TECHNISCHE UNIVERSITÄT DRESDEN

Vorlesung „Service and Cloud Computing"

8. Cloud Computing - Erweiterte Konzepte

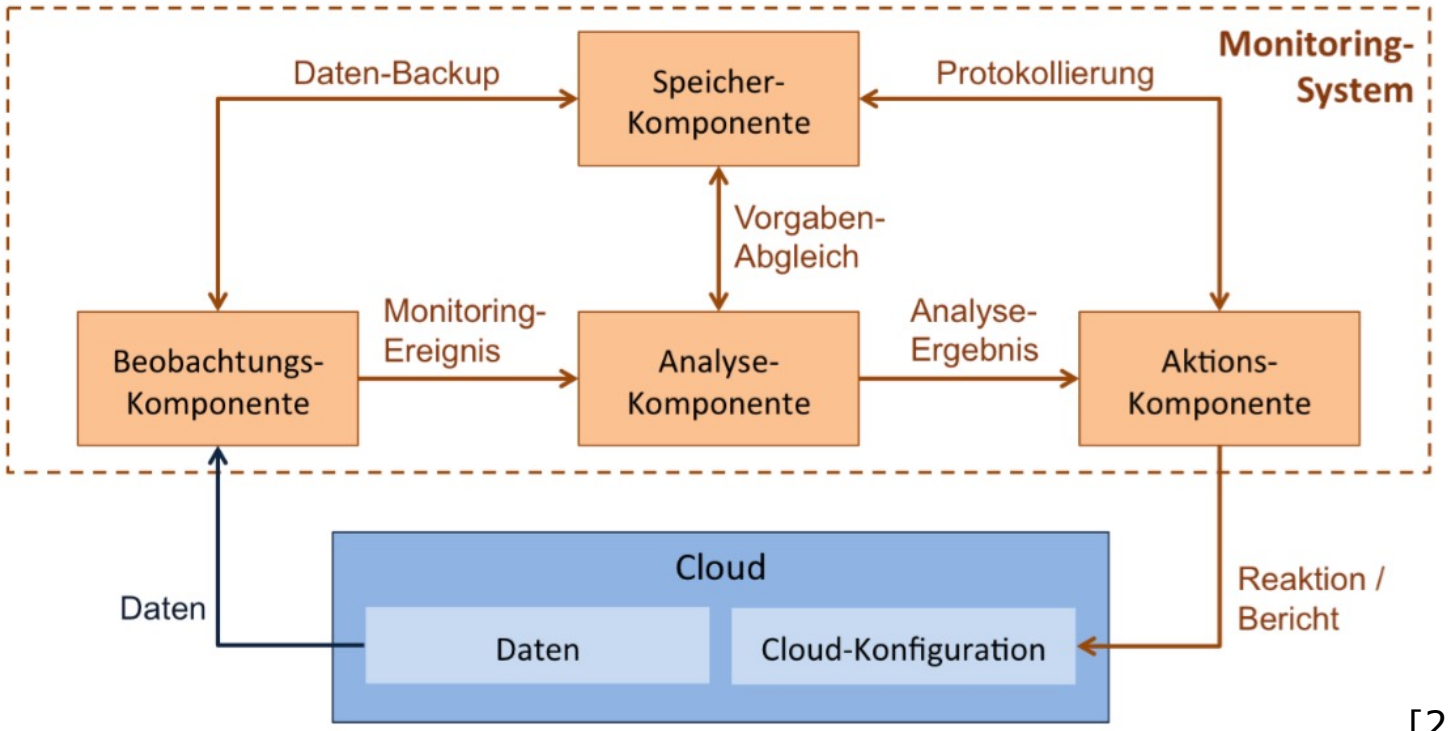
Dr.-Ing. Iris Braun

Cloud Computing – Erweiterte Konzepte

- Monitoring, Überwachungstools der Cloudplattformen
- Service Level Agreements
- Load Balancing
- Autoscaling

- Multicloudansteuerungstools – Terraform, Cloudify, Sparkleformation
- Konfigurationsmanagement + Provisioning – Ansible, Chef, Puppet, etc.
- DevOps, Container, Docker

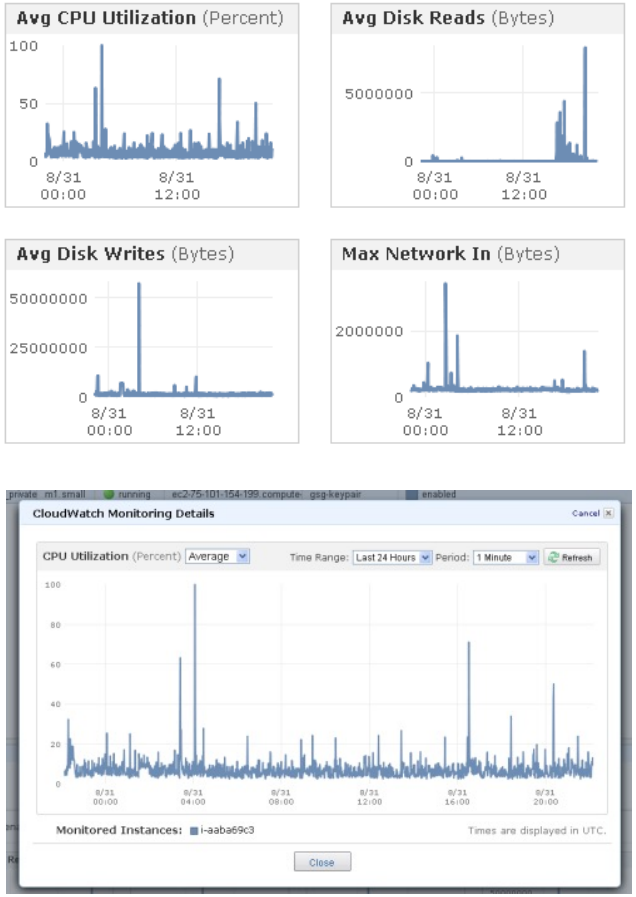
- **„Monitor"** beschreibt ein System,
  „welches das Verhalten eines anderen Systems beobachtet und entscheidet, ob dieses Verhalten mit den gegebenen Vorgaben übereinstimmt" [1]



[2,3]

AWS CloudWatch

- überwacht automatisch die Elastic Load Balancer für Metriken wie die eingehenden Requests und Latenz
- Überwachung von Schreib-/ Leselatenzen, Datenbankinstanzen und verfügbaren Speicherplatz, Warteschlangen
- Definition von Alarmen, welche Benachrichtigungen auslösen oder automatische Aktionen durchführen, wenn die Metrik einen bestimmten Schwellenwert überschritten hat
- benutzerdefinierte System-, Anwendungs- und Protokolldateien in Amazon CloudWatch Logs

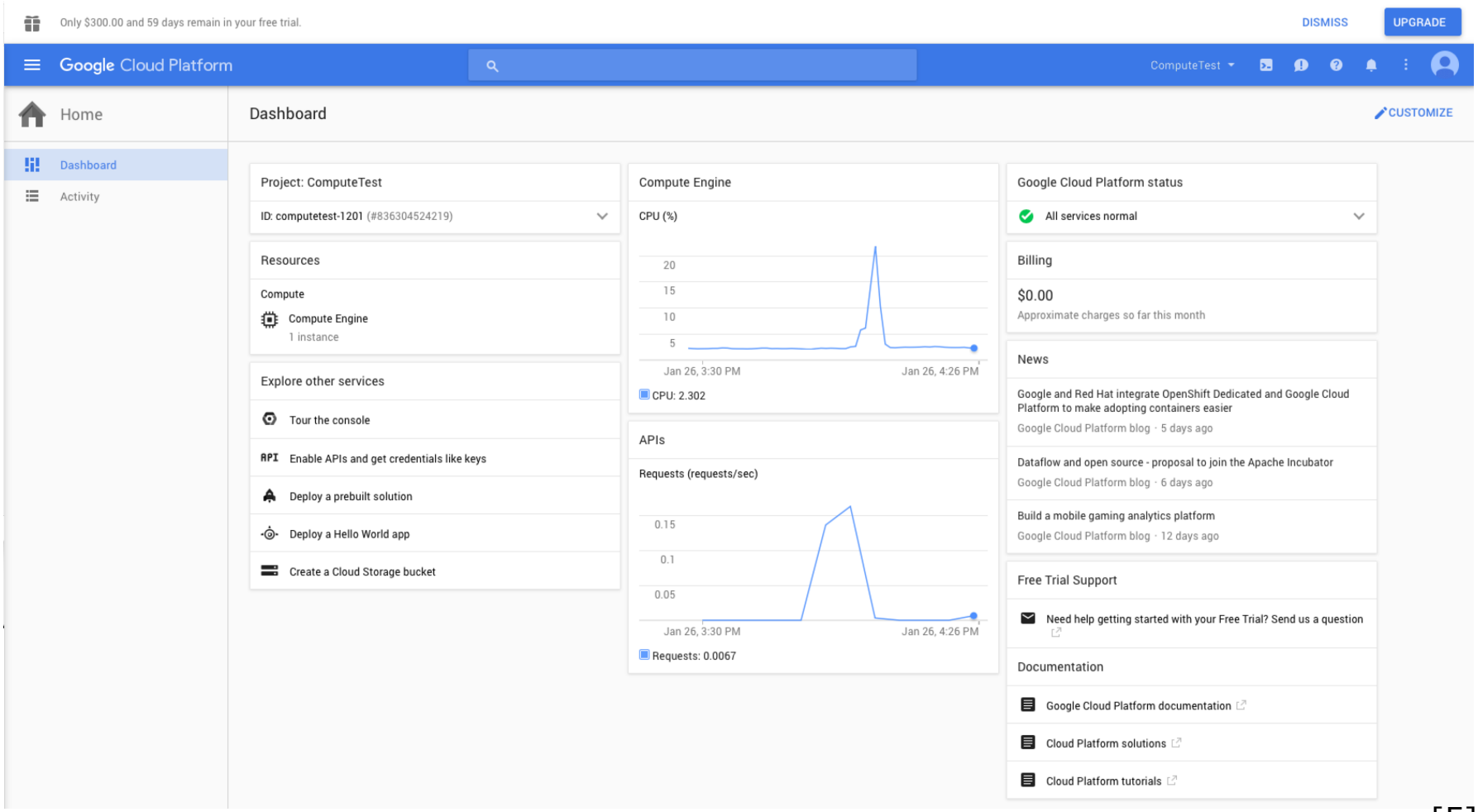


[4]

Google Stackdriver Monitoring

- Dashboard - umfangreiches Interface mit Diagrammen zur Performance, Auslastung, Gesundheit etc. der Anwendungen
- benutzerdefinierte Metriken auf Business- und Applikationsebene
- auch kompatibel mit AWS-API
- kompatibel mit bekannten Software-Lösungen wie Cassandra, Nginx, Apache Web Server, Elasticsearch
- Push-Benachrichtigungen und Alarme via Slack, PagerButy, HipChat, Campfire

Google Dashboard



[5]

Klassische Quality of Service Parameter (QoS)

- Verfügbarkeit, Service-Zeit
- Ausfallsicherheit
- Netzwerk-Anbindung/Bandbreite
- Antwortzeit (Response Time) einzelner Services (SaaS)
- Support-Zeiten

- Definition von Strafzahlungen/ Rabatten bei Nichteinhaltung
- Monitoring durch Provider
- Meist keine formale Definition, sondern textuell
- Keine Standardisierung

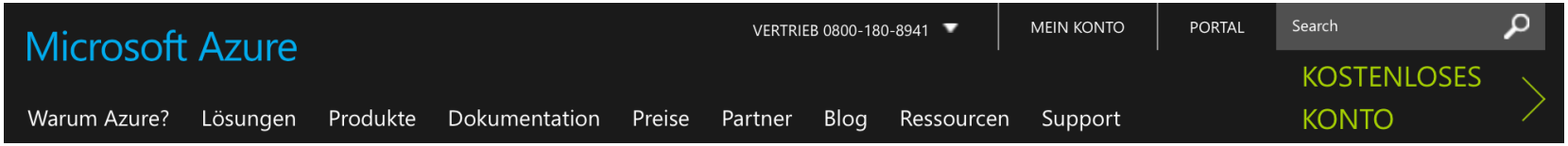
Service Level Auswahl

Definition	Service Level		
Service-Betrieb	7 x 24		
Bediente Service-Zeit	Mo-Fr 8.00 - 17.00 MEZ		
	Allround	Advanced	Superior
Verfügbarkeit der Infrastruktur im Rechenzentrum von Fujitsu, Mes Verfügbarkeit ist die Internet-Sch Rechenzentrum von Fujitsu (auß Wartungszeiten)			99.5 %
Einrichten neuer Kunden			10 Arbeitstage
Bereitstellung virtueller Server (~30Min.)	5 Arbeitstage	3 Arbeitstage	3 Arbeitstage
Bereitstellung dedizierter Server	10 Arbeitstage	5 Arbeitstage	5 Arbeitstage

SLA: Viele Business-orientierte Cloud Dienstleister - im Bild ein Beispiel von Fujitsu - verwenden abgestufte Service Levels, so dass unterschiedliche Anforderungen bedient werden.

