CLOUD-NATIVE PROGRAMMIERUNG

Unit 03:

Infrastructure as Code

Stand: 09.10.2020

TECHNISCHE HOCHSCHULE LÜBECK

1

INHALTSVERZEICHNIS

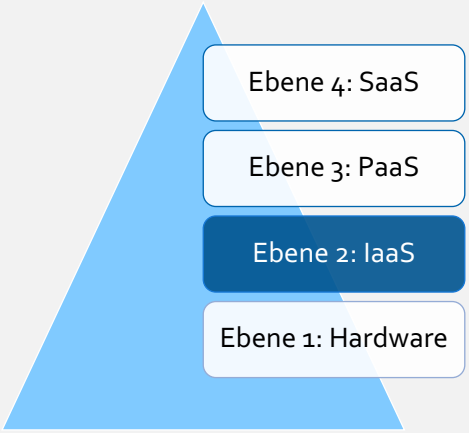
Überblick über Units und Themen dieses Moduls

TECHNISCHE HOCHSCHULE LÜBECK

2

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

TECHNISCHE HOCHSCHULE LÜBECK

INHALTE

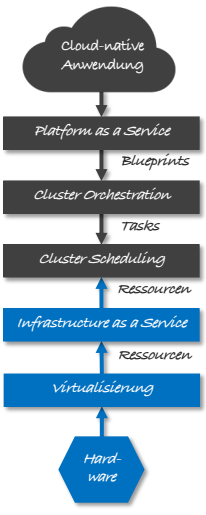
Virtualisierung
- Hardware-Virtualisierung (Typ-1, Typ-2)
- Betriebssystem-Virtualisierung
- Applikations-Virtualisierung
- Emulation

Infrastructure as a Service
- Definition, Eigenschaften, Marktüberblick
- Private Cloud Infrastrukturen
- Public Cloud Infrastrukturen (am Bsp. des Typvertreters AWS)

Provisionierung in IaaS-basierte Infrastrukturen
- Historische Entwicklung und
- Ebenenmodell
- Immutable Infrastructures

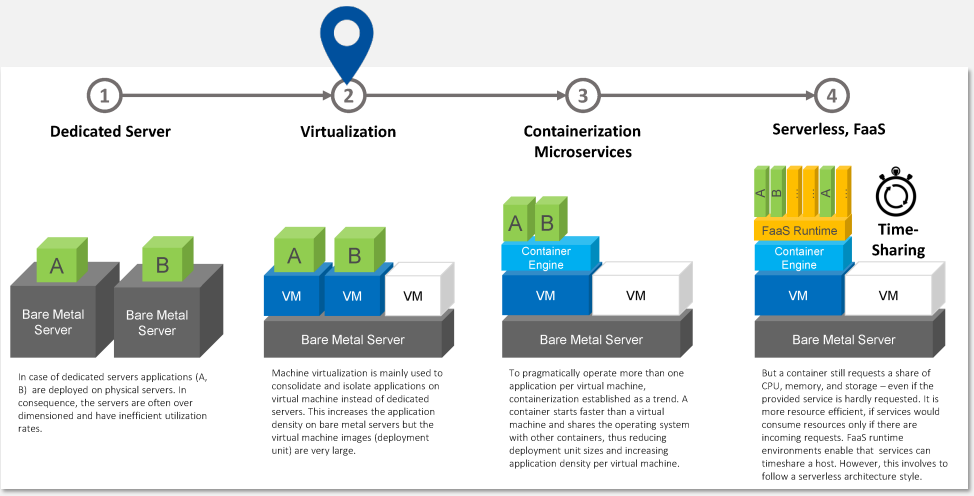
Infrastructure as Code
- Definition, Ansätze und Methoden
- Immutable Architecture
- Überblick gängiger Provisionierungswerkzeuge
- Single VM-Provisioning mittels Vagrant (Deklaratives Push-basiertes IaC)
- Multi VM-Provisioning mittels Terraform (Deklaratives Push-basiertes IaC)

TECHNISCHE HOCHSCHULE LÜBECK



EINE KURZE GESCHICHTE DER CLOUD

Wo sind wir?



TECHNISCHE HOCHSCHULE LÜBECK

5

VIRTUALISIERUNG

TECHNISCHE HOCHSCHULE LÜBECK

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.

