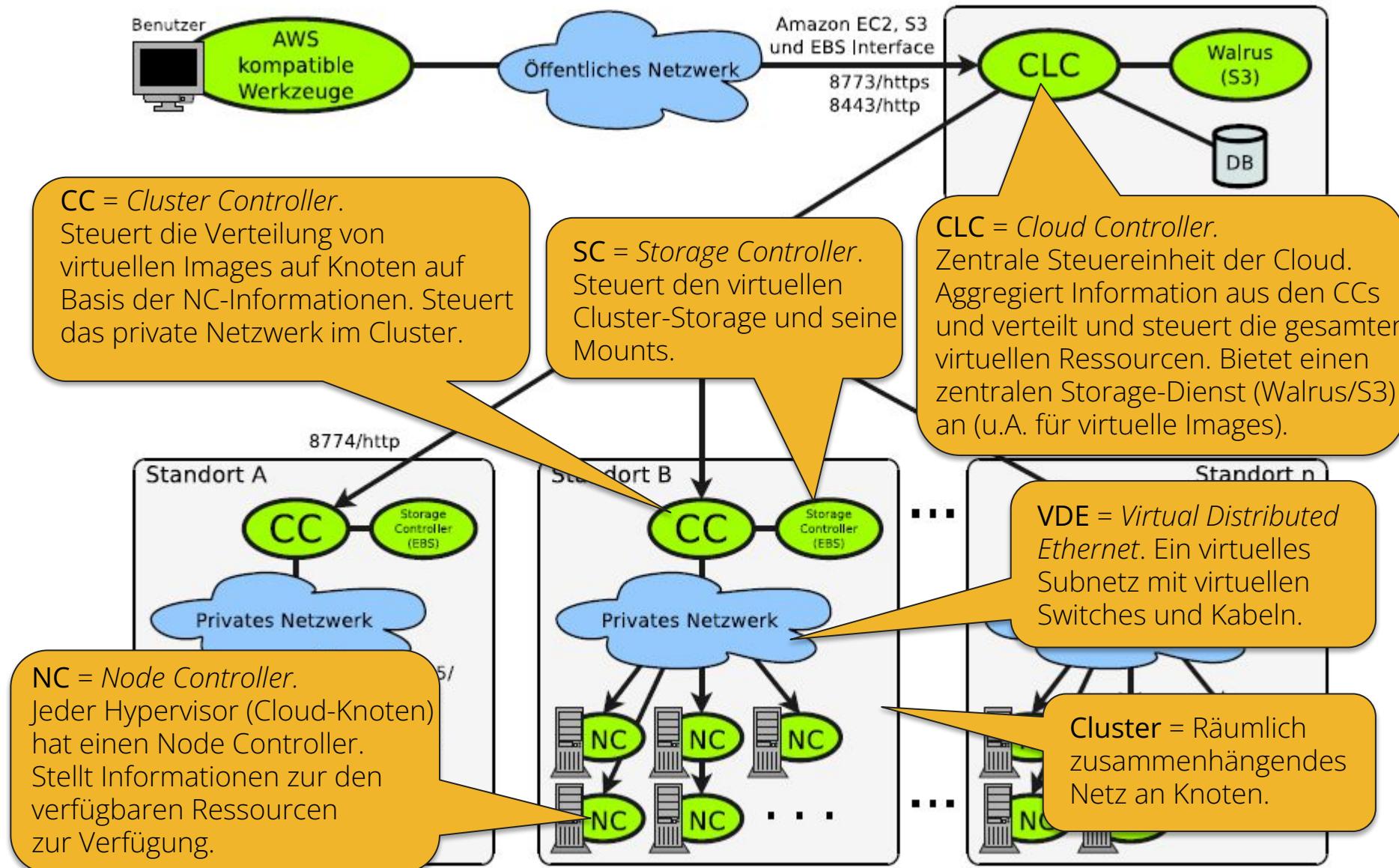
1. Hardware und Betriebssystem
2. Virtuelles Netzwerk und Netzwerkdienste
3. Virtualisierung
4. Datenspeicher und Image-Verwaltung
5. Managementschnittstelle für Administratoren und Benutzer
6. Cloud Controller für das mandantenspezifische Management der Cloud-Ressourcen

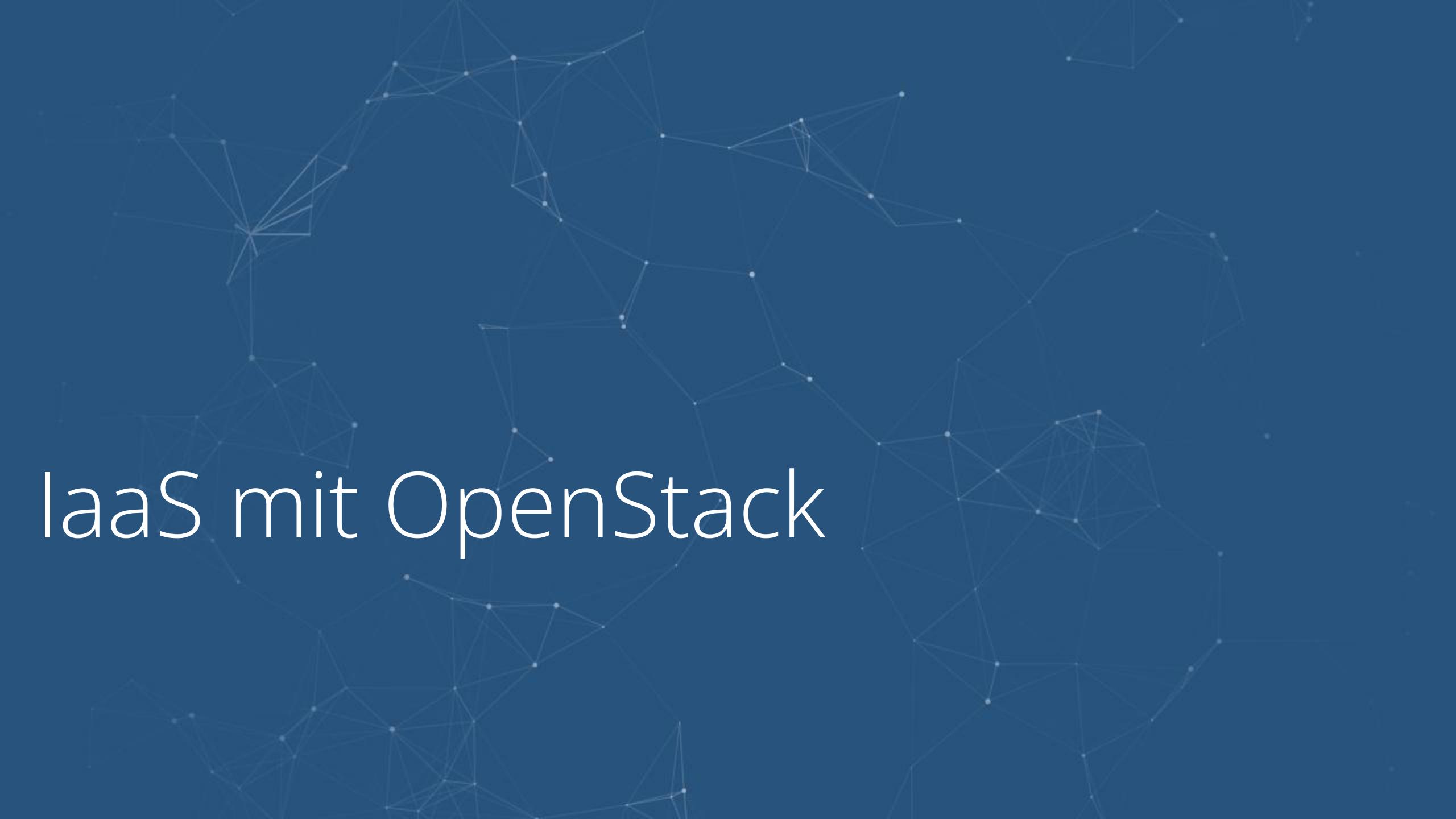
Peter Sempolinski and Douglas Thain,  
"A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus",  
IEEE International Conference on Cloud Computing Technology and Science, 2010.

# Der interne Aufbau einer IaaS-Cloud am Beispiel Eucalyptus



# Der interne Aufbau einer IaaS-Cloud am Beispiel Eucalyptus



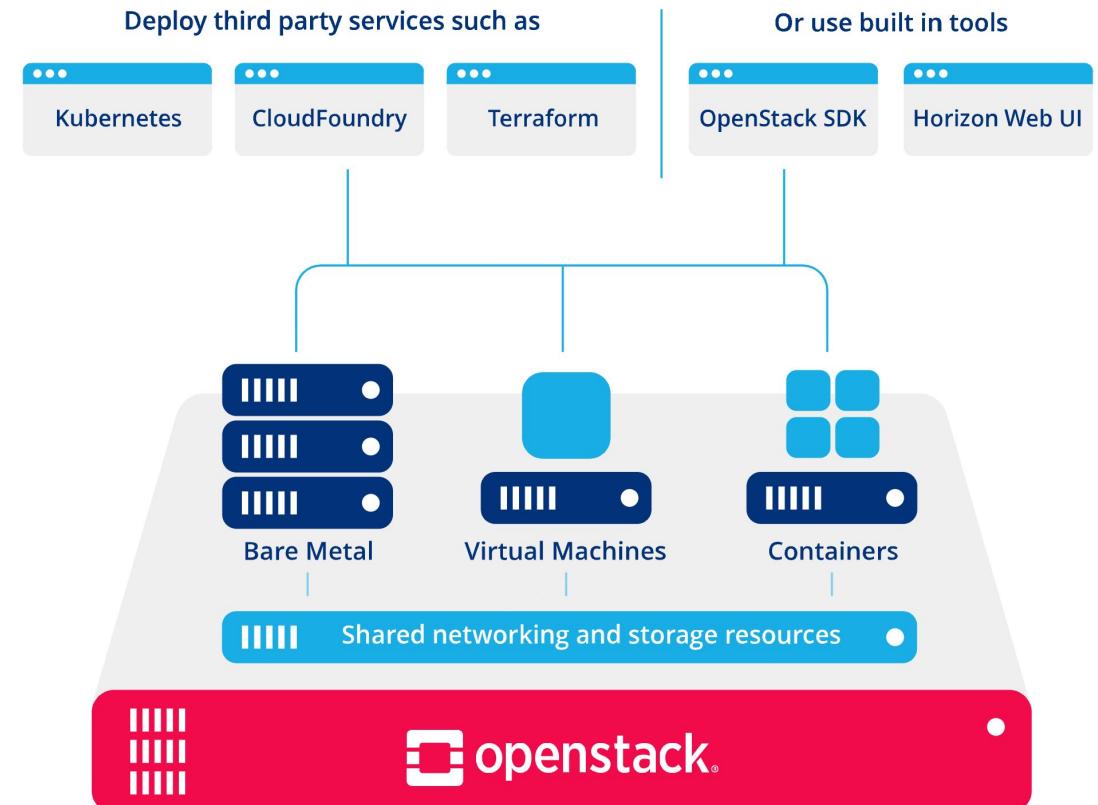
The background of the slide features a complex, abstract network graph composed of numerous small, semi-transparent white dots connected by thin white lines. These dots form a dense web of triangles and larger polygons, creating a sense of connectivity and data flow across the dark blue background.