Optionale Services:
- ✓ Administration Support
- ✓ Erstellen VMDK (VM Disk Format)
- ✓ Active Directory Integration
- ✓ System zurücksetzen (Reset)
- ✓ Wiederherstellung von Daten aus Snapshot

SLA am Beispiel von Microsoft Azure



Berechnung der Monatlichen Betriebszeit und Servicelevel für Virtuelle Computer

„**Maximal Verfügbare Minuten**" ist die Gesamtzahl der Minuten während eines Monats der Rechnungsstellung für alle internetseitigen Virtuellen Computer, für die zwei oder mehr Instanzen in der gleichen Verfügbarkeitsgruppe bereitgestellt sind. Die Maximal Verfügbaren Minuten werden ab dem Zeitpunkt gemessen, zu dem mindestens zwei Virtuelle Computer in der gleichen Verfügbarkeitsgruppe als Ergebnis einer vom Kunden initiierten Aktion gestartet sind, und bis zu dem Zeitpunkt, an dem der Kunde eine Aktion eingeleitet hat, die zum Stoppen oder Löschen der Virtuellen Computer führt.

„**Ausfallzeit**" ist die Gesamtzahl der Minuten unter den Maximal Verfügbaren Minuten, für die keine externe Verbindung besteht.

Der „**Prozentsatz der Monatlichen Betriebszeit**" für Virtuelle Computer wird für ein bestimmtes Microsoft Azure-Abonnement berechnet als die Maximal Verfügbaren Minuten minus Ausfallzeit geteilt durch die Maximal Verfügbaren Minuten im Monat der Rechnungsstellung. Der „Prozentsatz der monatlichen Betriebszeit" wird mithilfe der folgenden Formel berechnet:

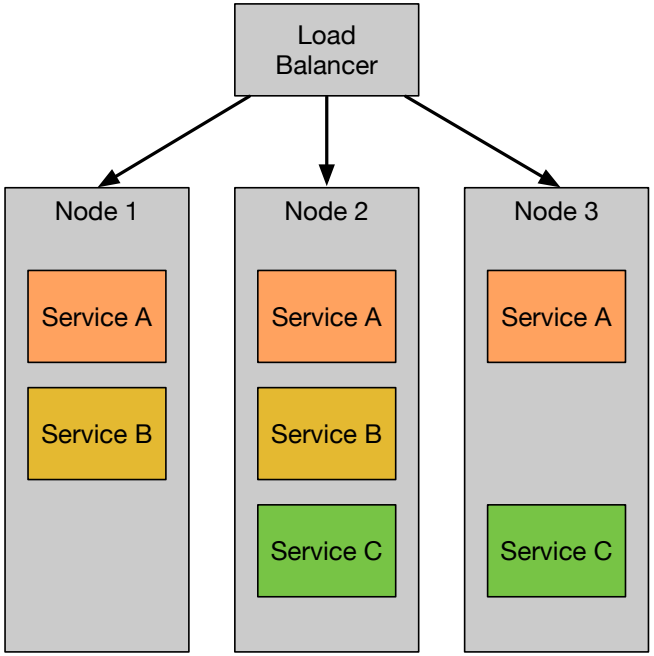
Prozentsatz der Monatlichen Betriebszeit = (Maximal verfügbare Minuten – Ausfallzeit) / Maximal verfügbare Minuten

Die folgenden Servicelevel und Dienstgutschriften gelten für die Nutzung von Virtuellen Computern durch den Kunden:

PROZENTSATZ DER MONATLICHEN BETRIEBSZEIT	DIENSTGUTSCHRIFT
< 99.95%	10%
< 99%	25%

Load Balancing

- Bedarfsorientierte Verteilung der Last auf mehrere Rechnerknoten
- Erhöhung der Ausfallsicherheit bei Infrastrukturproblemen
- Verteilung der eingehenden Requests auf mehrere virtuelle Maschinen, die redundante Service Deployments ausführen



2 Arten von Load Balancern:

- **Packet-based**
  - Für jedes ankommende Paket wird ein Zielhost ausgewählt
- **Flow-based**
  - Pro Verbindung wird ein Ziel für die Datenübertragung selektiert, das während der Verbindung konstant bleibt

Load Balancing Algorithmen

- **Round Robin:** verwaltet alle Server in einer Liste und arbeitet nacheinander die Liste der Requests und der Server ab (FIFO), wurde der letzte Eintrag in der Liste erreicht, fängt er wieder von vorn an.
- **Gewichtetes Round Robin:** Wie Round-Robin, jedoch bekommen durch die Gewichtung gewisse Requests mehr Ressourcen zugeteilt.
- **Random:** Zufällige Zuteilung des Traffics.
- **Geringste Verbindungen:** Der LB zählt die offenen Verbindungen jedes Servers und sendet weiter an den Server mit der geringsten Last.
- **Geringster Traffic:** Der LB überwacht den Traffic des Servers und leitet die Anfrage an den mit dem geringsten ausgehenden Traffic weiter.
- **Geringste Latenz:** LB baut einen kurzen HTTP OPTIONS Request zum Backend Server auf und sendet die Anfrage zum Ersten weiter, der antwortet.
- **Source IP Hash:** Verbindungen sind basierend auf ihrer Source-IP auf Backend-Server aufgeteilt. Fällt ein Server aus wird eine neue IP zugeordnet. Anderenfalls werden Anfragen eines zugewiesen Clients immer an den gleichen Webserver weitergeleitet.
- **URL Hash:** Ähnlich wie bei dem vorherigen Verfahren, nur das eine URL immer auf den gleichen Webserver zeigt.

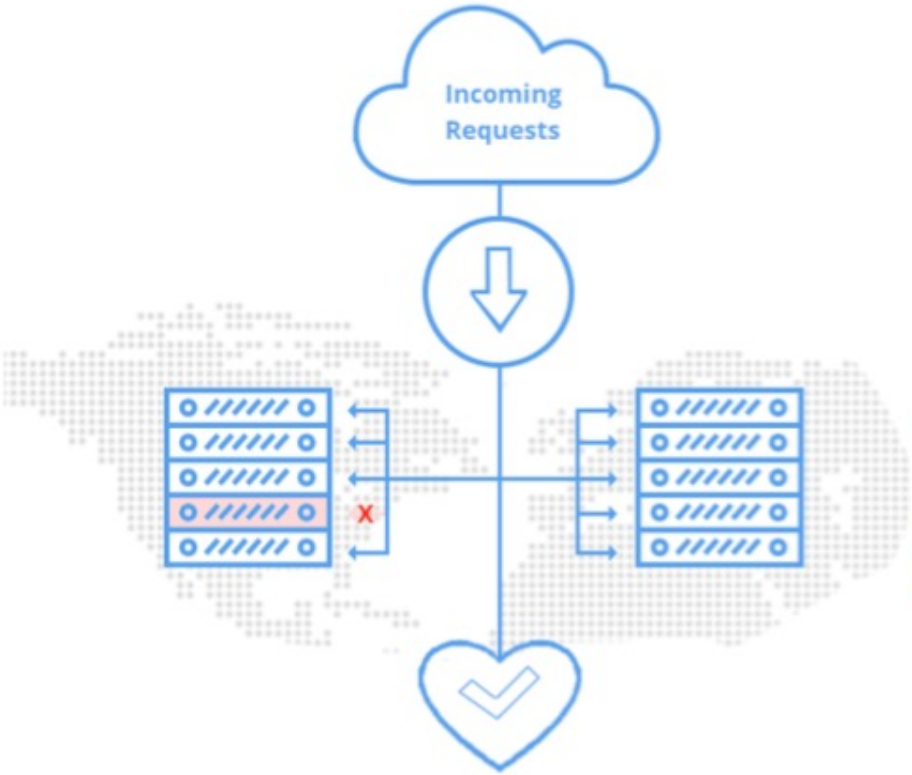
DNS-based Load Balancing

- Domain Name System (DNS) unterstützt mehrere Resource Records für eine einzelne Klasse:

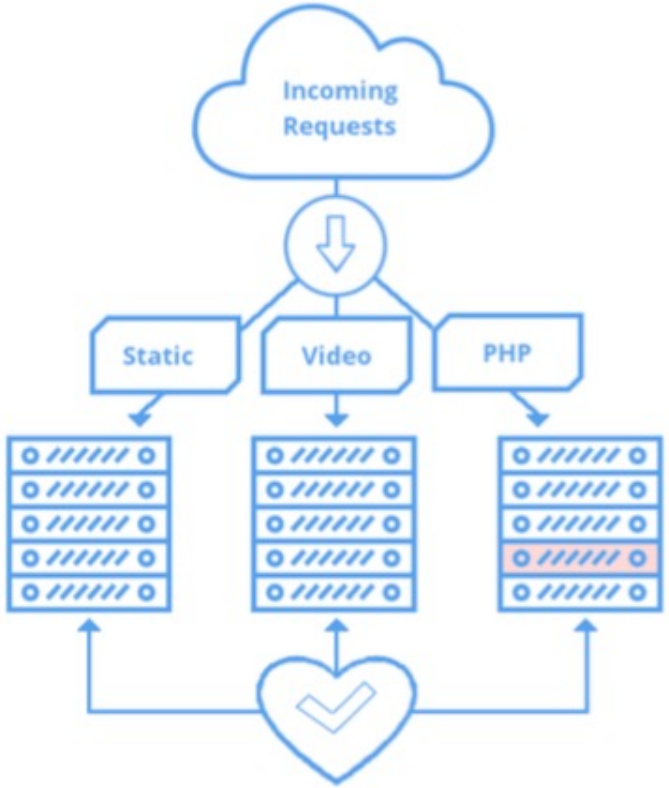
```
www.example.com. 2400 IN A 141.76.40.2
www.example.com. 2400 IN A 141.76.40.8
…
```

- DNS Server antwortet mit der Liste der Ergebnisse in unterschiedlicher Reihenfolge
- Sortierung der Ergebnisliste kann konfiguriert werden:
  - **Fixed:** vorgegebene Sortierung in Zonefile
  - **Random:** zufällig sortierte Liste
  - **Cyclic:** Round-Robin (Default-Einstellung)
- Einschränkungen:
  - Server-Status und -Auslastung werden nicht berücksichtigt
  - Ansatz wird durch Caching ausgehebelt