6

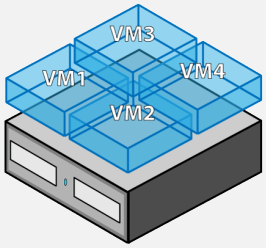
VIRTUALISIERUNGSARTEN

TECHNISCHE HOCHSCHULE LÜBECK

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)
- Anwendendungs-Virtualisierung (Runtime, z.B. die Java Virtual Machine)

VIRTUALISIERUNG

Im Cloud Computing

TECHNISCHE HOCHSCHULE LÜBECK

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/

A1 T3 T3a T2 M6g M5 M5a M5n **M4**

Die M4-Instances bieten ausbalancierte Datenverarbeitungs-, Arbeitsspeicher- und Netzwerkressourcen und eignen sich für zahlreiche Anwendungen.

Funktionen:

- 2,3-GHz-Prozessoren Intel Xeon® E5-2686 v4 (Broadwell) oder 2,4-GHz-Prozessoren Intel Xeon® E5-2676 v3 (Haswell)
- Standardmäßig und ohne Zusatzkosten für EBS optimiert
- Unterstützung für Enhanced Networking
- Ausbalancierte Datenverarbeitungs-, Arbeitsspeicher- und Netzwerkressourcen

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?

TECHNISCHE HOCHSCHULE LÜBECK

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

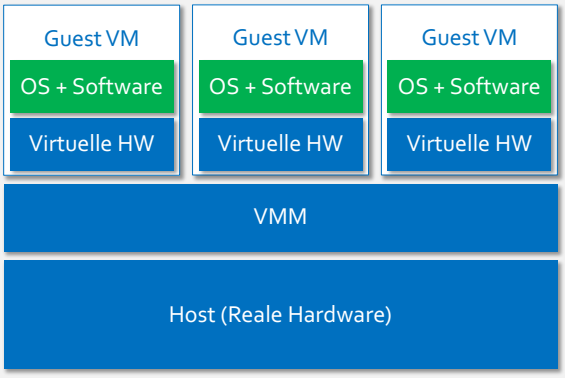
Storage

- 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?

TECHNISCHE HOCHSCHULE LÜBECK



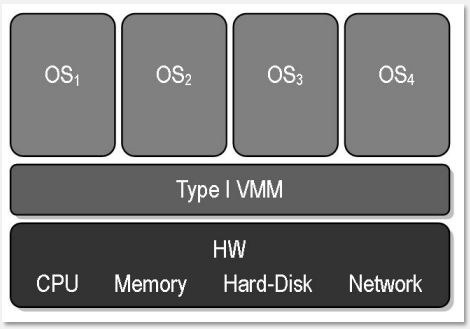
Host: Der physikalische Rechner, der eine oder mehrere VM ausführt und entsprechende HW-Ressourcen zur Verfügung stellt.
Guest: Eine laufende (oder lauffähige) virtuelle Maschine (VM).
VMM (Virtual Machine Monitor): Die Steuerungssoftware zur Verwaltung von Guests und Host-Ressourcen.

- 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 brauch im Hypervisor nicht aufwändig eigene Treiber implementiert zu werden.



TECHNISCHE HOCHSCHULE LÜBECK

Leistungsverlust:
ca. 2-3%

Xen

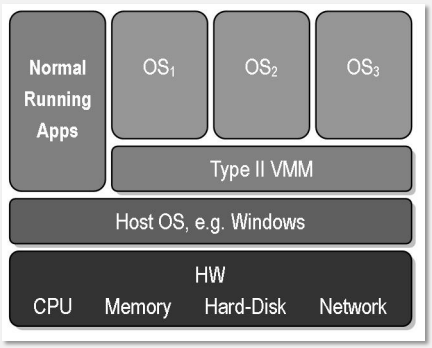
Microsoft Hyper-V

vmware ESXi

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).



TECHNISCHE HOCHSCHULE LÜBECK

Leistungsverlust:
ca. 5-10%

VirtualBox

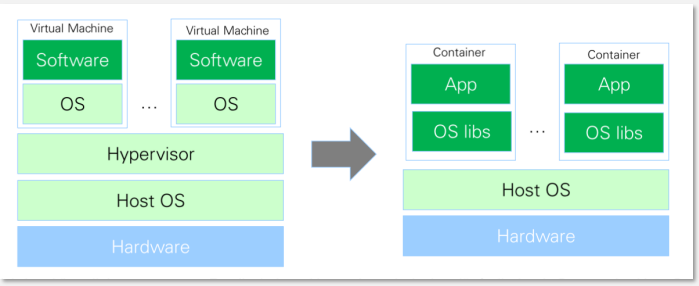
Parallels

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.



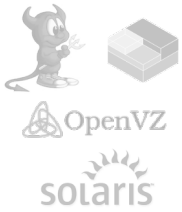
- Free BSD Jails (2000)
- Solaris Zones (2005)
- Linux OpenVZ (2005)
- Linux LXC (2008)
- und mehr ...