# IaaS mit OpenStack

# OpenStack

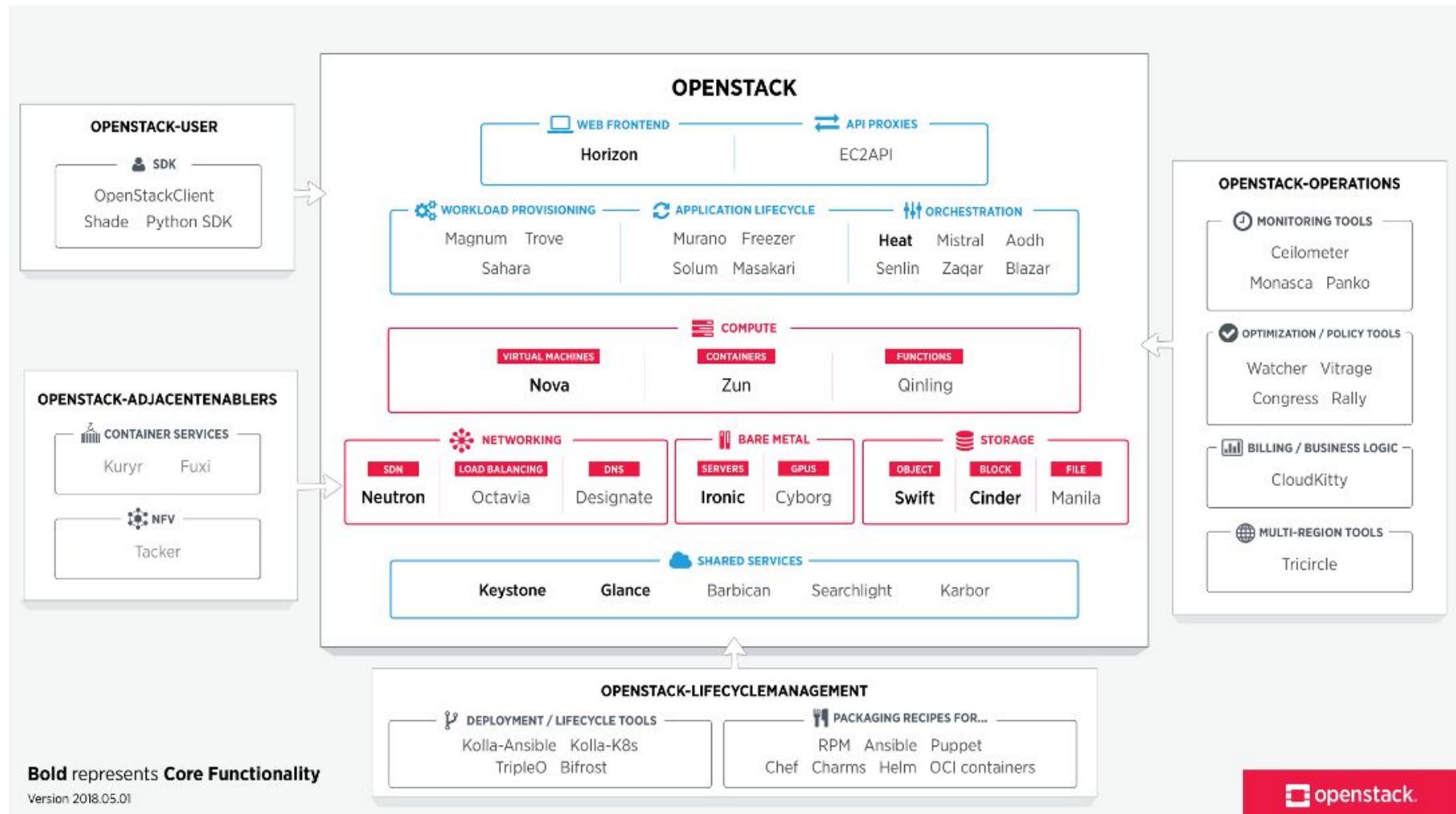
## Der de-facto Standard für Open-Source Private IaaS Clouds

- Open Source Projekt wurde maßgeblich von RackSpace und der NASA initiiert
- Das erste vollständige Release erfolgte im Oktober 2010
- Lizenziert unter der Apache Lizenz
- Eine Vielzahl der klassischen IT-Player (SAP, IBM, VMware, HP, Oracle, Cisco) sind Teil der OpenStack-Community
- Sehr aktives Open-Source-Projekt mit > 400 aktiven Committern
- Ausgelegt eher als Framework denn als fertiges System für IaaS-Clouds



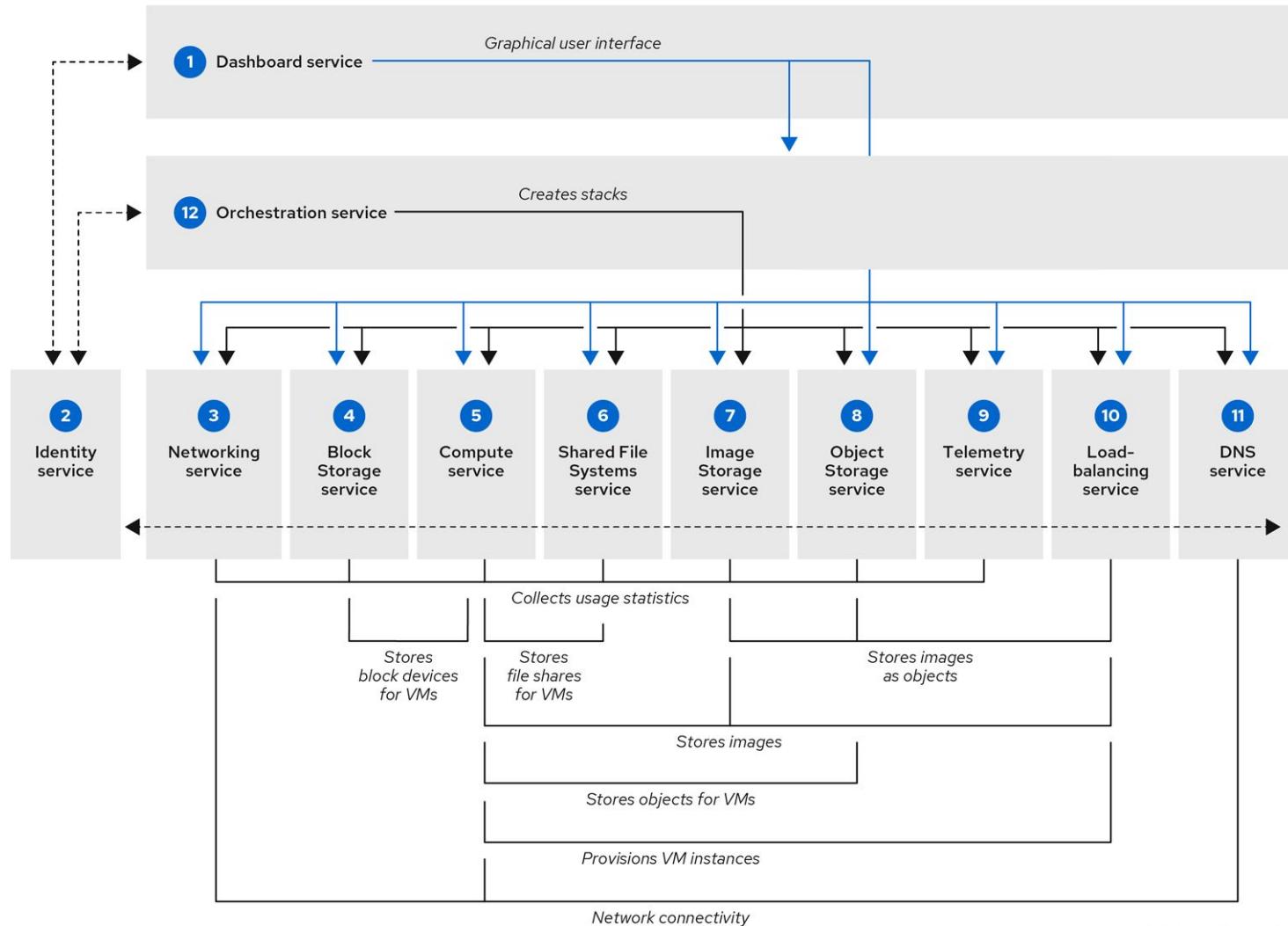
Quellen: <https://www.openstack.org>

# Die OpenStack Komponenten



Quelle: <https://docs.openstack.org/contributors/common/introduction.html>

# Das Zusammenspiel der Kern-Komponenten in OpenStack



Quelle:

[https://access.redhat.com/documentation/en-us/red\\_hat\\_openstack\\_platform/17.1/html/introduction\\_to\\_red\\_hat\\_openstack\\_platform/assembly\\_rhosp-software](https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/17.1/html/introduction_to_red_hat_openstack_platform/assembly_rhosp-software)

# Das Zusammenspiel der Kern-Komponenten in OpenStack

	Service	Code	Description
1	Dashboard	horizon	Web browser-based dashboard that you use to manage OpenStack services.
2	Identity	keystone	Centralized service for authentication and authorization of OpenStack services and for managing users, projects, and roles.
3	Networking	neutron	Provides connectivity between the interfaces of OpenStack services.
4	Block Storage	cinder	Manages persistent block storage volumes for virtual machines.
5	Compute	nova	Manages and provisions virtual machines running on hypervisor nodes.
6	Shared File Systems	manila	Provisions shared file systems that multiple compute instances, bare metal nodes, or containers can consume.
7	Image	glance	Registry service that you use to store resources such as virtual machine images and volume snapshots.
8	Object Storage	swift	Allows users to store and retrieve files and arbitrary data.
9	Telemetry	ceilometer	Provides measurements of cloud resources.
10	Load-balancing	octavia	Provides load balancing services for the cloud.
11	DNS	designate	Manages Domain Name System (DNS) records and zones for the cloud.
12	Orchestration	heat	Template-based orchestration engine that supports automatic creation of resource stacks.
13	Key Manager	barbican	REST API designed for the secure storage, provisioning and management of secrets.