Weitere Load Balancing Arten



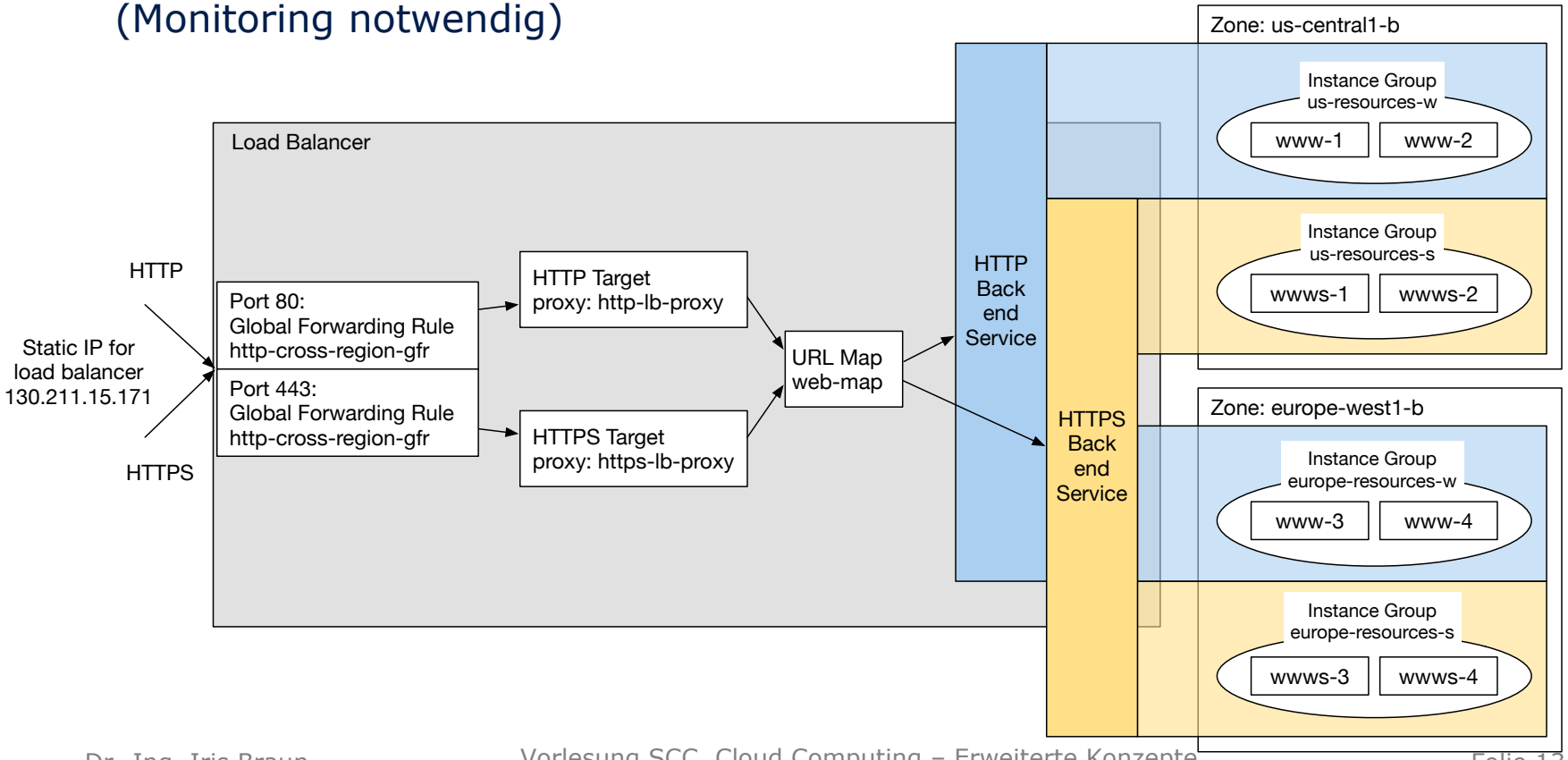
Cross-region LB



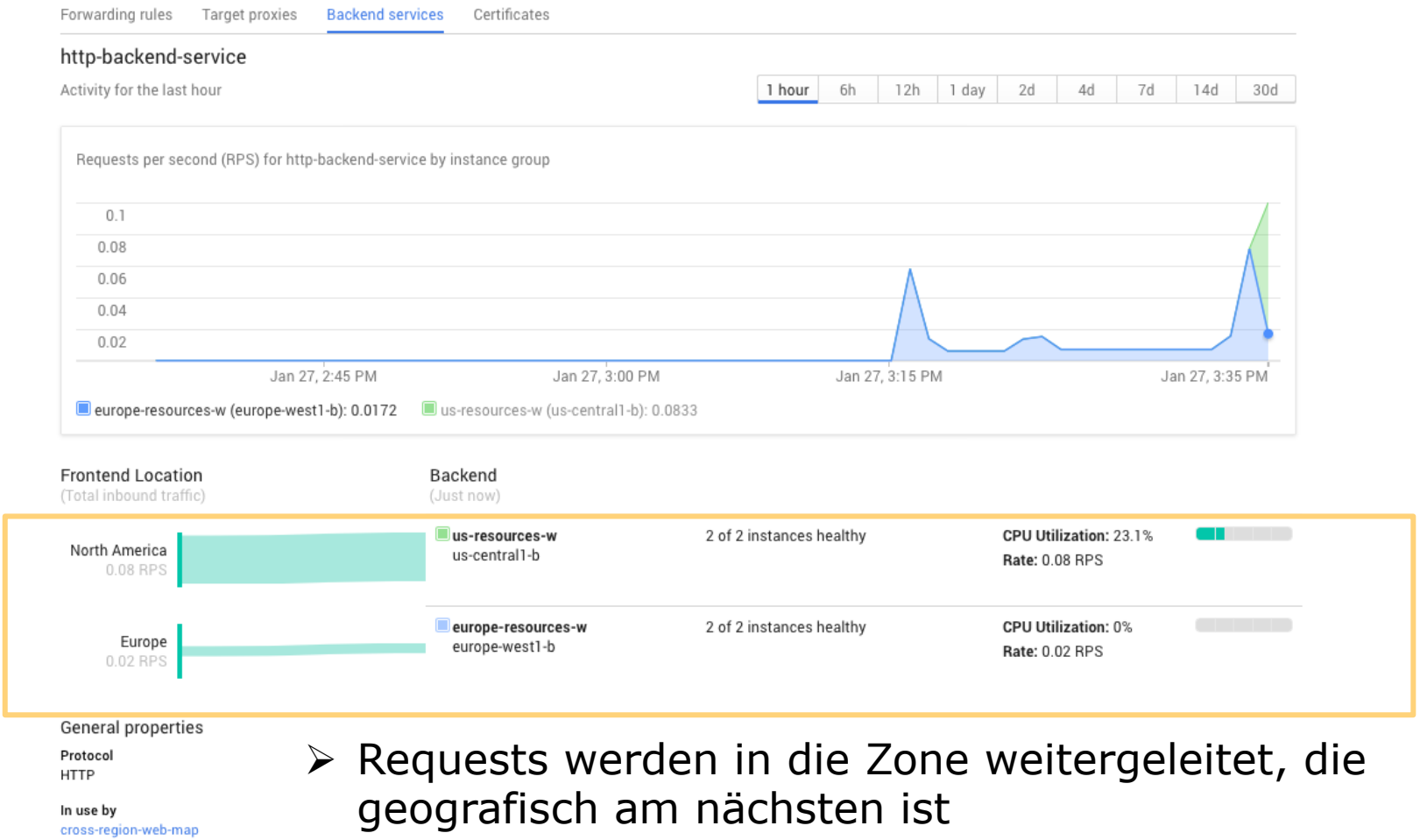
Content-based LB

Cross-Region Load Balancing

- Load Balancer mit statischer IP-Adresse
- Weiterleitung an den Server, der am nächsten beim Anfrager ist
- Load Balancer kann auch die Auslastung der Server berücksichtigen (Monitoring notwendig)



Cross-Region Load Balancing
Google Compute Engine



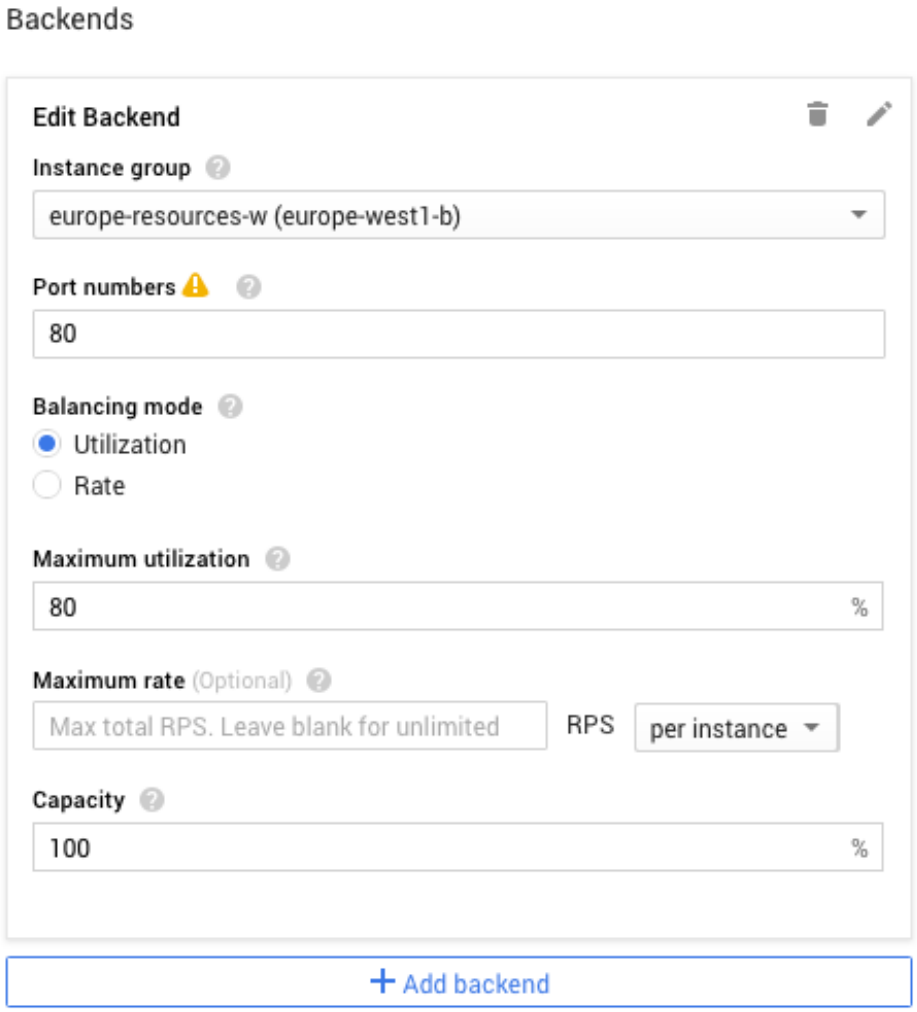
- Requests werden in die Zone weitergeleitet, die geografisch am nächsten ist

- Innerhalb der Zone werden die Requests entsprechend der Auslastung verteilt
- Balancing Mode

  Definiert, ob der Scheduler abhängig von der CPU-Last oder der Anzahl der eingehenden Requests pro Sekunde entscheidet
- Balancing mode = Utilization

  Festlegung einer maximalen CPU-Auslastung
- Balancing mode = Rate

  Maximale Anzahl von Requests pro Mintute



Skalierbarkeit

Stellen Sie sich vor, der von Ihnen entwickelte Service kommt an die Grenzen seiner Leistungsfähigkeit, weil plötzlich Millionen Nutzer darauf zugreifen. Was würden Sie (als erstes) tun, um die Performanz wieder zu verbessern?

- Die Leistungsfähigkeit meines Servers erhöhen durch mehr RAM und bessere Prozessoren.
- Meinen Service auf mehreren Servern/virtuellen Maschinen installieren und diese parallel betreiben.
- Meine Anwendungslogik anpassen oder eine performantere Programmiersprache oder DB nutzen.
- Meine Implementierung in mehrere Ebenen teilen - WebApp, Business-Logik, Persistenz-Schicht und die auf verschiedenen Servern laufen lassen.
- Meine Anwendung in viele kleine Services zerlegen und diese auf verschiedene Server verteilen.