Docker V0.0.1 release: 2013

- 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!)

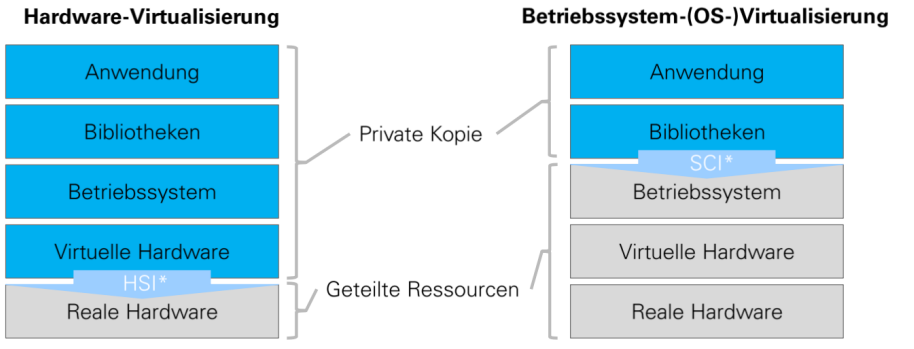
TECHNISCHE HOCHSCHULE LÜBECK

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



HARDWARE- VS. OS-VIRTUALISIERUNG

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



- Stärkere Isolation
- Höhere Sicherheit

- Geringeres Volumen der privaten Kopie
- Geringerer Overhead
- Kürzere Startup-Zeit

HSI = Hardware Software Interface

SCI = System Call Interface

TECHNISCHE HOCHSCHULE LÜBECK

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

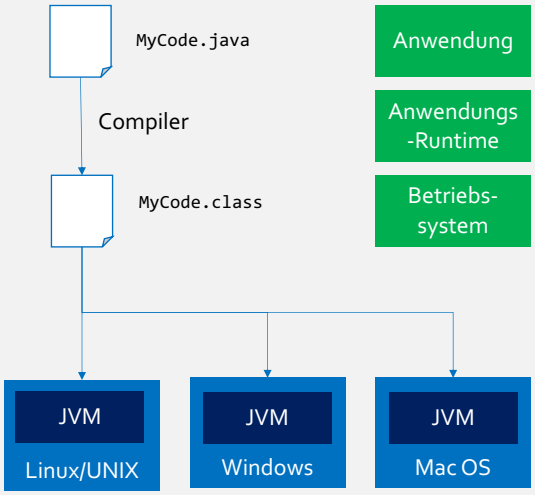
EMULATION VS ANWENDUNGS-VIRTUALISIERUNG

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 Programmier-sprachen genutzt.

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

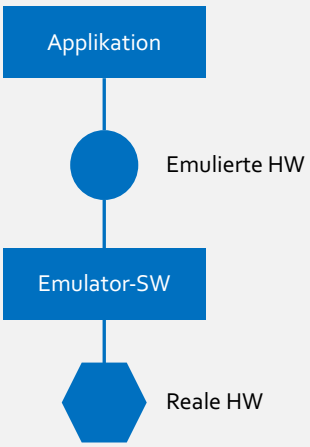
EMULATION VS ANWENDUNGS-VIRTUALISIERUNG

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.

INHALTE

TECHNISCHE HOCHSCHULE LÜBECK

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 Typvertreters 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

TECHNISCHE HOCHSCHULE LÜBECK

Definition und Eigenschaften

Unter IaaS versteht man ein Geschäftsmodell, das entgegen dem klassischen Kaufen 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 verbrauchte Ressourcen

Ressourcen-Typen

- **Rechenleistung:** Rechen-knoten 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:** Schneller Aufgabe, kurze Kündigungsfrist (Reduktion der Bindungsdauer)
- **Echtzeit-Elastizität:** Allokation und Freigabe von Ressourcen innerhalb von 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.

Angebotselastizität:
Die allokierten Ressourcen steigen/sinken mit dem Angebot Nachfrage

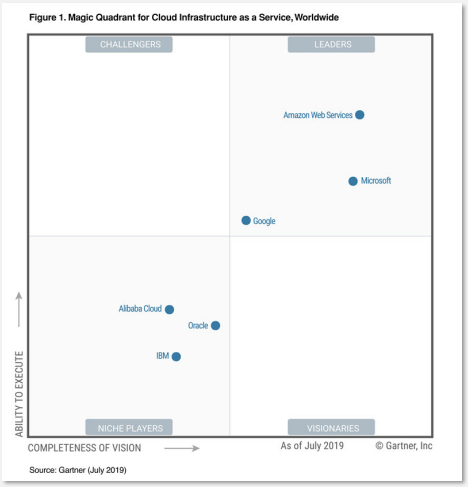
- Dies ist das typische Verhalten eines batch-getriebenen Grids: Alle verfügbaren Rechner werden allokiert.
- Es sind auch Varianten verfügbar, bei denen man für freie Ressourcen bieten kann.