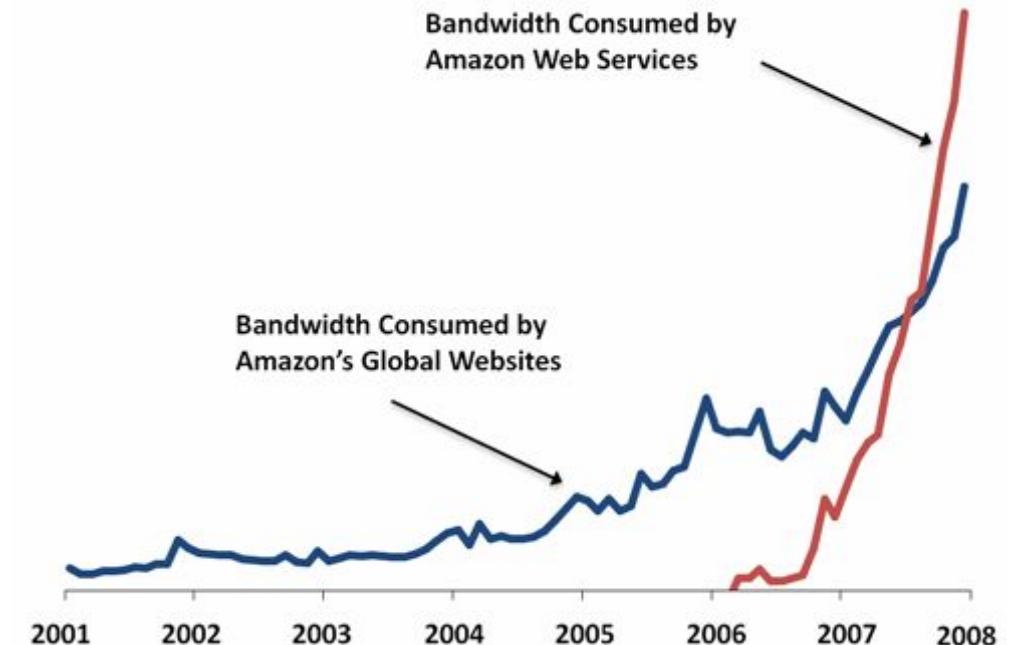
Quelle:

[https://access.redhat.com/documentation/en-us/red\\_hat\\_openstack\\_platform/17.1/html/introduction\\_to\\_red\\_hat\\_openstack\\_platform/assembly\\_rhosp-software](https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/17.1/html/introduction_to_red_hat_openstack_platform/assembly_rhosp-software)

# IaaS mit Amazon EC2

# Amazon EC2 (Elastic Compute Cloud)

- Amazon bietet im Rahmen der AWS (Amazon Web Services) auch eine IaaS-Cloud an.
- Historie:
  - Start innerhalb von Amazon im Jahr 2001
  - Öffentliche Beta ab 25. August 2006
  - Ab Mitte 2007 mehr Bandbreite durch Dritte in der Cloud konsumiert, als durch die Amazon Webseiten
  - Produktionsreife ab 23. 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 und unterschiedlicher Hardware: (<http://aws.amazon.com/de/ec2/instance-types>) sowie diverse Storage- und Netzwerkdienste.

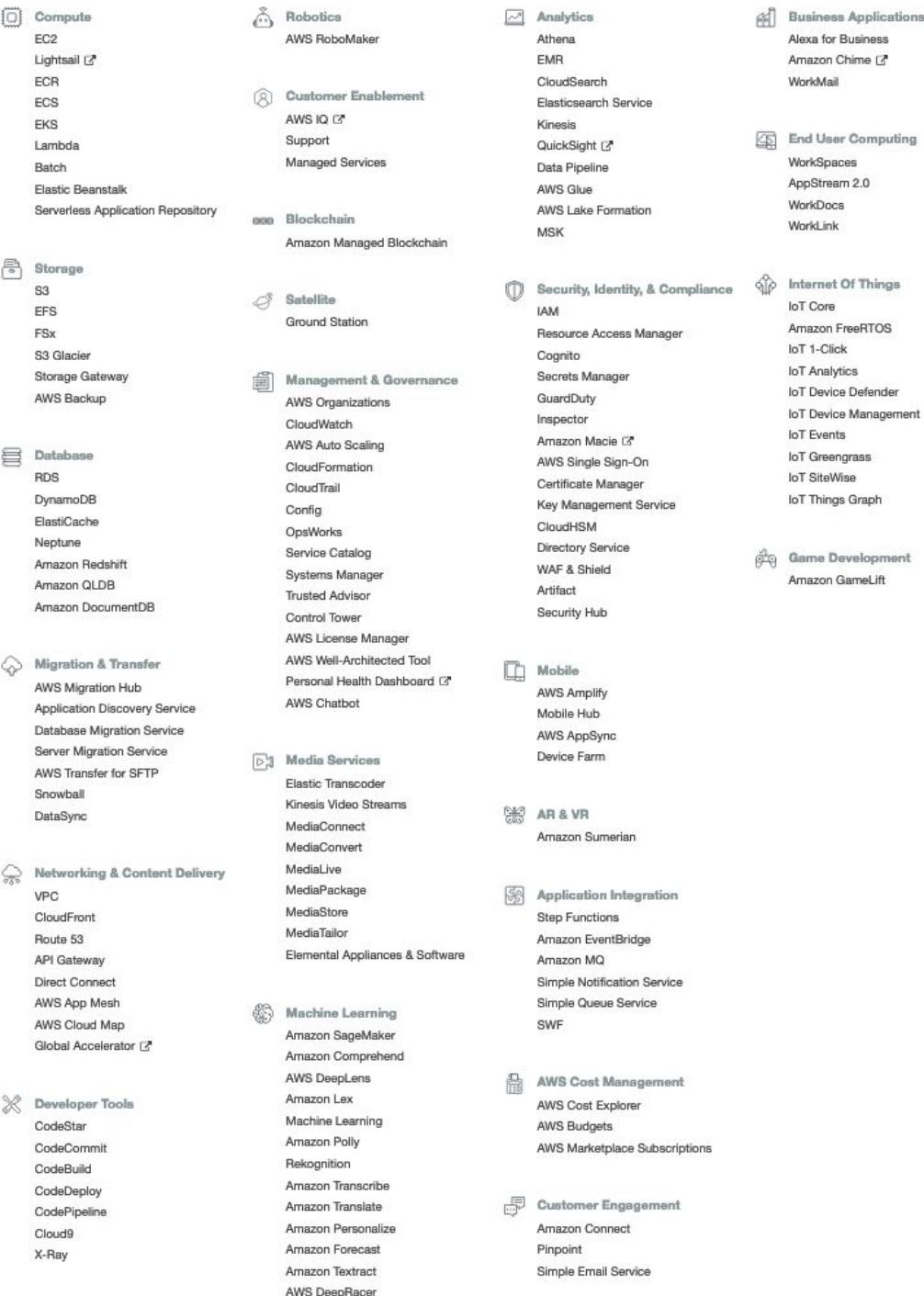


Quelle: <https://aws.amazon.com/blogs/aws/lots-of-bits/>

# Neben der Amazon EC2 IaaS Cloud bietet Amazon noch viele weitere IaaS-Komponenten, PaaS, Serverless-, und SaaS-Dienste.

Aktuelle Anzahl an Services: **353**

Vollständige Auflistung: <https://aws.amazon.com/de/products/>

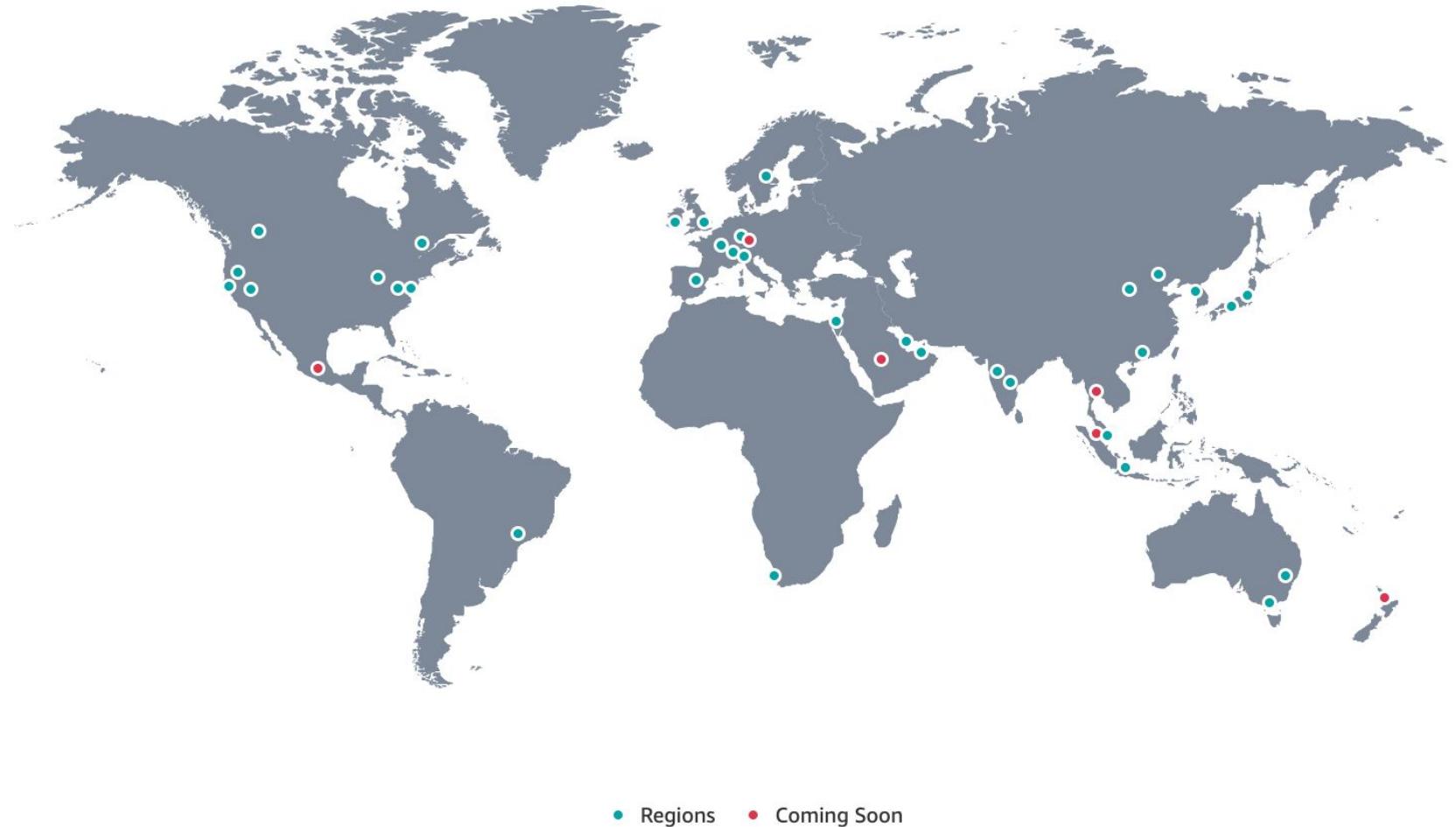


# Die globale Verteilung von AWS

- **33** (+6) Regionen
- **105** (+18) Availability Zones
- **410** Points of Presence

Beispiel:

- eu-central-1 (Frankfurt)
- seit 2014
- 3 Availability Zones

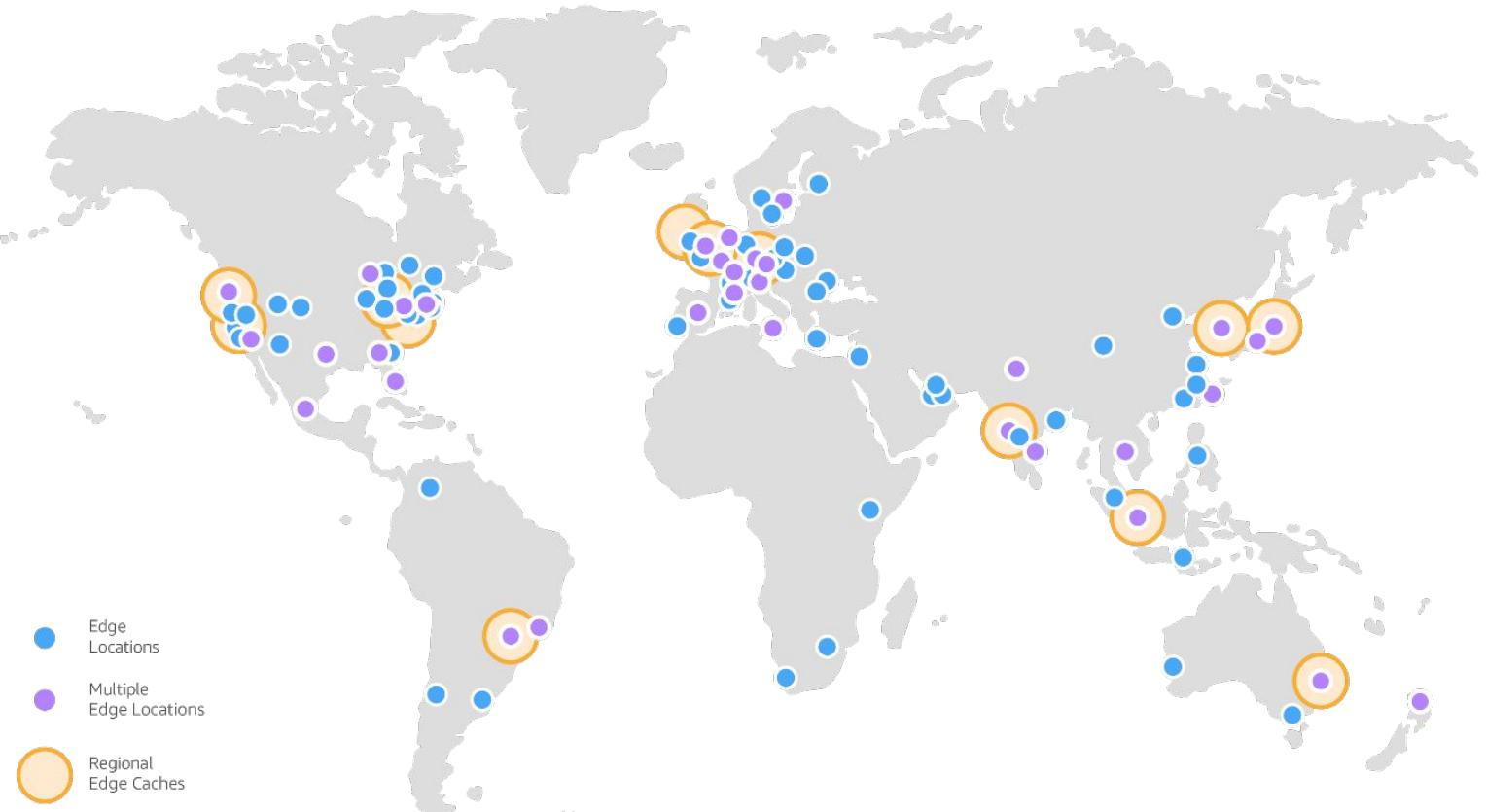


# Die globale Verteilung von AWS

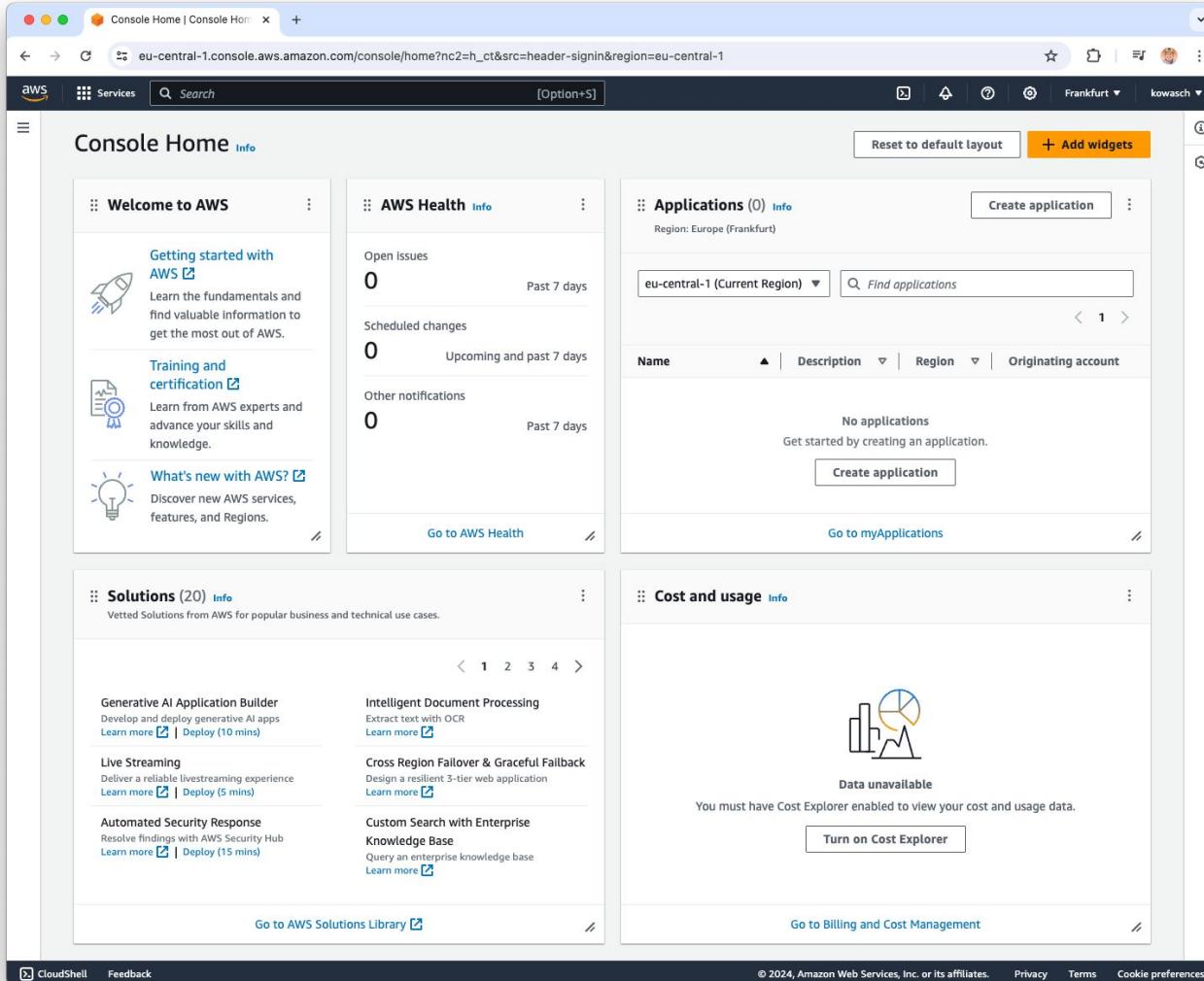
- **33** (+6) Regionen
- **105** (+18) Availability Zones
- **410** Points of Presence

Beispiel:

- eu-central-1 (Frankfurt)
- seit 2014
- 3 Availability Zones



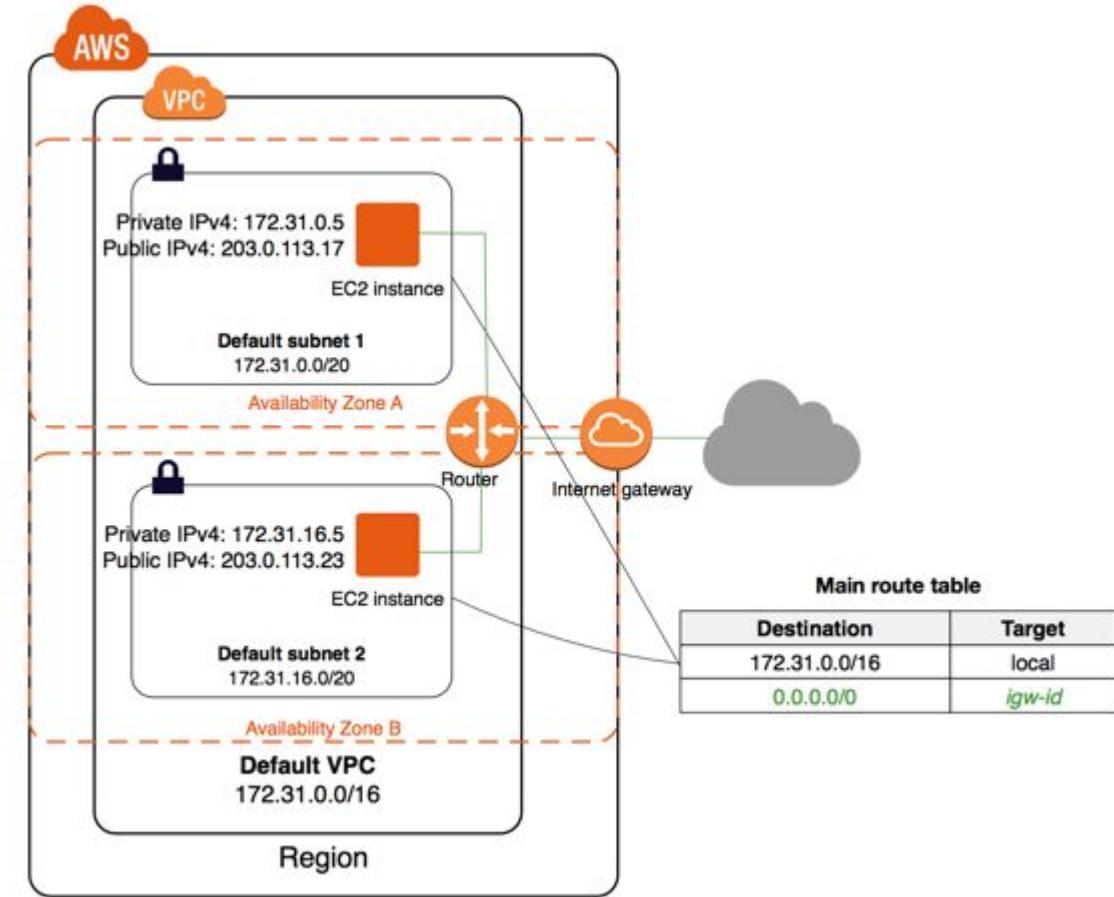
# Über die AWS Management Console können alle Dienste der Amazon-Cloud gesteuert werden.



# Amazon VPC (Virtual Private Cloud)

- Software Defined Network und zwingende Voraussetzung einer IaaS Architektur auf AWS
- Erlaubt das Definieren und stellt Bereit:
  - VPCs und Subnetze
  - Netzwerkinterfaces
  - Security Groups
  - Routing Tabellen
  - Internet Gateways
  - NAT
  - DHCP
  - DNS
  - Elastic IP Adressen
  - ...