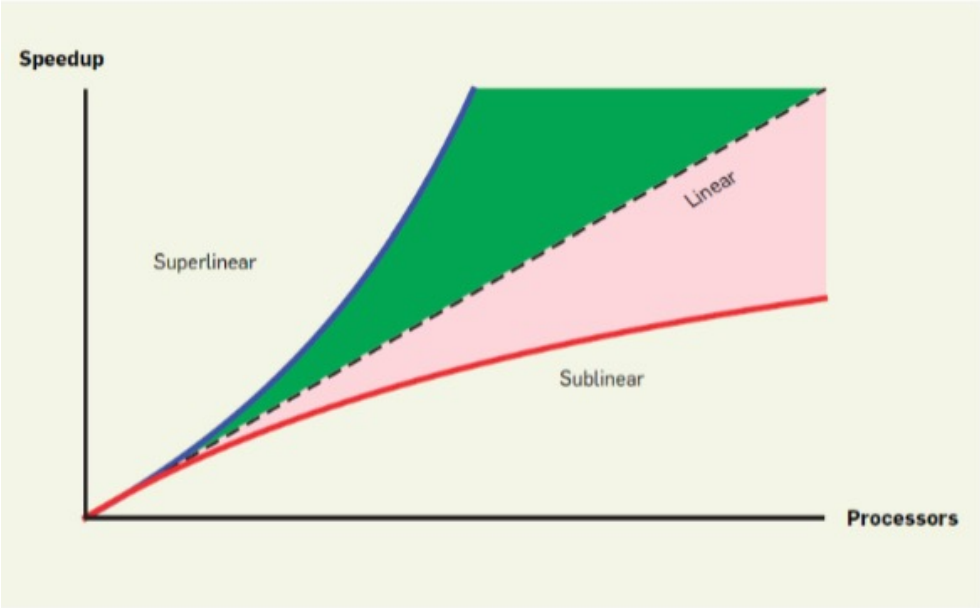
Skalierbarkeit - Definition

Skalierbarkeit ist die Fähigkeit eines Systems, sich wachsenden Ansprüchen an die Leistungsfähigkeit anzupassen.

Der Skalierbarkeitsfaktor (*Speedup*) ist das Verhältnis von zusätzlicher Leistung zu zusätzlich eingesetzten Ressourcen.

Universal Scalability Law
(Neil J. Gunther 1993 [9])

- Superlinear
- Linear
- Sublinear
- Negative



Vertikale Skalierung

- Ressourcen zu vorhandenen Knoten (Hosts) hinzufügen

\+ Einfach umzusetzen

– Limitiert durch die Grenzen der eingesetzten Hardware

– Kostenintensiv

Horizontale Skalierung

- weitere Knoten (Hosts) hinzufügen

\+ Grenzenlos erweiterbar

\+ Kosten entstehen nur bei Bedarf

– Virtualisierung der Hardware-Ressourcen erforderlich

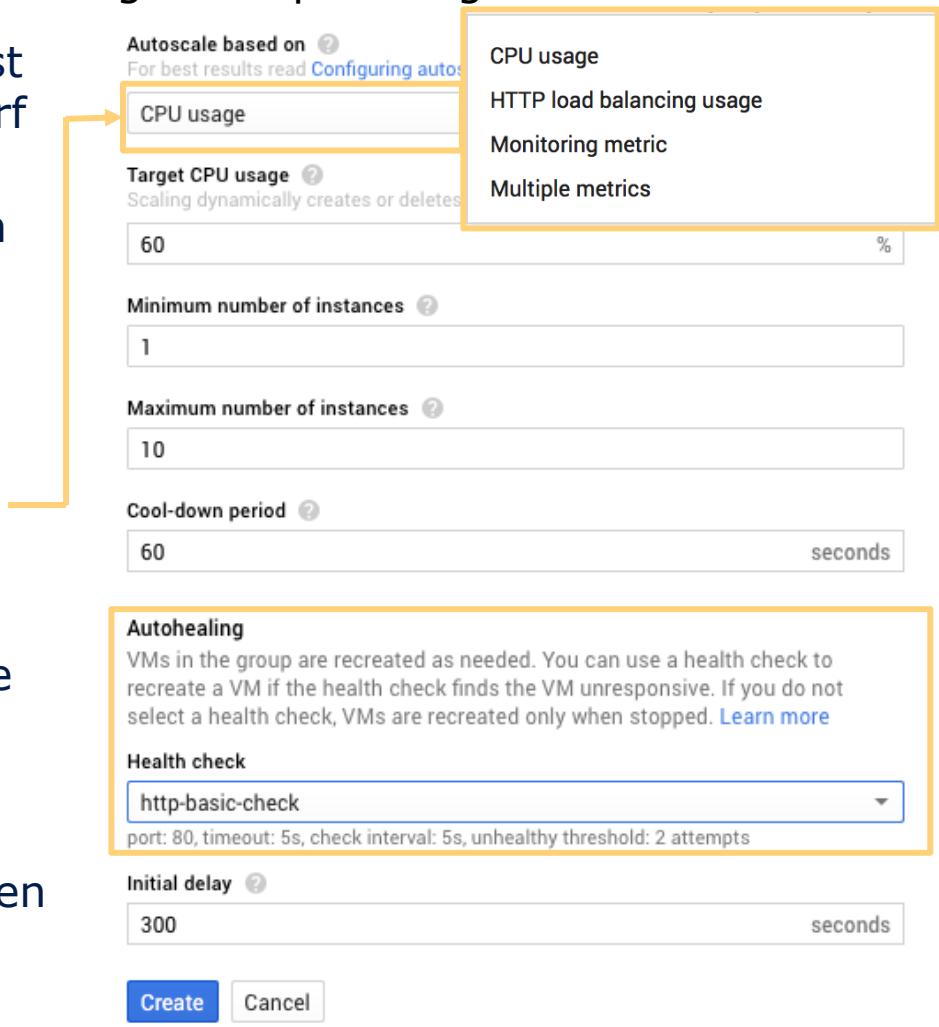
– Replikation der Rechnerknoten und Daten muss unterstützt werden

Autoscaling: automatisch neue Instanzen hinzufügen, wenn die Last steigt oder löschen, wenn der Bedarf nachlässt

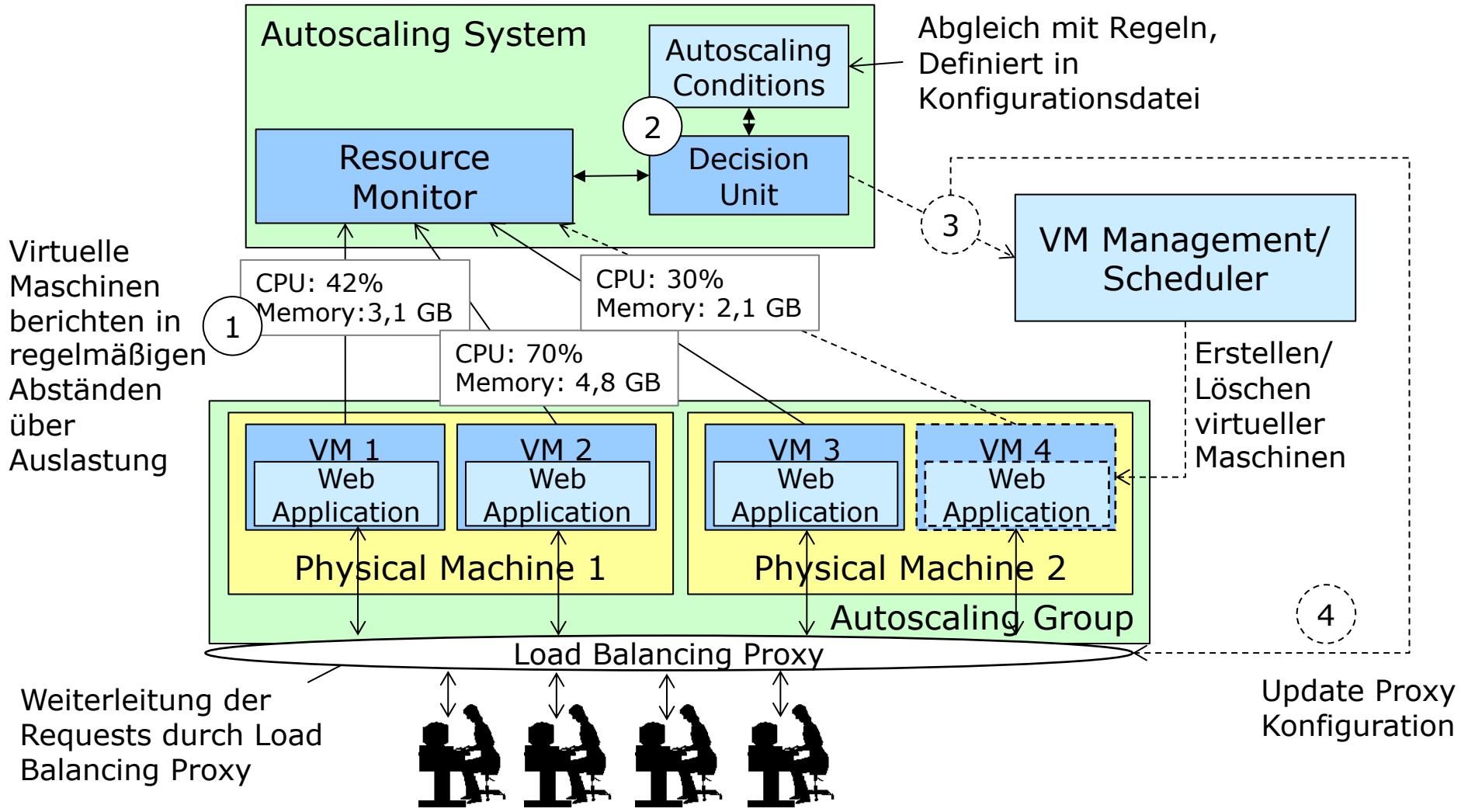
- Erhöhung der Elastizität – Abfangen von Lastspitzen
- Kostenreduktion, wenn Last sinkt (Pay-per-Use)
- Autoscaling Policy definiert die Regeln für das An-/Abschalten
  - Verschiedene Metriken sind konfigurierbar
  - Z.B.: wenn die durchschnittliche CPU Nutzung aller Instanzen >=60%, dann erstelle neue Instanz