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

TECHNISCHE HOCHSCHULE LÜBECK

INFRASTRUCTURE AS A SERVICE

Der momentane IaaS Markt



TECHNISCHE HOCHSCHULE LÜBECK

INFRASTRUCTURE AS A SERVICE

Auswahlkriterien

- Unterstütze Cloud Variante (Privat Cloud, Public Cloud, Hybrid Cloud, Community Cloud)
- Zuverlässigkeit / Verfügbarkeit
- Sicherheit und Datenschutz
- Gesetzliche Konformität (DGSVO!)
- 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
- Support

TECHNISCHE HOCHSCHULE LÜBECK

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

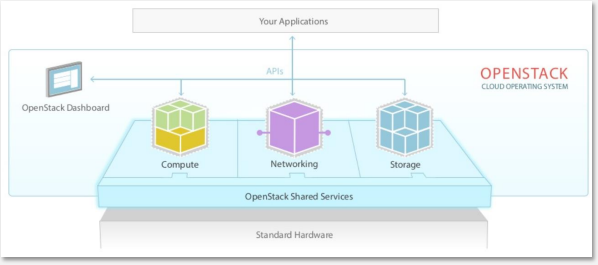
AWS will use commercially reasonable efforts to make Amazon S3 available with a Monthly Uptime Percentage (defined below) of at least 99.9% during any monthly billing cycle (the "Service Commitment"). In the event Amazon S3 does not meet the Service Commitment, you will be eligible to receive a Service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

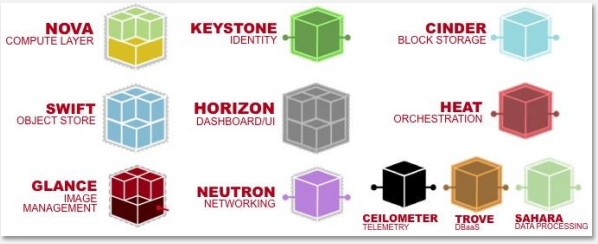
Monthly Uptime Percentage	Service Credit Percentage
Equal to or greater than 99% but less than 99.9%	10%
less than 99%	25%

TECHNISCHE HOCHSCHULE LÜBECK

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.

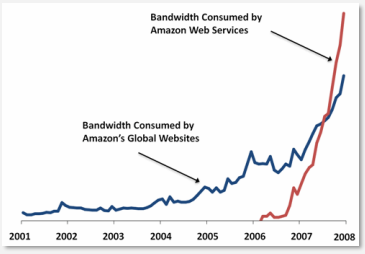
Extrem hohe Installations- und Konfigurationskomplexität.

TECHNISCHE HOCHSCHULE LÜBECK

AMAZON EC2 (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 Websiten (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

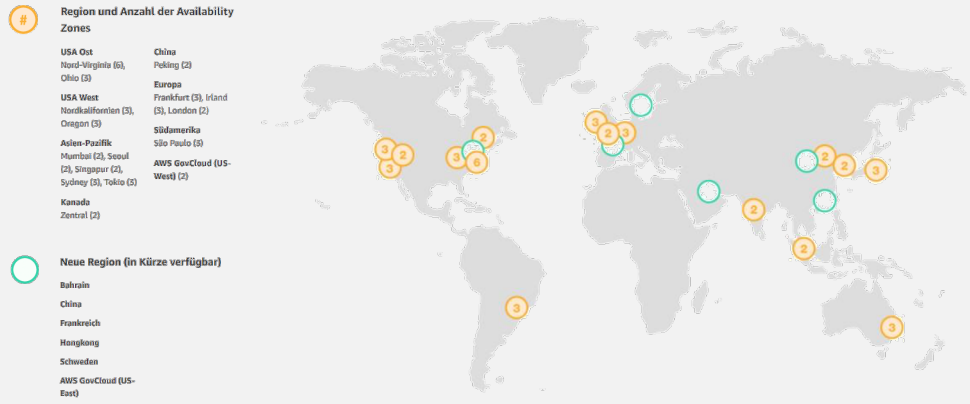


Wir nehmen AWS als Typ-Repräsentant, um zu zeigen, wie alle Public Cloud Provider üblicherweise ihr Geschäft zum Kunden hin organisieren.