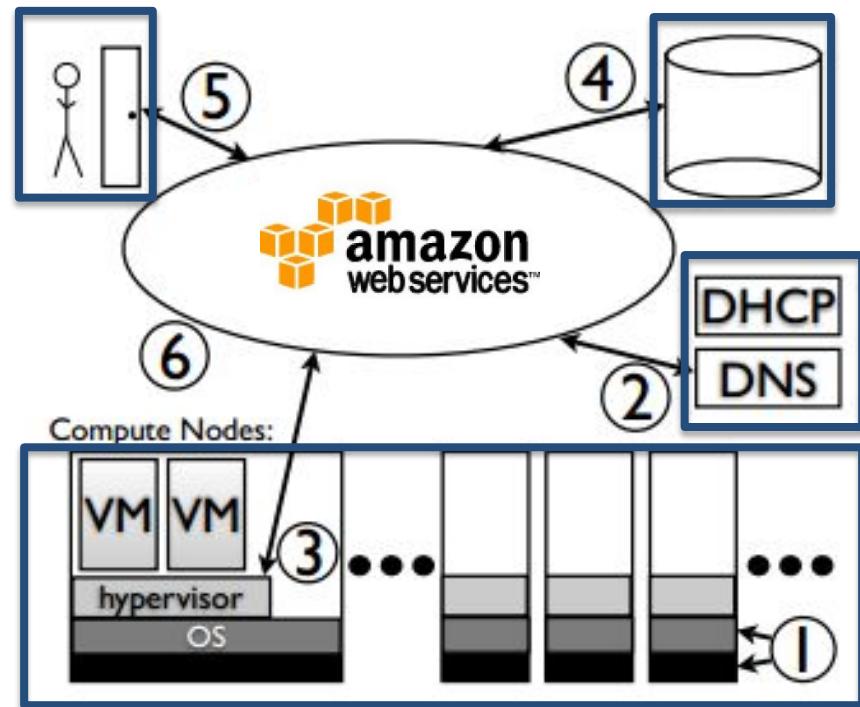
Vorkonfiguriertes Standard Netz:



# Architektur der Amazon EC2

- AWS Management
- Console
- Webservice-API

- EBS (Elastic Block Store)
- S3 (Simple Storage Service)



- VPC (Virtual Private Cloud)
- Route 53
- Elastic Load Balancer
- CloudFront CDN

- EC2-Knoten mit Xen- und HVM-Virtualisierung
- Monitoring über CloudWatch
- AutoScaling auf Basis von CloudWatch-Metriken

# EC2 Metadata Service

- Bereitgestellt vom Hypervisor und Verfügbar auf jeder EC2-Instanz unter <http://169.254.169.254/latest/meta-data>
- Erlaubt einer Instanz Daten abzufragen:
  - über ihre Umgebung
  - sich selbst
  - selbst zu definierende Nutzerdaten
- Ermöglicht die Implementierung von fortgeschrittenen Sicherheits- und Automatisierungsmechanismen
  - Instance Profile über Tokens um AWS Services aufzurufen inkl. automatischen Token tausch
  - Kurzlebige Verwaltung von SSH Keys (EC2 Instance Connect)
  - Taggen von Instanz-Metriken in CloudWatch usw.
  - Provisionierung per Cloud Init

Demonstration (5 Min): <https://www.youtube.com/watch?v=tPQsl8n6er0>

# cloud-init



## cloud-init

*“Cloud-init is the defacto multi-distribution package that handles early initialization of a cloud instance.”*

- seit 2008
- einfaches Init System für die Cloud
- initial nur AWS und Ubuntu
- inzwischen de-facto Standard auf allen Cloud Umgebungen
- Konfiguration über User-Data aus dem Metadata service

# cloud-init - Konfigurationsoptionen

Shell / Bash Skript:

```
#!/bin/sh
echo "Hello cloud-init!"
```

Templated Shell / Bash Skript:

```
## template: jinja
#!/bin/bash
{% if v1.region == 'us-east-2' -%}
echo 'Installing custom proxies for {{ v1.region }}'
sudo apt-get install my-xtra-fast-stack
%- endif %}
```

Cloud-Config:

```
#cloud-config
packages:
  - cowsay
users:
  - default
  - name: app
    groups: docker
write_files:
  - content: nVc+Xj7rPhMqb...
    encoding: b64
    owner: app:app
    path: /home/app/application.yml
    permissions: '0655'
```

# EC2 AutoScaling



EC2 AutoScaling ermöglicht das Organisieren von Instanzen in **Gruppen** als logische Einheit. Beispielsweise lässt sich über die Gruppe die Anzahl der Instanzen steuern und damit bei Ausfall einzelner automatisch neue erzeugen.



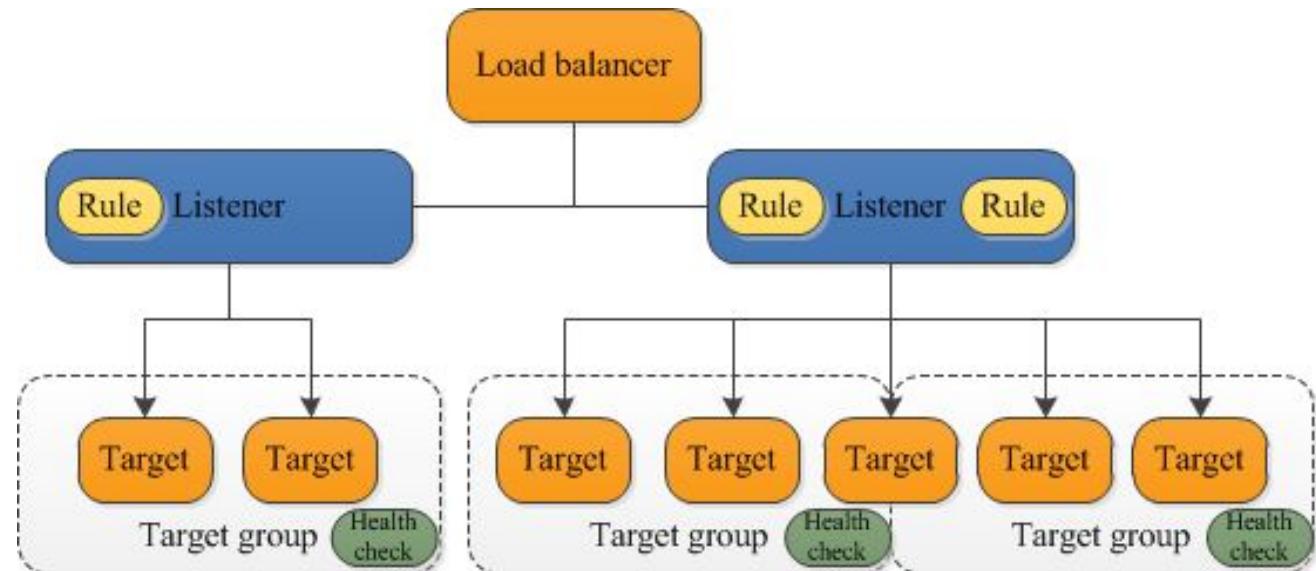
EC2 AutoScaling Gruppen nutzen **Launch Templates** um neue Instanzen zu erzeugen oder rollierend auszutauschen. Launch Templates definieren Instanzparameter wie AMI ID, Instanztyp, Security Group oder Block Device Mappings.



**Skalierungsoptionen** erlauben das automatische Skalieren der Instanzen abhängig von Bedingungen, z.B. CPU Last, zeitliche Steuerung oder Vorhersage.

# Elastic Load Balancing

- Nimmt öffentlichen Verkehr entgegen und verteilt diesen auf Instanzen.
- Überwacht die Funktionalität der Instanzen / Applikation (Health Check) und verteilt die Anfragen / Verbindungen nur auf "gesunde" Ziele.
- Verschiedene Varianten:
  - Application Load Balancer - Layer 7
  - Network Load Balancer - Layer 4
  - Classic Load Balancer - Legacy
- Unterstützen TLS, Integration mit AutoScaling usw.

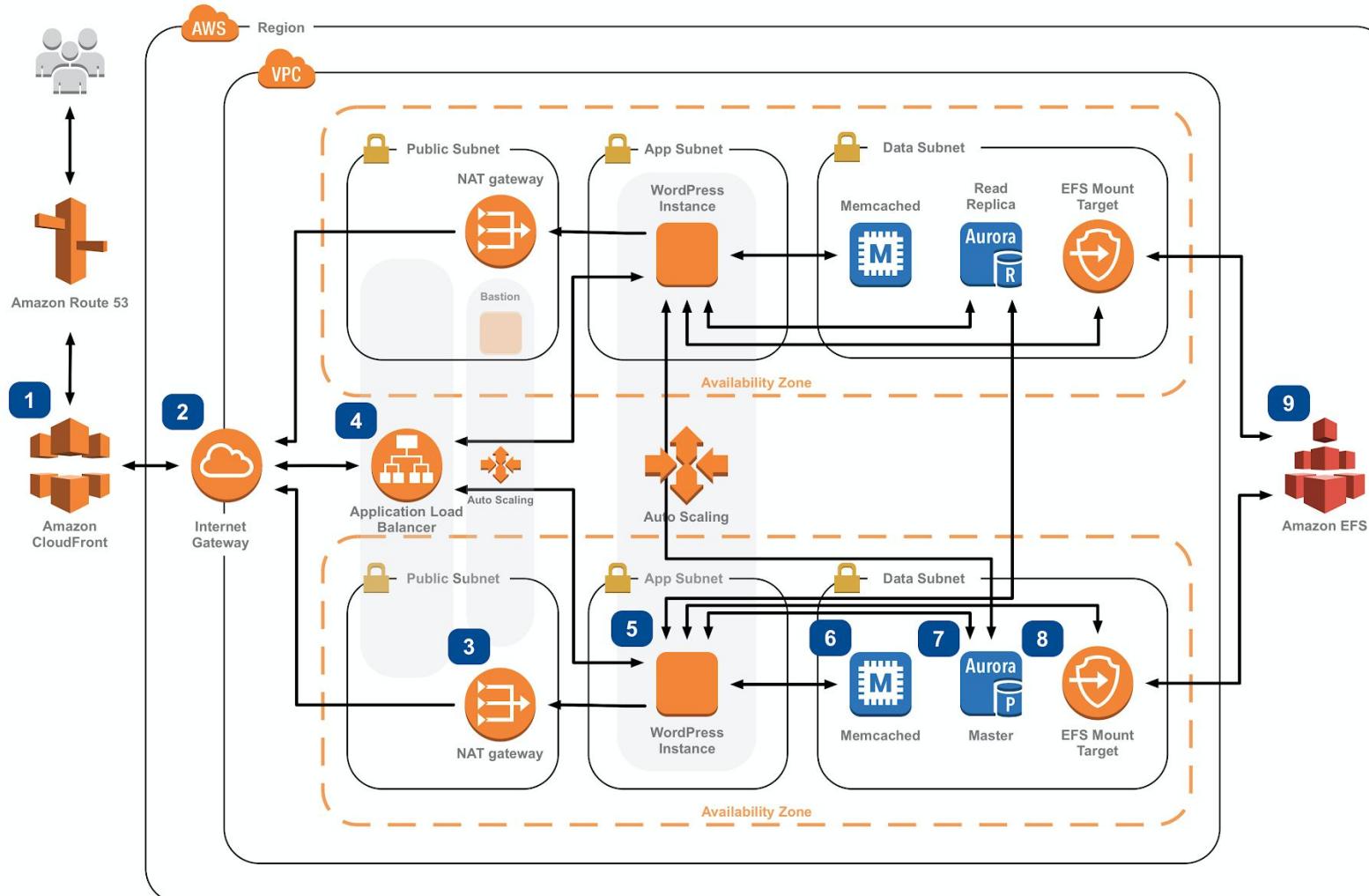


Beispiel: Application Load Balancer

# WordPress Hosting

## How to run WordPress on AWS

WordPress is one of the world's most popular web publishing platforms, being used to publish 27% of all websites, from personal blogs to some of the biggest news sites. This reference architecture simplifies the complexity of deploying a scalable and highly available WordPress site on AWS.