Autohealing: automatisches Neustarten von VMs, wenn diese nicht mehr reagieren

Google Compute Engine



Elastic Load Balancing, Autoscaling

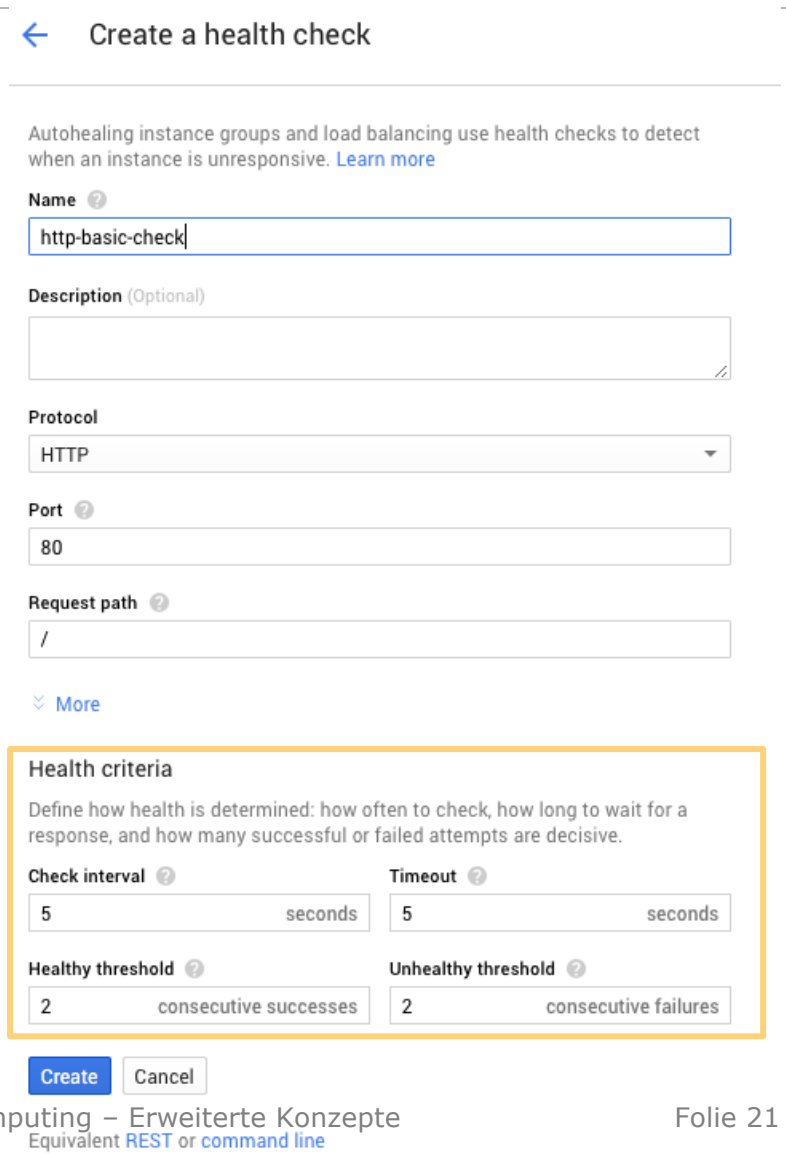


Health Checks

- Periodische Tests, ob die Instanzen noch korrekt arbeiten
- Als Ergebnis Werte HEALTHY oder UNHEALTHY

z. B. *http-basic-check*

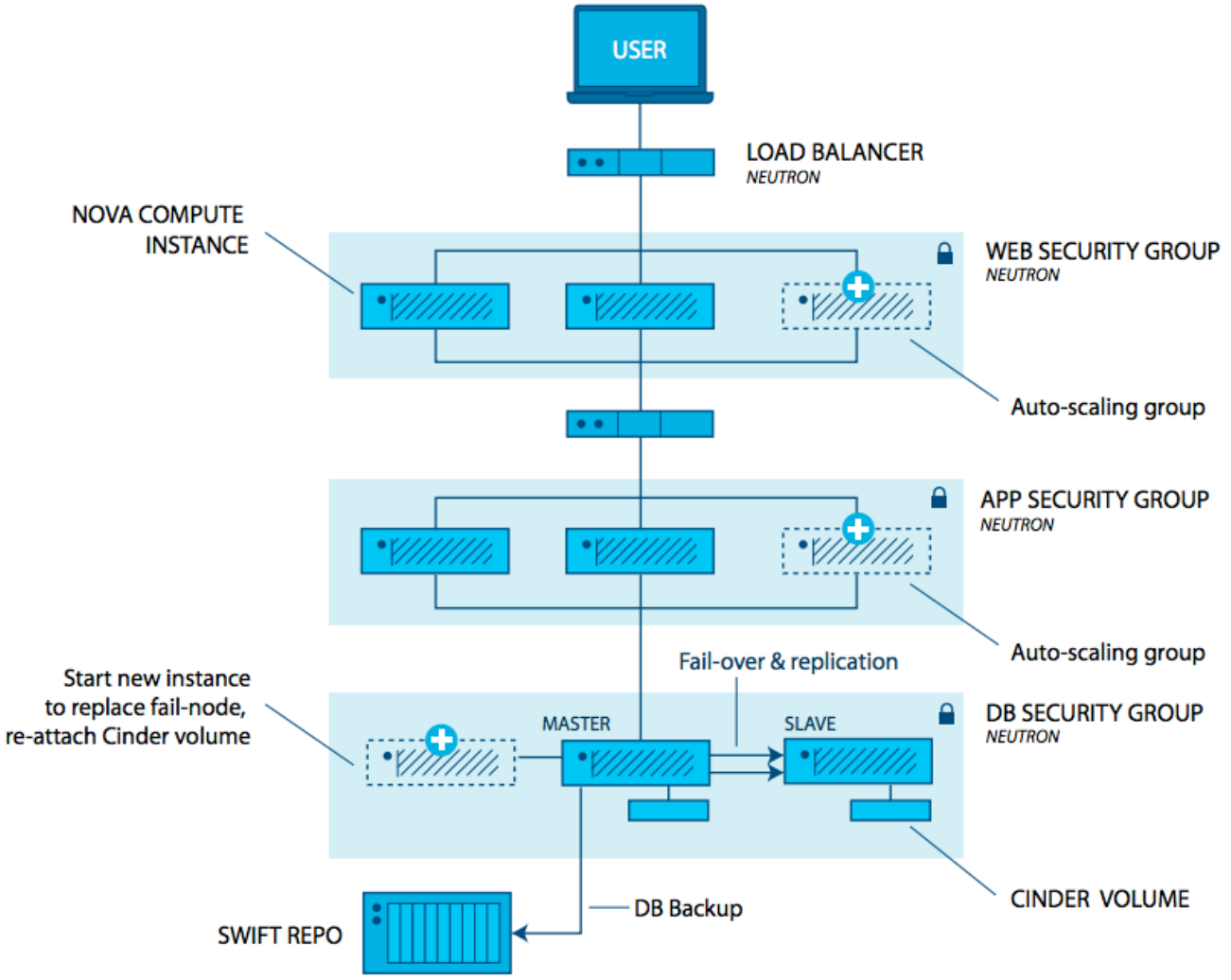
- Sendet periodisch HTTP-Anfragen an Webserver auf virtueller Maschine
- Definition von Timeout, Testintervall und Grenzwert
- Wenn HTTP-Response 200 OK, dann Instanz „gesund"



Web Application

Bussiness Logic

Database/ Persistence Layer



Cloud Konfigurationsmanagement

- **Autoscaling und Load Balancing** setzen eine Automatisierung der Bereitstellung von Cloud-Instanzen voraus
- Automatisierung der Erstellung und Konfiguration von Cloud-Ressourcen (z.B. mit Terraform, sparkleformation, cloudify)
- Multicloud-Ansteuerung, bedienen APIs vieler Cloud-Anbieter (Google, AWS, Azure, OpenStack ...)
- **Konfigurations-Management**
  - Verwaltung von Systemkonfigurationen und Änderungen derselben zur Erhaltung von Integrität und Konsistenz der verwalteten Systeme
  - Management von einzelnen Instanzen und Clustern
  - Werkzeuge, Regelwerke, Prozeduren und verschiedene Techniken zur Verwaltung und Evaluierung von Änderungen, Durchgeführte Änderungen sind über die Zeit nachvollziehbar und zurücksetzbar

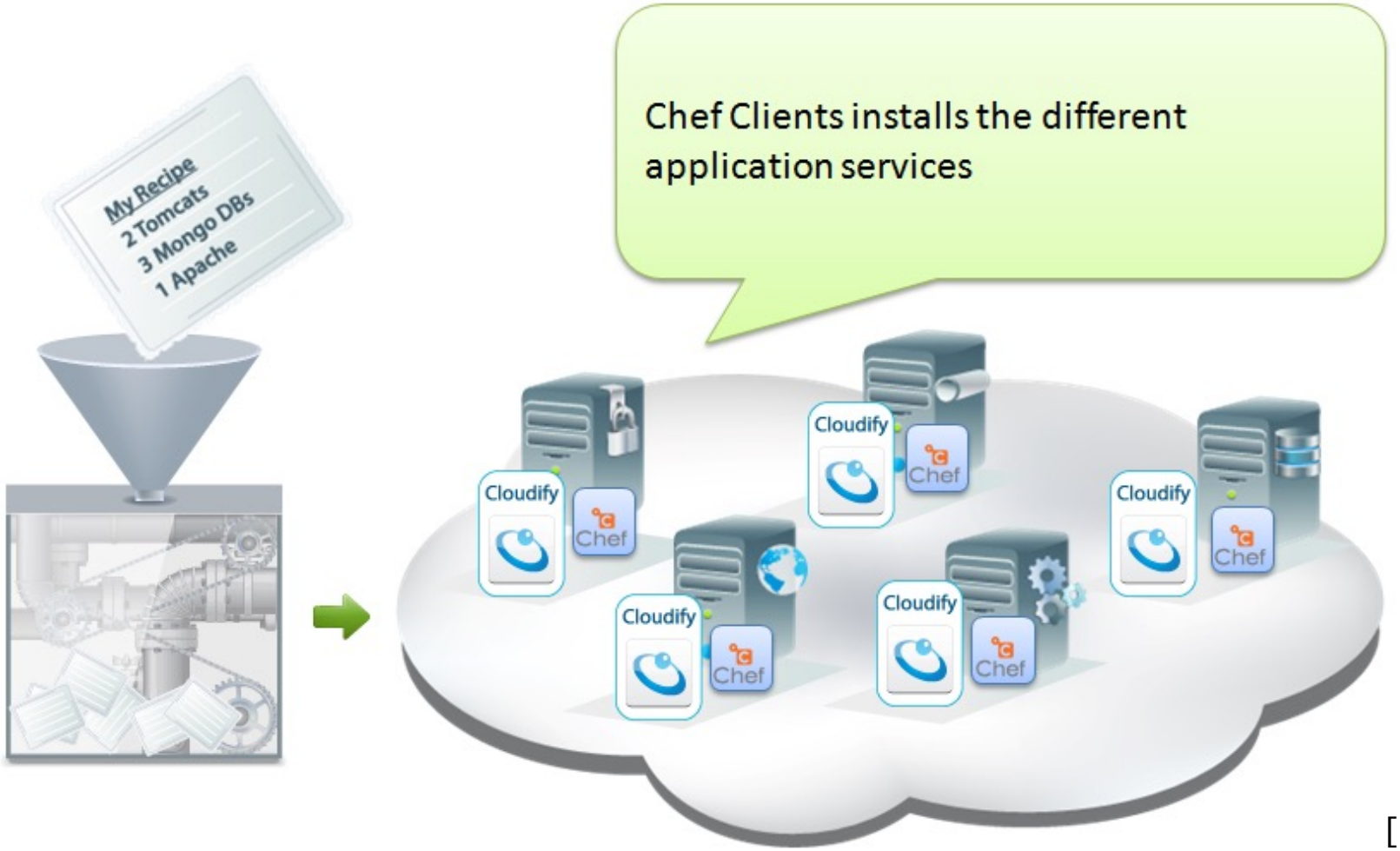
Konfiguration einer AWS-Instanz mit Terraform

```
PS C:\Automation\AWSExample1> terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but
will not be persisted to local or remote state storage.


The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed. Cyan entries are data sources to be read.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

+ aws_instance.2ninjasexample1
    ami:                          "ami-13be557e"
    availability_zone:            "<computed>"
    ebs_block_device.#:           "<computed>"
    ephemeral_block_device.#:     "<computed>"
    instance_state:               "<computed>"
    instance_type:                "t2.micro"
    key_name:                     "<computed>"
    network_interface_id:         "<computed>"
    placement_group:              "<computed>"
    private_dns:                  "<computed>"
    private_ip:                   "<computed>"
    public_dns:                   "<computed>"
    public_ip:                    "<computed>"
    root_block_device.#:          "<computed>"
    security_groups.#:            "<computed>"
    source_dest_check:            "true"
    subnet_id:                    "<computed>"
    tenancy:                      "<computed>"
    vpc_security_group_ids.#:     "<computed>"


Plan: 1 to add, 0 to change, 0 to destroy.
PS C:\Automation\AWSExample1>
```