TECHNISCHE HOCHSCHULE LÜBECK

AMAZON EC2

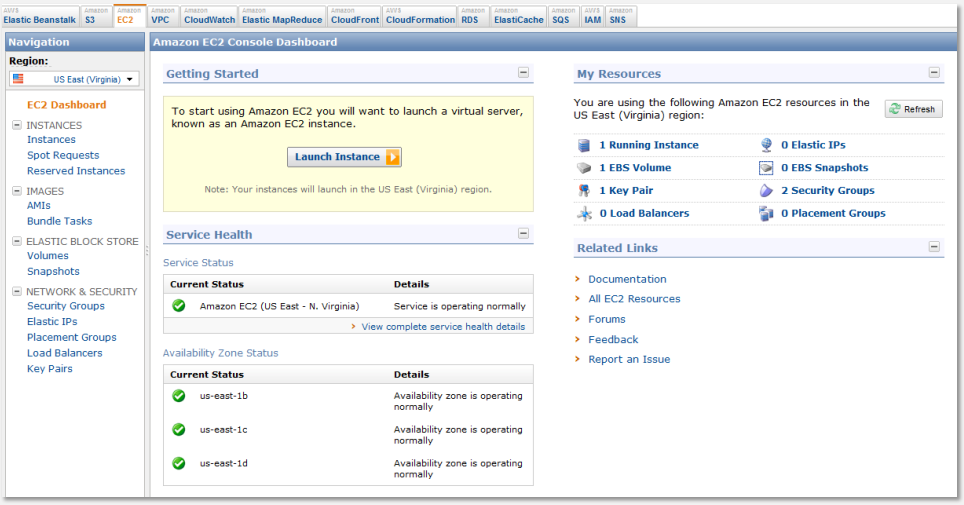
Globale Verteilung



Darstellung kann veraltet sein.

AMAZON EC2

Web-basierte Management Console



Darstellung kann veraltet sein.

AMAZON EC2

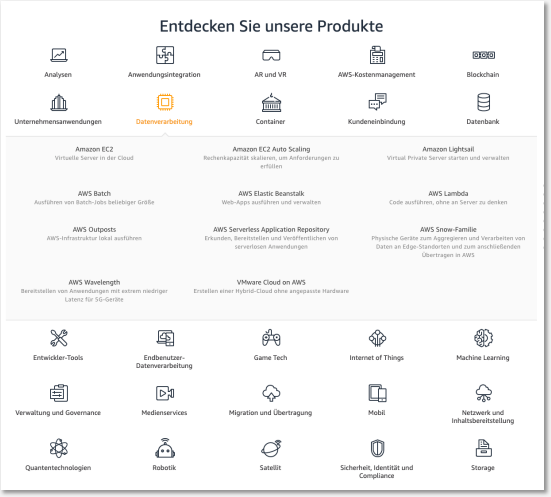
Sicherheitsaspekte

- Zertifiziert nach ISO 27001 (Empfehlung BSI). In europäischen Datencentern den EU-Datenschutzrichtlinien unterworfen. Amazon ist ebenso global dem US Patriot Act unterworfen.
- Daten und Instanzen können global auf alle Rechnenzentren 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 Zertifkaten).
- Zugriff auf Linux Instanzen erfolgt per default mittels Schlüssel-basiertem SSH (kein Passwort).

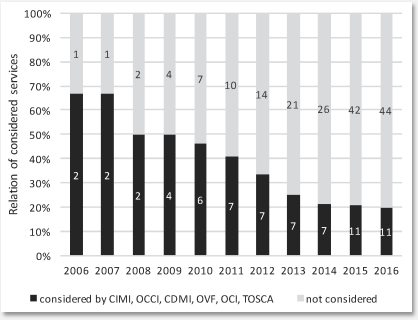
TECHNISCHE HOCHSCHULE LÜBECK

AMAZON WEB SERVICES

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



Quelle: AWS, Darstellung kann veraltet sein.



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.

TECHNISCHE HOCHSCHULE LÜBECK

INHALTE

TECHNISCHE HOCHSCHULE LÜBECK

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 Typvertreters 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