## AWS Reference Architectures

<https://github.com/aws-samples/aws-refarch-wordpress>

© 2017, Amazon Web Services, Inc. or its affiliates. All rights reserved.



# Sicherheitsaspekte der AWS

- Zertifiziert nach ISO 27001 / C5 (Empfehlung BSI) und vielen weiteren Standards:  
<https://aws.amazon.com/de/compliance/programs/>
- Europäische Rechenzentren und Niederlassungen sind den EU-Datenschutzrichtlinien unterworfen.
- Amazon ist ebenso dem US Patriot Act und dem CLOUD Act unterworfen.
- AWS bietet Dienste und Produkte für die Umsetzung von Sicherheits- und Complianceanforderungen:
  - Identity and Access Management: IAM, Single Sign-On, Cognito...
  - Erkennung: GuardDuty, Config, CloudTrail...
  - Infraukturschutz: Shield, Firewall für Webanwendungen, Firewall Manager
  - Datenschutz: KMS, CloudHSM, Macie...
  - Vorfallreaktion: Detective, CloudEndure Disaster Recovery
  - Compliance: Artifact

# Infrastructure as Code

# Infrastructure as Code

- Provisionieren und Managen ganzer Rechenzentren - nicht nur einzelne virtuelle Maschinen
- Abgrenzung zu Configuration Management (z.B. Ansible):
  - Explizite Erzeugung und Zerstörung der Infrastruktur eines (virtuellen) Rechenzentrums
  - Immutable Infrastructure, statt kontinuierlichem ändern existierender Ressourcen
  - Typischerweise Deklarativ statt Imperativ
- Erstmals für die Cloud in 2010 mit AWS CloudFormation

# Infrastructure as Code

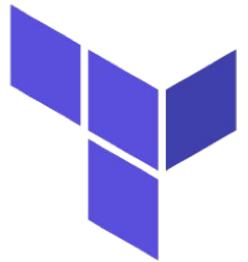
Vorteile:

- Versionierung des Rechenzentrums und damit einfaches Staging und Rollbacks
- Beschleunigte Auslieferung von Infrastrukturänderungen
- Konsistenz über Umgebungen hinweg
- Dadurch auch Sicherheit und Auditierbarkeit der Infrastruktur im Code
- Wiederverwendbar und Modularisierbar
- Ermöglicht Kollaboration über Code Verwaltung

# Infrastructure as Code - Beispiele

Tool	Plattform	Beschreibung
Terraform (Hashicorp)	Multi-Cloud (AWS, Azure, GCP, ....)	Eigenes Format (HCL), viele unterstützte Anbieter
Pulumi	Multi-Cloud	Ähnlich wie Terraform, aber erlaubt <b>Programmierung in TypeScript, Python, Go, .NET</b>
ARM Templates	Azure	JSON-basierte, native IaC-Lösung von Azure
Bicep	Azure	Neue deklarative Sprache von Microsoft, speziell für Azure
AWS CDK (Cloud Development Kit)	AWS	<b>Programmierung in TypeScript, Python, Go, .NET</b>
Cloudformation	AWS	YAML basierte IaC-Lösung
Google Cloud Deployment Manager	GCP	YAML/JSON-basierte IaC-Lösung speziell für GCP. Vergleichbar mit CloudFormation in AWS.

# Infrastructure as Code mit Terraform



HashiCorp

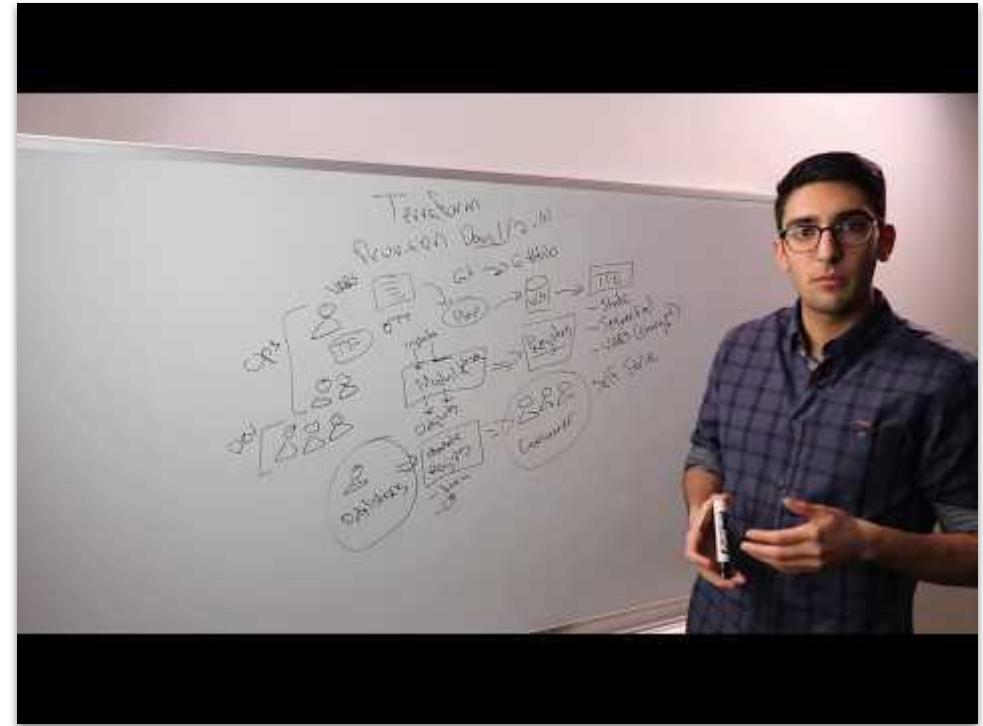
# Terraform

"Write, Plan, and Create  
Infrastructure as Code"

- Seit 2014, Open Source von Hashicorp
- Unterstützt die gängigen Cloud Provider
- Darüber hinaus noch Integrationen in Datenbanksysteme, Monitoring- und Infrastruktur-Software wie z.B. Kubernetes
- Große Auswahl von Plugins und wiederverwendbaren Modulen
- Deklarative Konfigurationssprache
- Kommerzielle Erweiterungen erhältlich

# Terraform Grundlagen

- **Write:** Beschreibung Zielzustand über eine domänenspezifische Sprache HCL (HashiCorp Configuration Language)
- **Plan** (terraform plan): Ist-Zustand ermitteln. Notwendige Änderungen planen (entsprechend Abhängigkeiten geordnet und parallelisiert, Unterbrechungen möglichst minimal)
- **Apply** (terraform apply): Idempotente Herstellung des Zielzustands. Der Zustand (.tfstate Datei) wird meist in einem Remote Storage (S3, HTTP, ...) gespeichert



Video Pause bei 06m:30s, dann weiter bis 10m18s:  
<https://www.youtube.com/watch?v=h970ZBgK1Ng>

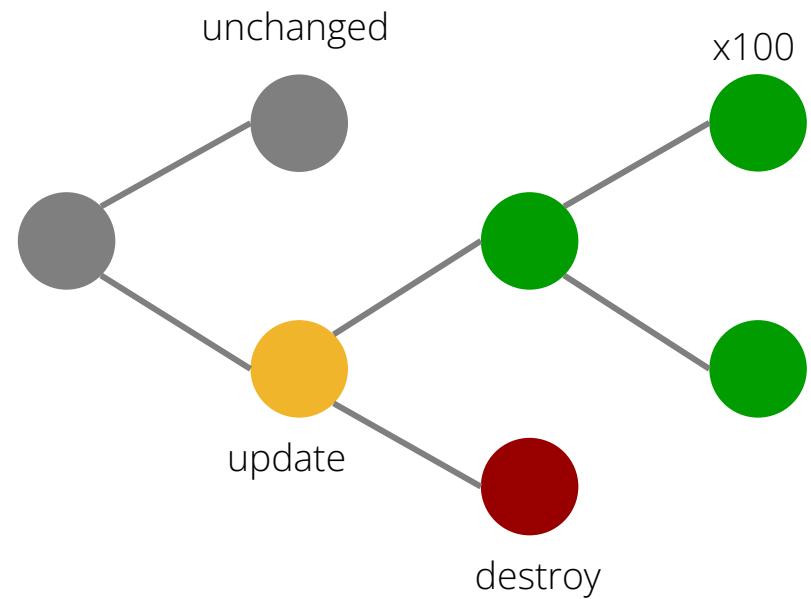
# Terraform Core Funktionsweise

- **Plan**

- Einlesen der Benutzerkonfiguration (.tf-Dateien)
- Einlesen des aktuellen Zustands (Terraform State)
- Aufbau eines Abhängigkeitsgraphen an *Ressourcen*
  - Create, Update, Destroy

- **Apply**

- Ausführung des Plans durch *Provider*



# Die Kern-Entitäten einer Terraform-Konfiguration

Ressource: zu verwaltende Komponente der Infrastruktur

```
resource "aws_instance" "web" {  
    ami           = "ami-408c7f28"  
    instance_type = "t1.micro"  
}
```

Haben Argumente und Attribute und definieren Abhängigkeiten zueinander, die einen gerichteten Graphen formen.

Provider: Integration der zu provisionierenden Infrastruktur oder Software.