- **Provisioning**
  - Automatisiertes Deployment von Applikationen (SaaS)
  - beinhaltet Systemkonfiguration, Softwareinstallationen und die Übertragung aller notwendigen Daten für den Betrieb einer Applikation
  - Wichtig für Test-Automatisierung, Continious Delivery and Integration
  - DevOps - ”You build it, you run it“ - Entwicklung und Betrieb der Anwendungen/Services durch ein Team

- **Tools**
  - Ansible, Chef, Puppet, Cloudify

Bsp. Chef


My Recipe
2 Tomcats
3 Mongo DBs
1 Apache
Chef Clients installs the different application services
Cloudify
Chef

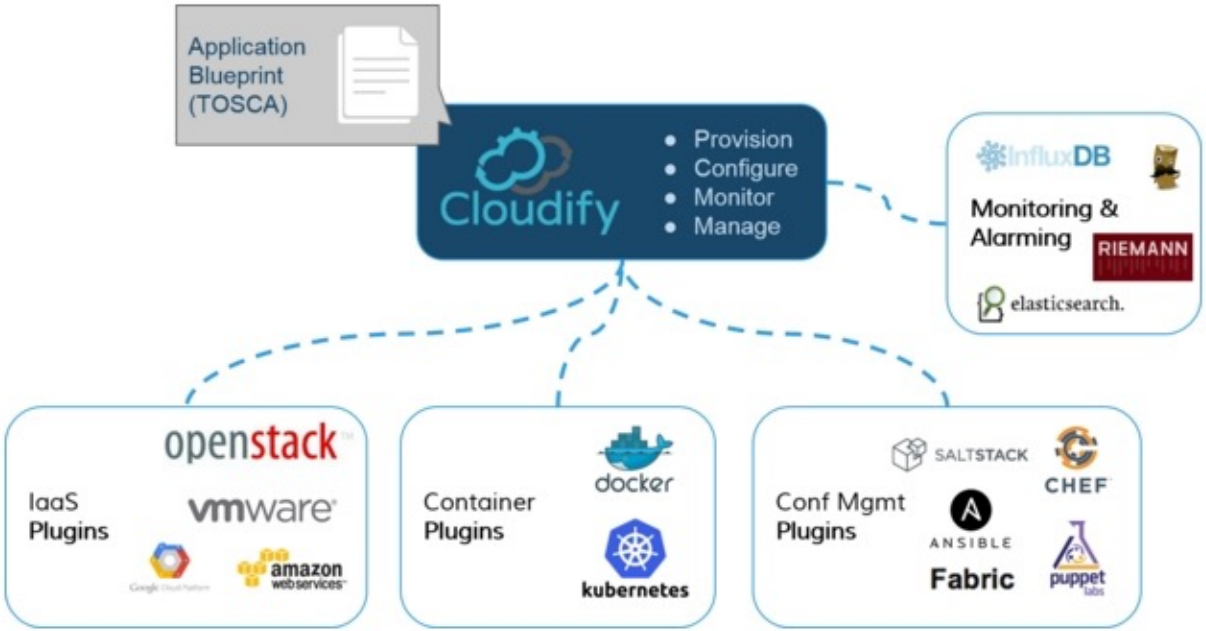
[7]

Bsp. Chef

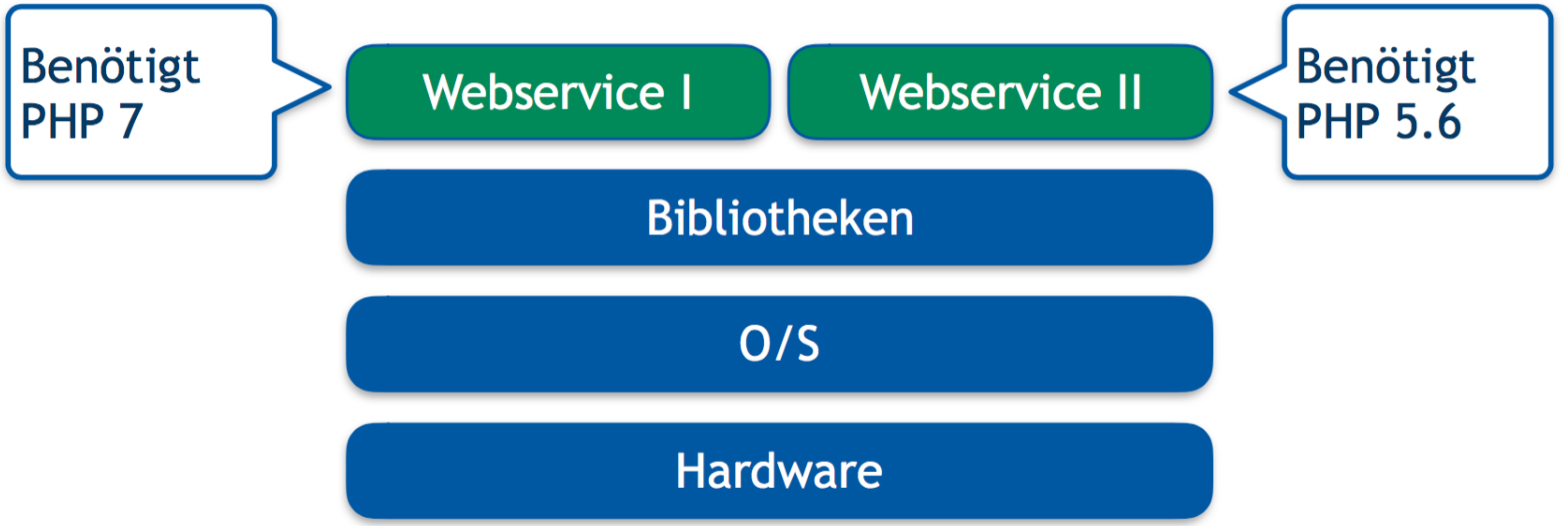
- **Chef-Konfiguration** besteht aus *Cookbooks*, welche einzelne *Recipes* enthalten
- **Cookbook** bezeichnet eine vollständige Konfiguration für einen oder mehrere Nodes, welche auf entsprechende Instanzen deployed werden
- **Recipes** sind Unterkonfigurationen, beispielsweise für eine Datenbank, Web-Server oder andere Konfigurationen
- **Chef-Server** ist die zentrale Verwaltungseinheit, verwaltet Repository für die erstellten Cookbooks, enthält alle Informationen über die Nodes, provisioniert diese
- **Workstation** ist ein Rechner, von dem aus die gesamte Administration der Cookbooks und Netzwerke stattfindet. Hier werden die Konfigurationen für einzelne Nodes erstellt, getestet und auf den Chef-Server abgespeichert
- **Nodes** sind physische Server oder Virtual Machines, die von Chef verwaltet werden

Orchestrierung verschiedener Cloud-Konfigurations- und Provisioning-Tools zur "voll"-automatisierten Bereitstellung von Multi-Thier-Applikationen in der Cloud

- Erstellung der Instanzen und Cluster in Multicloud-Umgebungen
- Provisioning der Plattformdienste und Applikationen
- Monitoring, Self-Healing, Auto-Scaling, Disaster Recovery Automation

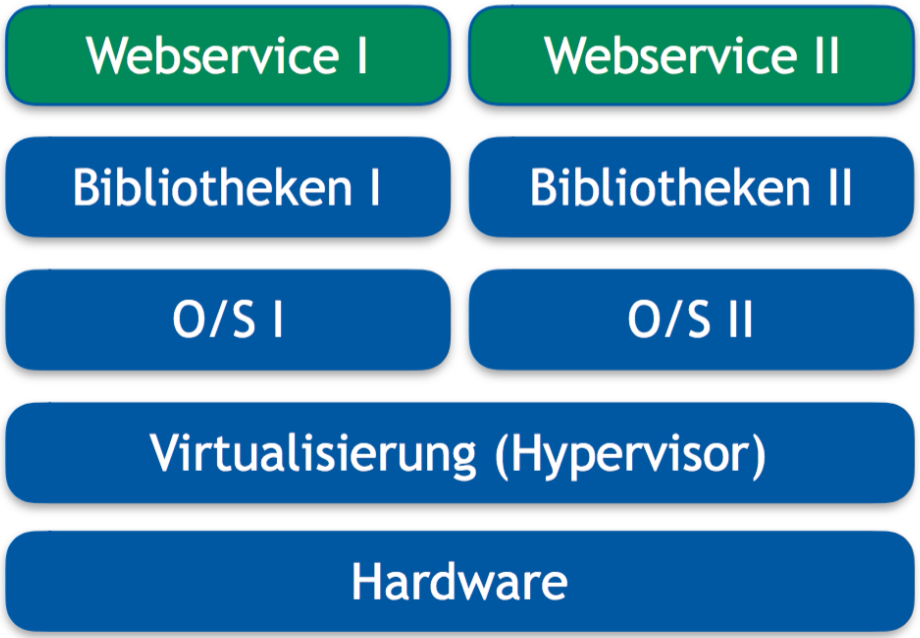


Klassisches Deployment von Webservices



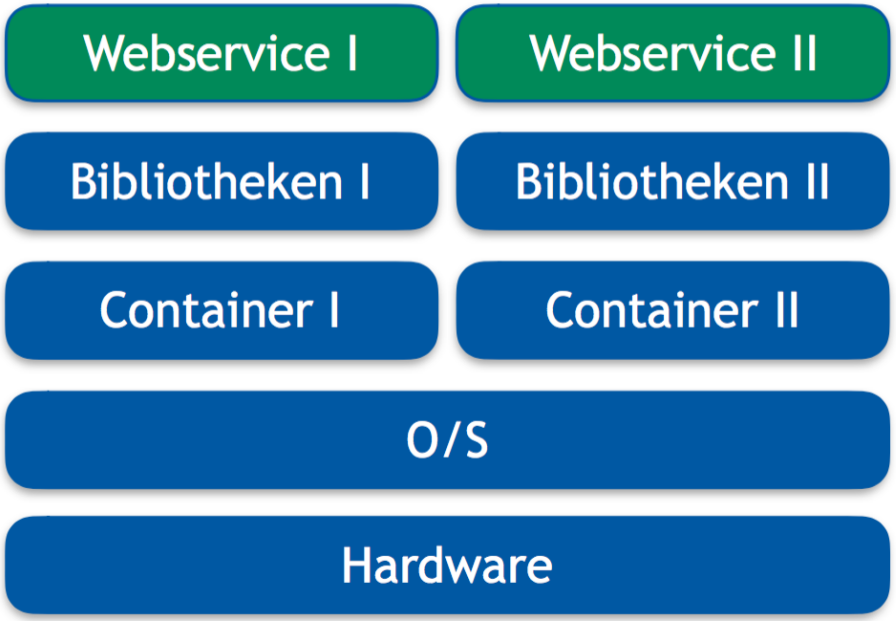
- Kollision, da unterschiedliche Versionen nötig

Lösung durch Virtualisierung



- Admin muss mehrere Betriebssysteme aktualisieren, absichern, etc.
- Overhead

Lösung durch Container (z.B. Docker)



- Keine Virtualisierung des Betriebssystems notwendig
- Anwendung und alle benötigten Bibliotheken als Image in Container ausführbar