TECHNISCHE HOCHSCHULE LÜBECK

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

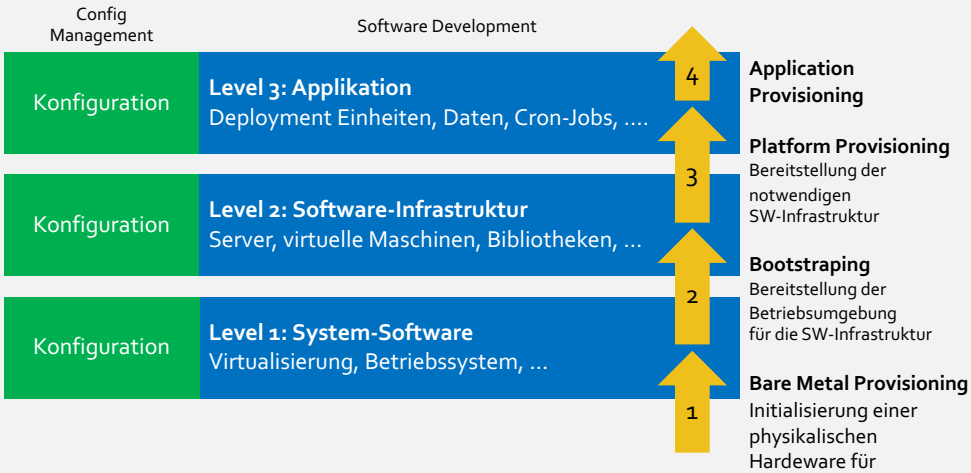
Virtualisierung einzelner Maschinen (2000 – heute)
- Manuelles Installieren von virtuellen Maschinen
- Manuelle Installation von Infrastruktur-Software
- Manuelle / teilautomatisierte / automatisierte Installation der Anwendungssoftware per Installer, Skript, proprietäre Lösungen

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



Application Provisioning

Platform Provisioning
Bereitstellung der notwendigen SW-Infrastruktur

Bootstraping
Bereitstellung der Betriebsumgebung für die SW-Infrastruktur

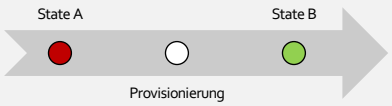
Bare Metal Provisioning
Initialisierung einer physikalischen Hardeware für den Betrieb

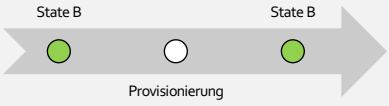
TECHNISCHE HOCHSCHULE LÜBECK

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.

TECHNISCHE HOCHSCHULE LÜBECK

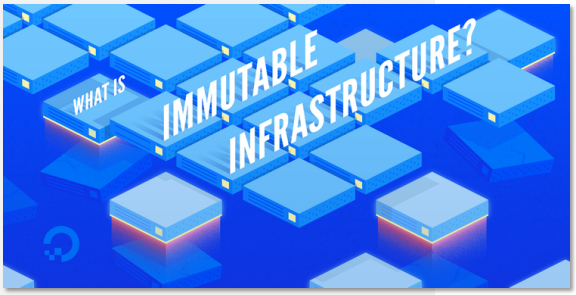
IMMUTABLE ARCHITECTURES

Pets vs. Cattle

TECHNISCHE HOCHSCHULE LÜBECK

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!

https://www.digitalocean.com/community/tutorials/what-is-immutable-infrastructure

IMMUTABLE ARCHITECTURES

Reduzieren zu berücksichtigende Zustände im Provisionierungsprozess

TECHNISCHE HOCHSCHULE LÜBECK

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

TECHNISCHE HOCHSCHULE LÜBECK

Imperativ
Shell Scripting

Shell Abstraktion

Deskriptiv
Zustandsautomaten

Bash PowerShell	Ruby Perl	Capistrano Dockerfile Vagrant	CFEngine	Chef Saltstack Ansible Otter	Puppet Terraform

INHALTE

TECHNISCHE HOCHSCHULE LÜBECK

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 Typvertreters 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)

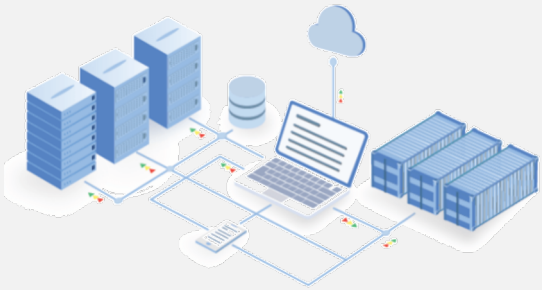
INFRASTRUCTURE AS CODE

Definition

TECHNISCHE HOCHSCHULE LÜBECK

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

TECHNISCHE HOCHSCHULE LÜBECK

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 injeziert der Provisionierungsmechanismus die anzuwendene Konfiguration auf die zu provisionierende Maschine.