Alicloud	Archive	AWS
Azure	Bitbucket	CenturyLinkCloud
Chef	Circonus	Cloudflare
CloudScale.ch	CloudStack	Cobbler
Consul	Datadog	DigitalOcean
DNS	DNSMadeEasy	DNSimple

```
provider "aws" {  
    access_key = "${var.aws_access_key}"  
    secret_key = "${var.aws_secret_key}"  
    region     = "us-east-1"  
}
```

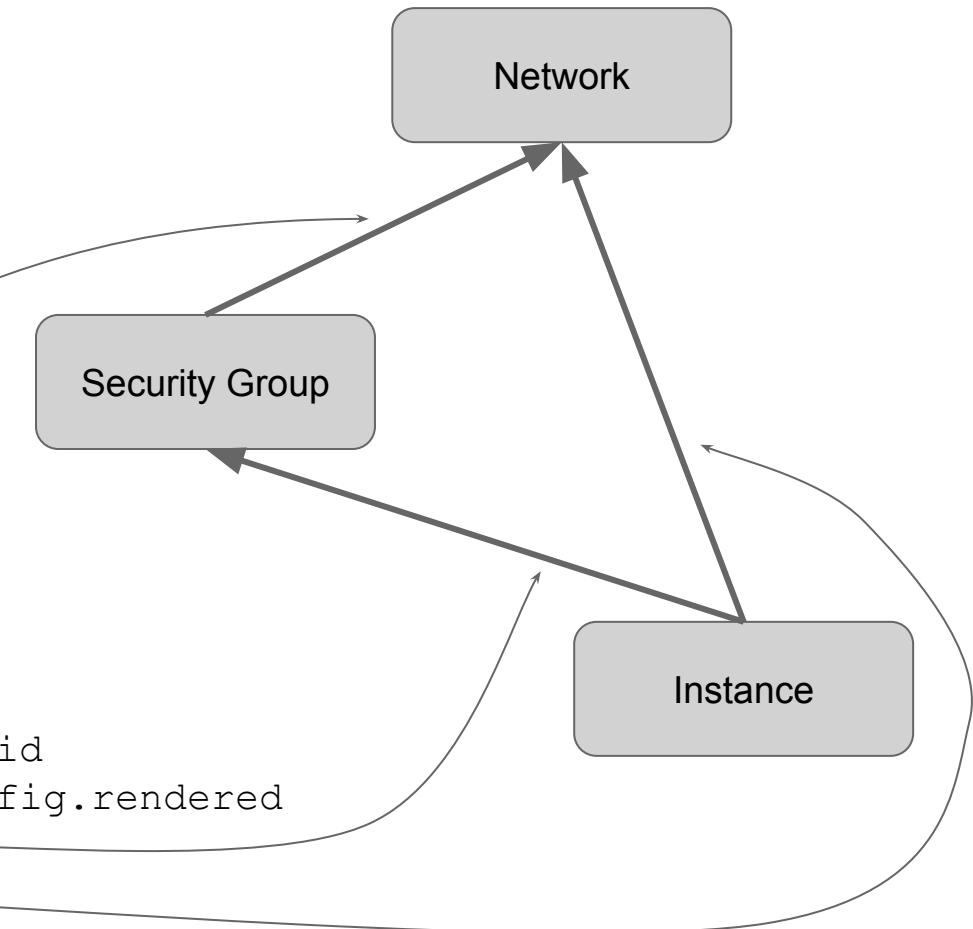
Nomad	NS1	Null
1&1	OpenStack	OpenTelekomCloud
OpsGenie	Oracle Public Cloud	Oracle Cloud Platform
OVH	Packet	PagerDuty
Palo Alto Networks	PostgreSQL	PowerDNS
ProfitBricks	RabbitMQ	Rancher
Random	Rundeck	Scaleway
SoftLayer	StatusCake	Spotinst
Template	Terraform	Terraform Enterprise
TLS	Triton	UltraDNS
Vault	VMware vCloud Director	VMware NSX-T

Provisioner: Erlauben das Ausführen von Aktionen im Graphen durch lokale oder remote Code Ausführung. Nur in **Ausnahmefällen** zu verwenden, wenn ein Provider an seine Grenzen kommt.

```
resource "aws_instance" "web" {  
    # ...  
  
    provisioner "local-exec" {  
        command = "echo ${self.private_ip} > file.txt"  
    }  
}
```

# Beispiel Hashicorp Configuration Language

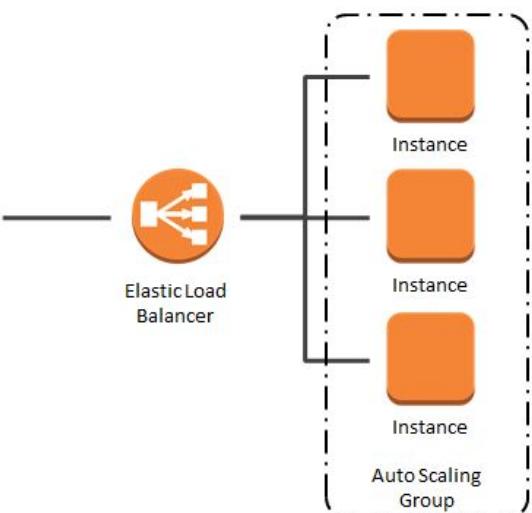
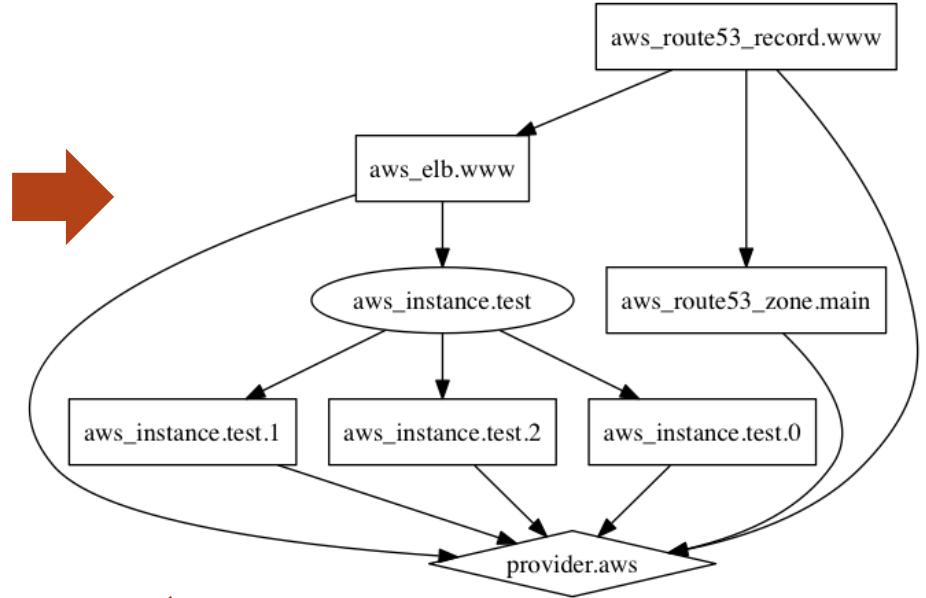
```
module vpc {  
    source  = "terraform-aws-modules/vpc/aws"  
    version = "2.18.0"  
    name    = local.env  
    # <shortened>  
}  
  
resource aws_security_group bastion {  
    name          = "${local.env}-bastion"  
    description   = "For Bastion Hosts"  
    vpc_id        = module.vpc.vpc_id  
    # <shortened>  
}  
  
resource aws_instance bastion {  
    instance_type      = "t3.nano"  
    ami                = data.aws_ami.amazon_linux_2.image_id  
    user_data_base64   = data.template_cloudinit_config.config.rendered  
    vpc_security_group_ids = [aws_security_group.bastion.id]  
    subnet_id          = module.vpc.public_subnets[0]  
    # <shortened>  
}
```



# Workflow

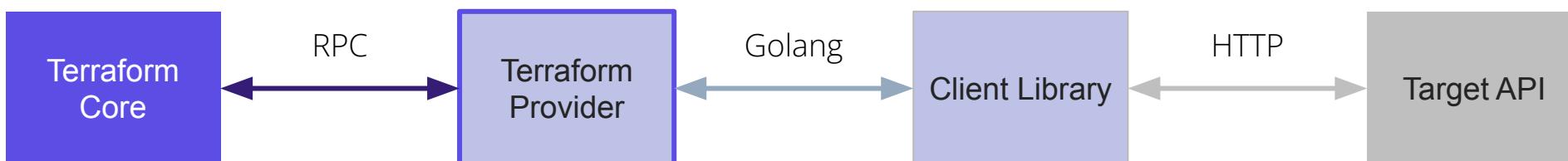
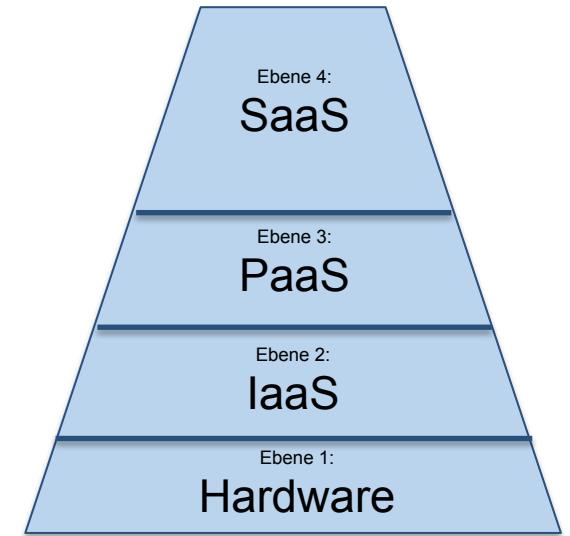
```
resource aws instance bastion {  
  instance type          = "t3.nano"  
  ami                  = data.aws ami.amazon linux 2.image id  
  user data base64      = data.template cloudinit config.config.rendered  
  vpc security group ids = [aws security group.bastion.id]  
  subnet id             = module.vpc.public_subnets[0]  
  # <shortened>  
}
```

terraform/  
└── main.tf  
└── terraform.tfvars

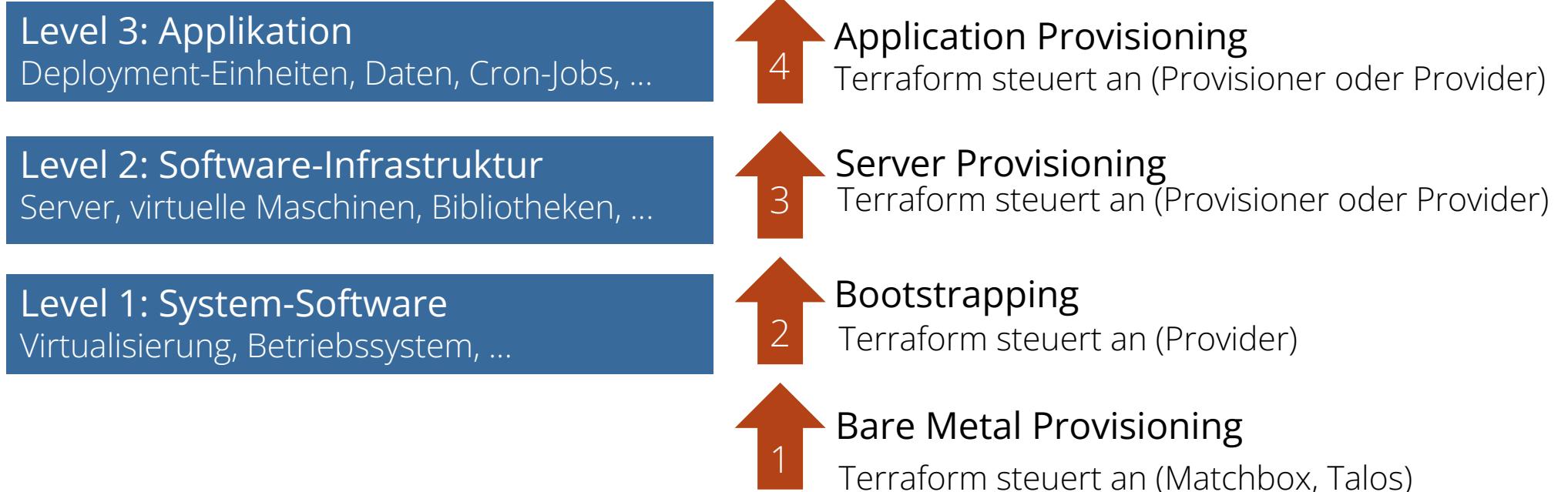


# Terraform Provider

- Sind Go-Plugins, mit denen TF Core über RPC kommuniziert
- Übersetzen TF CRUD in entsprechende Befehle der Client-Library
- Sind nicht auf IaaS beschränkt, es gibt sie für verschiedene Ebenen des bekannten Schichtenmodells:
  - IaaS: AWS, Azure, OpenStack, VMware, Matchbox
  - PaaS: Heroku, K8s, Lambdas
  - SaaS: DataDog (Monitoring), Fastly (CDN)
- Es gibt auch exotische Provider, z.B. für Spotify, Jira, ...
- Schreiben Sie Ihren eigenen Provider!