3-Phasen-Konzept

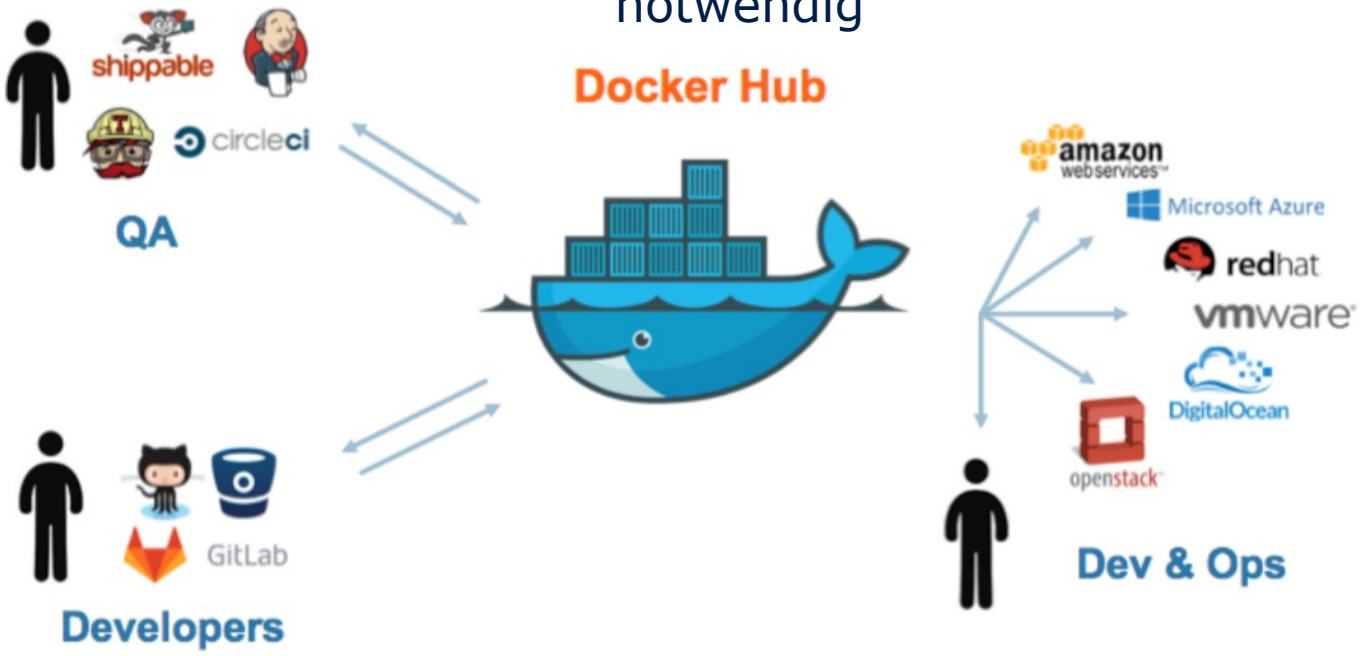
Build: Erstellung von Images

Ship: Verteilung der Images mit Docker Hub

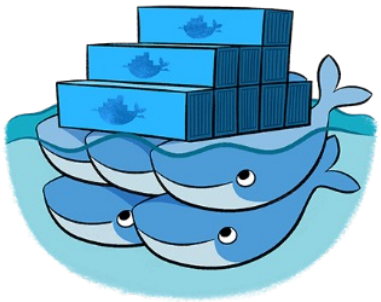
Run: Ausführung der Images mit der Docker Engine

Container (Instanz von Image)

- ohne Virtualisierung ausgeführt
- Nutzung von Linux-Technologien
- Namespaces, Control groups, Union file system
- Auf Windows Virtualisierung notwendig



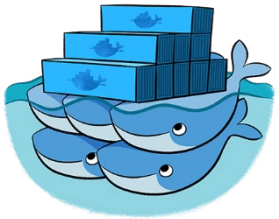
- Vorteil von Containern: Abgeschlossenheit erleichtert die Skalierung
- Containerorchestrierung ermöglicht von einer zentralen Stelle mehrere Knoten und Container zu verwalten
- Container werden automatisch verteilt
- Zugriff auf Container erfolgt transparent
- Populäre Orchestrationstools:
  - Docker Swarm
  - Kubernetes



Docker Swarm - Beispiel

- direkt in Docker Engine integriert
- „Schwarm" aus mehreren Docker-Hosts (Worker + Manager)
- Swarm Manager stellt den gewünschten Zustand (desired state) her

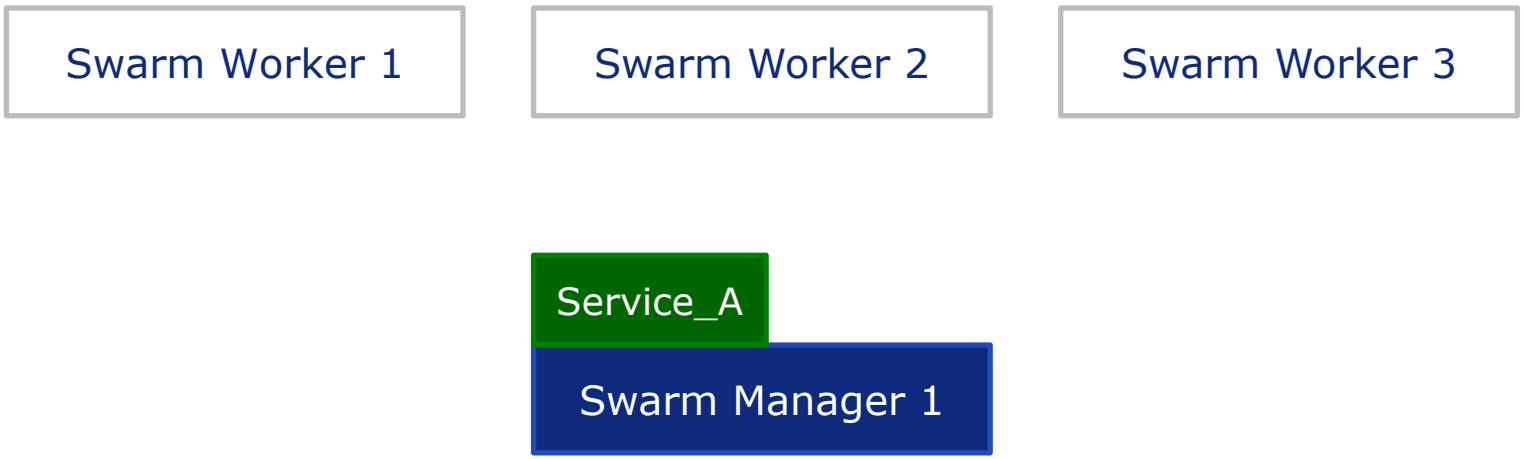
Swarm Worker 1 | Swarm Worker 2 | Swarm Worker 3



Swarm Manager 1

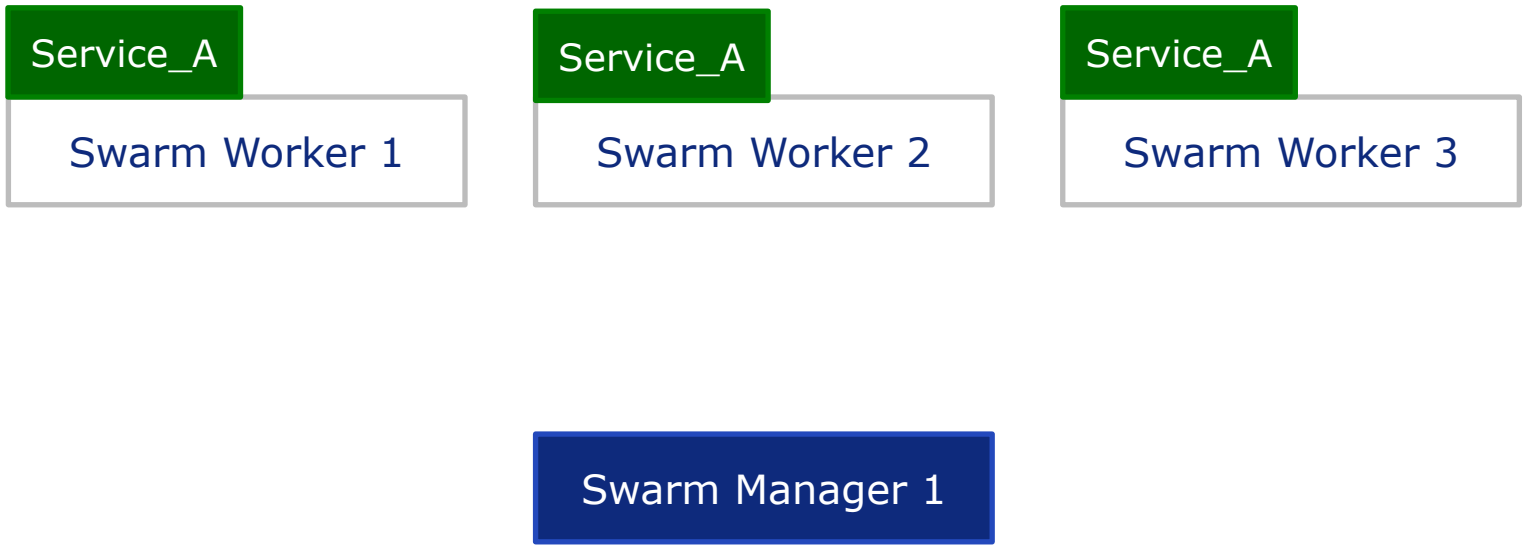
```
docker service create
--name Service_A nginx:latest
```