INFRASTRUCTURE AS CODE

Tool-Überblick

TECHNISCHE HOCHSCHULE LÜBECK

Tool	Released by	Method	Approach	Written in	Comments
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby	-
Otter	Inedo	Push	Declarative and imperative	-	Windows oriented
Puppet	Puppet (2005)	Pull	Declarative	C++ & Clojure since 4.0, Ruby	-
SaltStack	SaltStack	Push and Pull	Declarative and imperative	Python	-
CFEngine	Northern.tech	Pull	Declarative	C	-
Terraform	HashiCorp (2014)	Push	Declarative	Go	-
Ansible	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

TECHNISCHE HOCHSCHULE LÜBECK

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

Download:
`https://www.virtualbox.org`

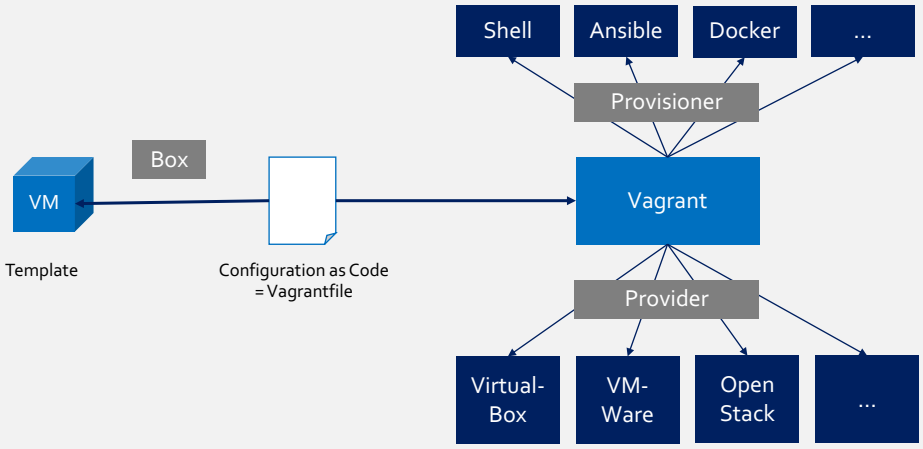
Shell-basierte Automationssoftware für virtuelle Umgebungen auf einem Rechner

Download:
`https://www.vagrantup.com/downloads`

Diese Kombination eignet sich gut für lokale Entwicklungs-maschinen.

VAGRANT

Konzepte



TECHNISCHE HOCHSCHULE LÜBECK

VAGRANTFILE

Beschreibung einer zu erstellenden Virtuellen Maschine

TECHNISCHE HOCHSCHULE LÜBECK

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

Weitere Kommandos: https://www.vagrantup.com/docs/cli

TECHNISCHE HOCHSCHULE LÜBECK

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 ...

GitLab

TECHNISCHE HOCHSCHULE LÜBECK

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

```
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).

TECHNISCHE HOCHSCHULE LÜBECK

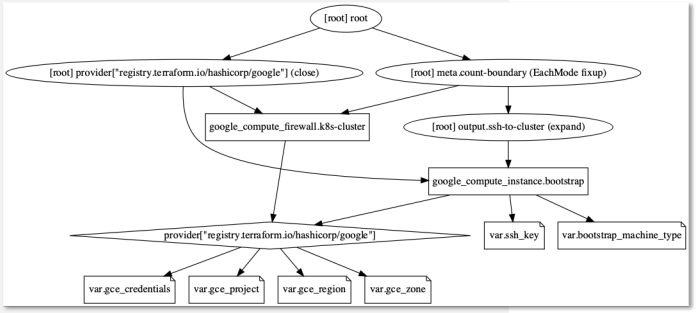
TERRAFORM

Deklarative Programmierung von Infrastrukturen

Write:
Beschreibung eines Zielzustands über eine domänenspezifische Sprache HCL (HashiCorp Configuration Langugage)

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

TECHNISCHE HOCHSCHULE LÜBECK

TERRAFORM

Die Kern-Entitäten in Terraform Skripten

TECHNISCHE HOCHSCHULE LÜBECK

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

TECHNISCHE HOCHSCHULE LÜBECK

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