# Terraform Deployment Ebenen



# Beispiel: Provider

```
provider aws {  
    version = "2.56.0"  
    region = "eu-central-1"  
}
```

# Beispiel: Data

```
data aws_ami amazon_linux_2 {
    most_recent = true

    owners = ["amazon"]

    filter {
        name    = "name"
        values = ["amzn2-ami-hvm*"]
    }

    filter {
        name    = "root-device-type"
        values = ["ebs"]
    }

    filter {
        name    = "architecture"
        values = ["x86_64"]
    }
}
```

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/data-sources/ami>

# Beispiel: Resources

```
resource aws_instance bastion {
  instance_type          = "t3.nano"
  ami                     = data.aws_ami.amazon_linux_2.image_id
  user_data_base64        = data.template_cloudinit_config.config.rendered
  vpc_security_group_ids = [aws_security_group.bastion.id]
  subnet_id               = module.vpc.public_subnets[0]
  tags = merge(local.standard_tags, {
    "Name" = "${local.env}-bastion"
  })
}
```

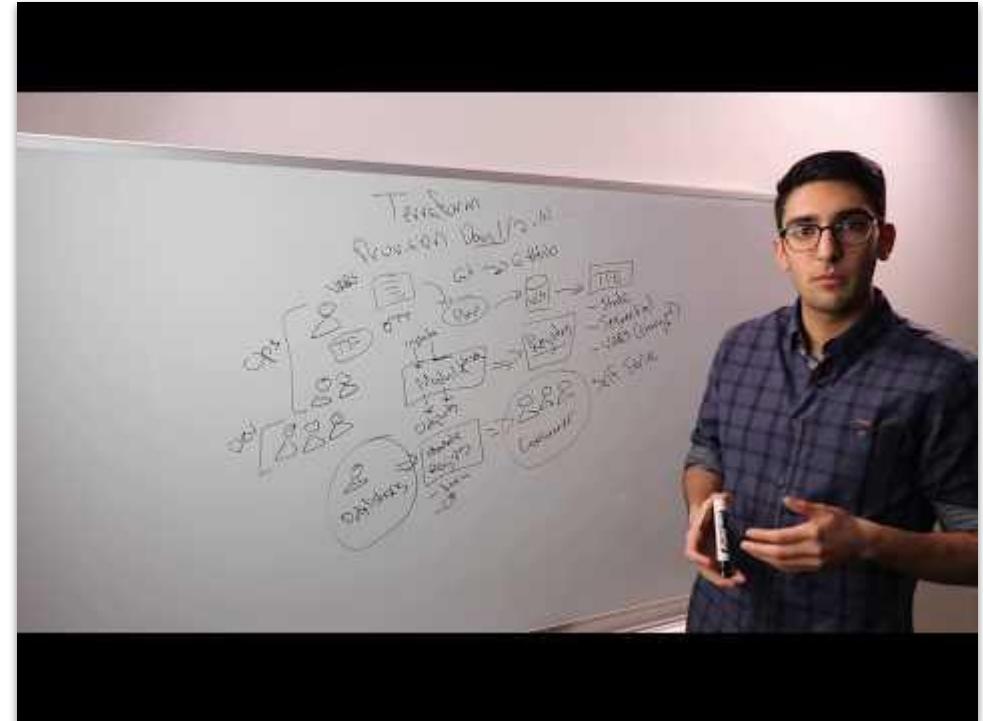
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance>  
<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

# Beispiel: Output

```
output "bastion" {
  description = "Information about the bastion host."
  value = {
    instance      = aws_instance.bastion.id
    dns_name      = aws_route53_record.bastion.name
    key_push_policy = aws_iam_policy.allow_bastion_ssh_access.arn
  }
}
```

# Module: Wiederverwendbare Abstraktionen

- Abgeschlossene Terraform Deklaration
- Enthält multiple Ressourcen-Definitionen
- "Inputs" sind die Parameter der Modul-Ressource
- "Output" sind die Attribute der Modul-Ressource
- Versioniert
- Veröffentlichung über <https://registry.terraform.io> oder Git-Repository



Video ab 13m53s, bis Ende:  
<https://www.youtube.com/watch?v=h970ZBgK1Ng>

# Beispiel: Modul

```
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"

  name      = "my-vpc"
  cidr     = "10.0.0.0/16"

  azs          = ["eu-west-1a", "eu-west-1b", "eu-west-1c"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets   = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  enable_vpn_gateway = true

  tags = {
    Terraform = "true"
    Environment = "dev"
  }
}
```

<https://registry.terraform.io/modules/terraform-aws-modules/vpc/aws/latest>

# Kollaboration und Zustand

- Terraform ist Zustandsbehaftet und speichert diesen Zustand normalerweise in der Cloud.  
Lokal ist möglich, aber nicht empfohlen.
  - Ermöglicht dadurch z.B., dass das Entfernen von Ressourcen im Code auch eine logische Entfernung der Cloud-Ressource bedeutet. Die Referenz im Zustand ist in diesem Fall der Unterschied zum Zielzustand (Drift).
  - Remote State Locking verhindert das parallele Anwenden von Änderungen
- Der Zustand kann schützenswerte Geheimnisse enthalten, z.B. Passwörter und Zertifikate

```
terraform {  
  backend s3 {  
    bucket          =  
    "prj-aws-dev-tfstate-storage"  
    key             = "prj-main"  
    dynamodb_table = "prj-tfstate-lock"  
    region          = "eu-central-1"  
  }  
}
```



Video ab 10m18s, bis 13m:15s:  
<https://www.youtube.com/watch?v=h970ZBgK1Ng>

# Terraform State

The screenshot illustrates the location of a Terraform state file within an AWS S3 bucket and its corresponding path in a local project structure.

**AWS S3 Console Path:** [Amazon S3](#) > [Buckets](#) > [REDACTED] > [REDACTED]-tfstate > dev/ > eu-central-1/ > int/ > cbs-icons/

**Local Project Structure:**

```
Project
  └── [REDACTED]
    └── .github
    └── .idea
    └── aws
      └── cn-dev
        └── dev
          └── .pluralith
          └── _global
            └── eu-central-1
              └── _global
                └── int
                  └── argo
                  └── atlantis
                  └── cbs-icons
                  └── check-control-messages
                  └── data-supply
```

An arrow points from the "dev/" part of the S3 path to the "dev" folder in the local project structure, indicating they represent the same environment.

# Terraform State

```
# Terraform source code ...
module "aurora_serverless" {
    source =
"git:::git@your-git-repo.com:tf-modules.git//db?ref=db-3.6.2"

    cluster_name =
"${var.hub}-${var.env}-${var.service_name}"

    vpc_id          = var.vpc_id
    database_subnets = var.database_subnet_ids
    allowed_cidr_blocks = var.private_subnet_ranges
<shortend>
```

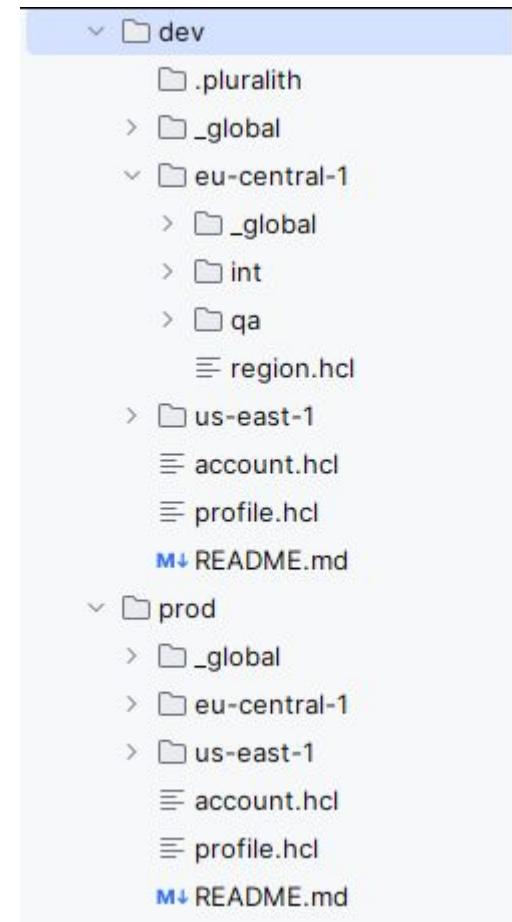
```
# Auszug aus terraform.tfstate
{
    "module": "module.aurora_serverless.module.db" ,
    "mode": "managed",
    "type": "aws_rds_cluster_instance" ,
    "name": "this",
    "provider": "provider[\"registry.terraform.io/hashicorp/aws \"]",
    "instances": [
        {
            "index_key": "one",
            "schema_version": 0,
            "attributes": {
                "apply_immediately": true,
                "arn": "arn:aws:rds:eu-central-1:ACCOUNTID:db:emea-int-wordpress-one" ,
                "auto_minor_version_upgrade": true,
                "availability_zone": "eu-central-1b",
                "ca_cert_identifier": "rds-ca-rsa2048-g1",
                "cluster_identifier": "emea-int-wordpress",
                "copy_tags_to_snapshot": false,
                "custom_iam_instance_profile": "",
                "db_parameter_group_name": "emea-int-wordpress-postgres-db-parameter-group",
                <shortend>
```

# Workspaces / Organisation

- Umgebungen wie Staging/QA/Produktion können über eine Ordnerstruktur oder über Workspaces organisiert sein
- Mehrere Zustände (Workspaces) für dieselbe Konfiguration ermöglichen:
  - einfaches Staging, pro Umgebung ein Workspace
  - komplett unabhängige Entwicklung
  - Steuerung über CLI

```
variable "instance_type" {
  type = string
  default = "t3.micro"
}

locals {
  instance_type = terraform.workspace == "prod" ?
    "t3.large" : var.instance_type
}
```



# Beispiel: Struktur

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

## Basis-Resourcen

Gemeinsam genutzte und gleich aufgebaute Komponenten aller Umgebungen

## QA = Qualitätsabsicherungsumgebung

Spezifische Ressourcen für diese Umgebung

## Produktionsumgebung

Spezifische Ressourcen für diese Umgebung