Docker Swarm - Beispiel



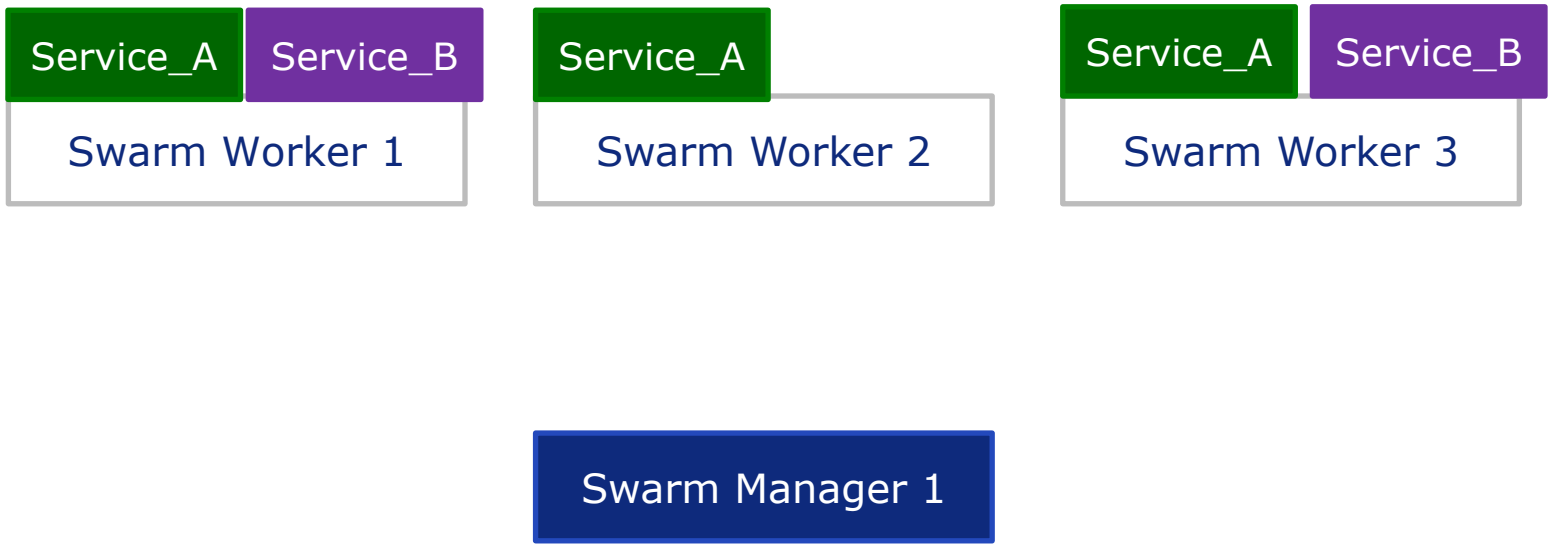
docker service scale Service_A=4

Docker Swarm - Beispiel



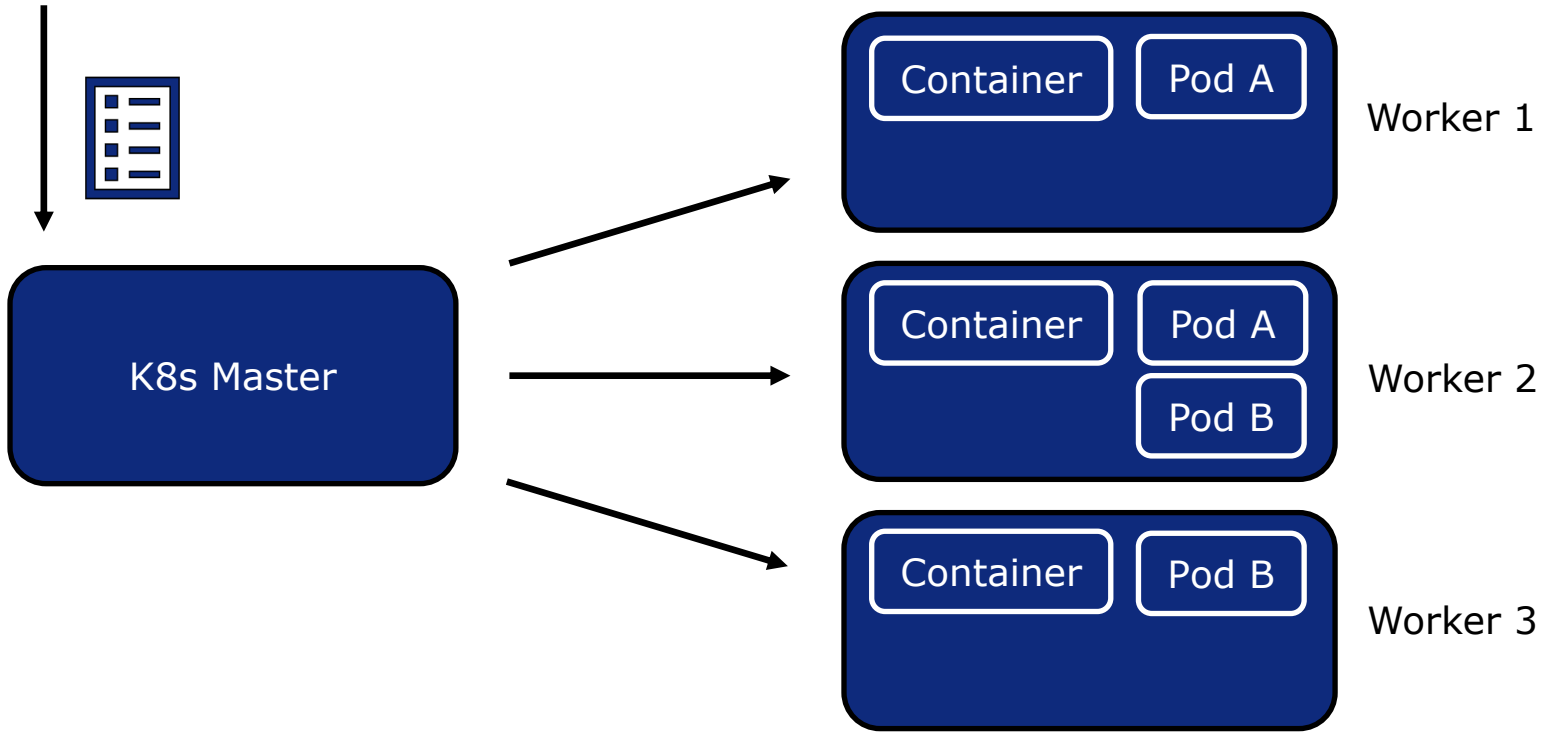

```
docker service create --replicas=2
--constraint node.role==worker --name Service_B redis:latest
```

Docker Swarm - Beispiel



Containerorchestrierungsplattform

- bietet Plattform zur Verwaltung von komplexen Containerinfrastrukturen
- „Kubernetes“ ist griechisch und bedeutet Steuermann
- baut auf 15 Jahre Forschung, Entwicklung von Google auf (Borg)
- seit 2015 Quelloffen und eines der Topprojekte auf Github
- 2015 Cloud Native Computing Foundation (CNCF) gegründet, Projekt der Linux Foundation
- Hoch erweiterbar durch Entwicklung eigener Komponenten (z.B. Scheduler)
- „pets“ (Pflege einzelner Instanzen) vs. „cattle“ (Pflege der Plattform)


K8s Master
Container
Pod A
Worker 1
Container
Pod A
Pod B
Worker 2
Container
Pod B
Worker 3

Kubernetes (K8s) - Komponenten

kube-apiserver: Zentrale Schnittstelle für alle Kubernetes-Ressourcen (Abfragen, Änderungen, Daten-Austausch)

kube-controller-manager: führt Kontrollschleife aus und leitet (Re-)Aktionen an API-Server weiter

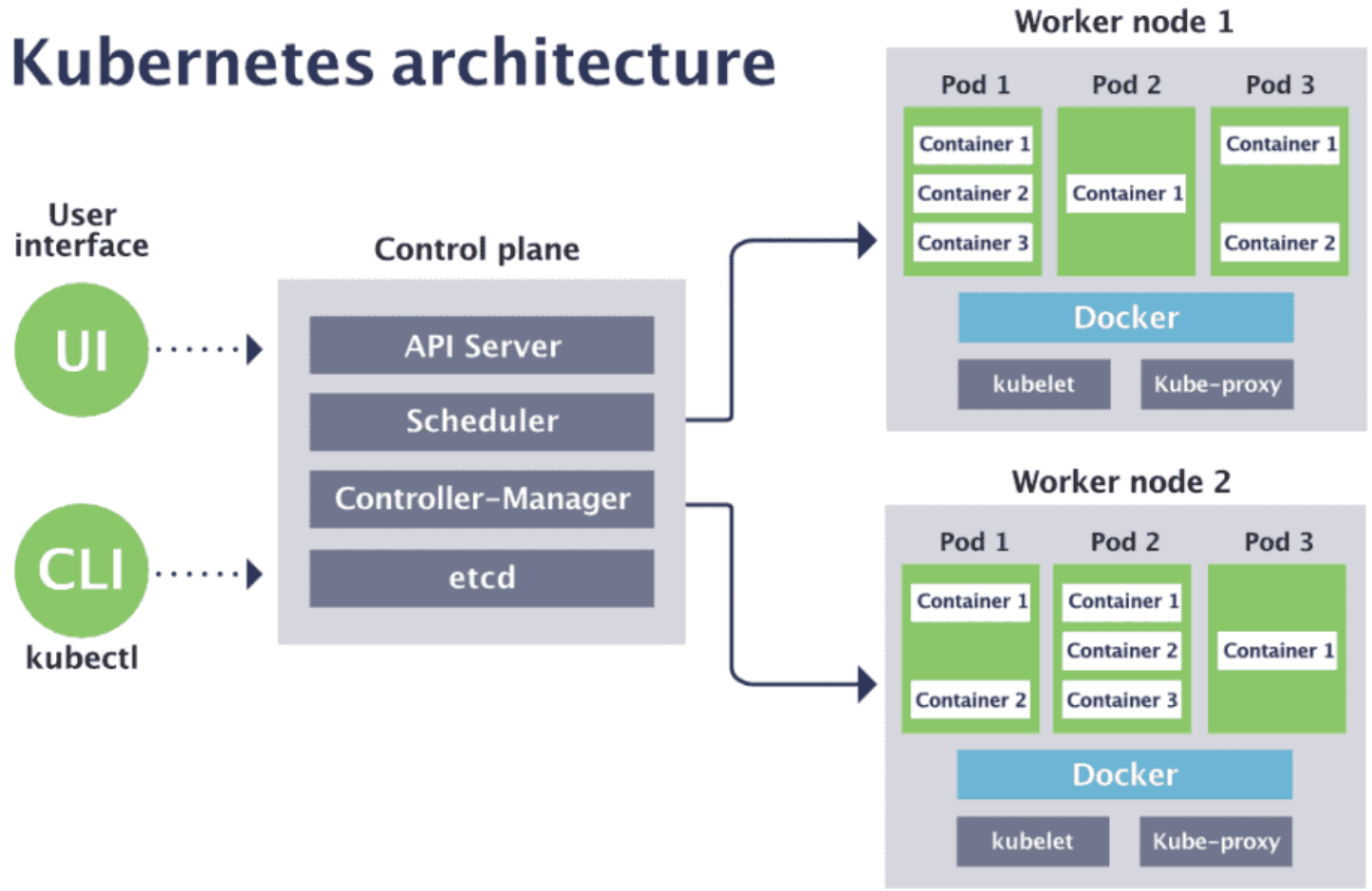
kube-scheduler: Verteilt Pods in vorhandener Infrastruktur (Ausführung auf Nodes)

kube-etcd: primärer Datenspeicher, sichert und repliziert alle Zustände der Kubernetes-Cluster

kubelet: Agent auf jedem Node im Cluster, stellt sicher, dass Container in einem Pod ausgeführt werden

kube-proxy: Netzwerk-Proxy auf jedem Node, für die Verwaltung der virtuellen IP-Adressen der Pods und Einhaltung der Netzwerkregeln

Kubernetes (K8s) - Architektur



https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/

Zusammenfassung

- **Ziel:** Bedarfsgerechte Bereitstellung von Cloud-Ressourcen
  - Setzt Monitoring voraus, Health checks
  - Horizontale Skalierung und Load Balancing zur Vermeidung von Bottlenecks
  - Elastizität durch Autoscaling bei Lastspitzen
  - Ausfallsicherheit durch AutoHealing
- **Voraussetzung:** Automatisierung der Cloud-Konfiguration und des Plattform- und Applikations-Provisioning
  - Terraform, sparkleformation, cloudify – Multicloud-Ansteuerung, Cloud-Konfiguration
  - Chef, Puppet, Ansible – Cloud-Konfigurationsmanagement und Provisioning
  - Docker – Deployment von Applikationen und benötigten Plattformdiensten in Containern
  - Docker-Swarm und Kubernetes – Container-Orchestrierung

Referenzen

1. Peters, D.K.: Automated Testing of Real-Time Systems. S.3. 1999. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.4833&rep=rep1&type=pdf
2. Ward, J. S., Barker, A. Observing the clouds: a survey and taxonomy of cloud computing. In: Journal of Cloud Computing: Advances, Systems and Applications. 2014. http://link.springer.com/article/10.1186/s13677-014-0 024-2
3. Lampesberger, H., Rady, M. Monitoring of Client-Cloud Interaction. S. 2ff. 2015. http://link.springer.com/chapter/10.1007/978-3-319-17112-8 _6#page-1
4. Amazon CloudWatch. https://aws.amazon.com/de/?nc2=h_lg
5. Google Google Compute Engine Documentation, https://cloud.google.com/compute/docs/
6. Microsoft Azure. https://azure.microsoft.com/de-de/
7. Chef, Tutorials - Learn Chef, https://learn.chef.io/tutorials/
8. Cloudify http://getcloudify.org/configuration_management_ssh_cloud_orchestration_puppet_ansible_chef_open_source.html
9. Gunther, N. J.: How to Quantify Scalability - The Universal Scalability Law (USL). http://www.perfdynamics.com/Manifesto/USLscalability.html

Referenzen

10. https://www.docker.com/resources/what-container
11. https://www.datadoghq.com/docker-adoption/
12. https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016
13. https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/

Weiterführende Tutorials

Learn how to scale your applications with Google Compute Engine
https://www.youtube.com/watch?v=TfbEwfYjKl4

Load Balancing with Cloud Computing
https://www.youtube.com/watch?v=_xH5POea0Jc

From Zero to Docker - Tutorial for Beginners
https://www.youtube.com/watch?v=JprTjTViaEA

Learn Docker in 20 Minutes
https://www.youtube.com/watch?v=wCTTHhehJbU