```
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
```

Basisdefinitionen, die für alle Environments gelten

**.tf: Terraform Deklarationen*

QA (Test/Staging) Environment

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

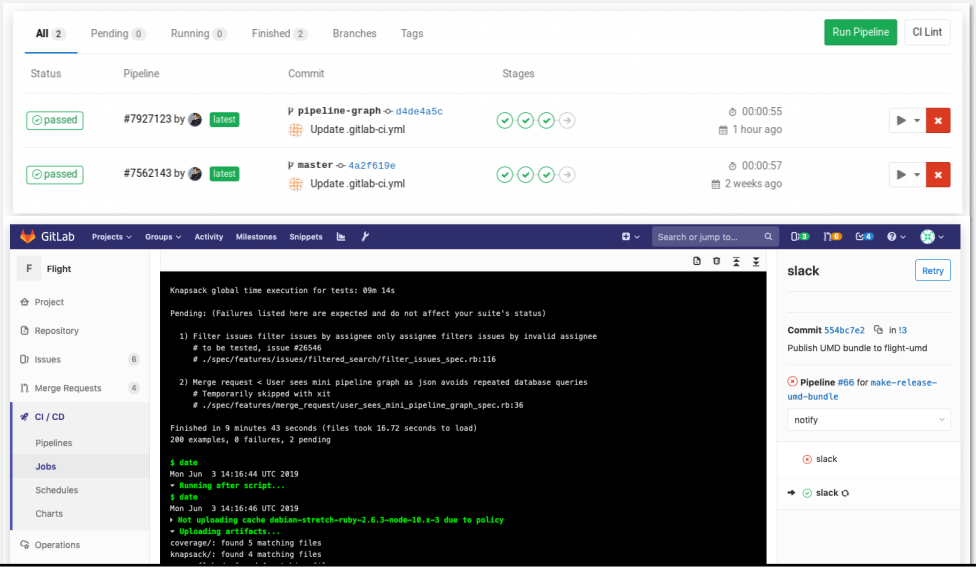
**.tfvars: Variablenwerte (ggf. Environment-spezifisch)*

TECHNISCHE HOCHSCHULE LÜBECK

PIPELINES

In Gitlab

GitLab



Pipelines werden bei jedem Push automatisch ausgeführt.

Für jeden Job gibt es auch Logs (für Fehlerdiagnosen).

Mehr dazu im Praktikum

TECHNISCHE HOCHSCHULE LÜBECK

TERRAFORM

Nur Versuch macht kluch …

TECHNISCHE HOCHSCHULE LÜBECK

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

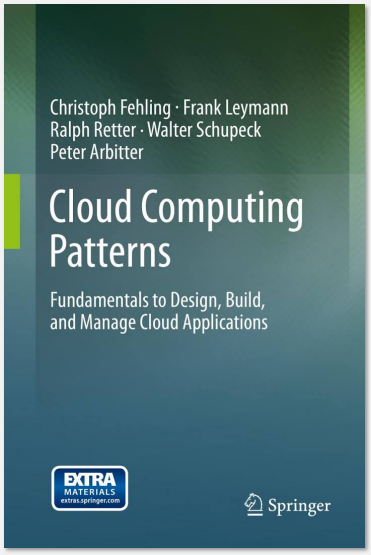
```
small_workers=2
xl_workers=2
```

um Ihren Cluster von einem Knoten auf fünf Knoten hochzuskalieren.

```
> terraform apply
```

Provisioning und Skalierung von Kubernetes Cluster in GCE.

ZUM NACHLESEN



TECHNISCHE HOCHSCHULE LÜBECK

2.3 **Cloud Service Models**
2.3.1 Infrastructure as a Service

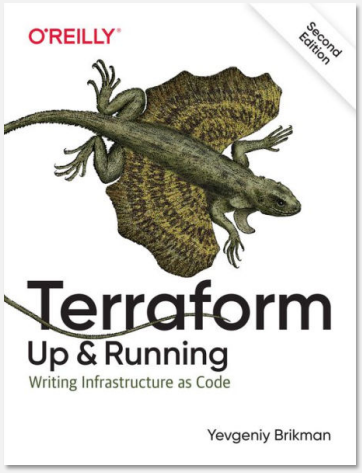
3.3. **Cloud Environments**
3.3.1 Elastic Infrastructure

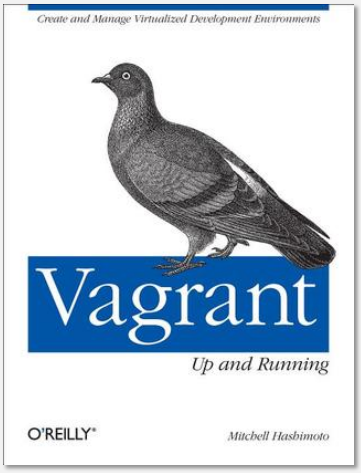
3.4 **Processing Offerings**
3.4.1 Hypervisor

3.5 **Storage Offerings**
3.5.1 Block Storage
3.5.2 Blob Storage

3.6 **Communication Offerings**
3.6.1 Virtual Networking

ZUM NACHLESEN





TECHNISCHE HOCHSCHULE LÜBECK

KONTAKT

Disclaimer

Nane Kratzke
+49 451 300-5549
nane.kratzke@th-luebeck.de
kratzke.mylab.th-luebeck.de