

Cloud Native Softwarearchitektur







Kategorien



- Business Education Test Prep
- Medical Engineering Computer

See all textbooks,

Furnish the Dorm



that fits your

study habits.

See all desks.

living space and





Sleep well with our low prices on bedding. See all college bedding.

Make dorm life easier with refrigerators, dishwashers and more.

See all small appliances,

Stay in Touch

notebook PCs

and printers





Get online for

only \$9.94 a



Call anytime for as low as 5 cents ner

Click here to get started. Games Gallery

PlayStation 2 GameCube

Xbox

See More

Back To College

Portable Electronics

Cameras **Shopping Cards**

See More

(Ko12

Clothing School Supplies

See More







Refill Prescriptions

Registrierung

Warenkorb

Artikel

Dienste

More... Tires

Electronics

Photo Digital Cameras

More...

 Computers & Printers Internet Service

Movies 12,000 Movies!

Books 500,000 Titles!

Music 80,000 CDs1

. TV, DVD & Video More...

Camcorders
 Photo Center

Video Games * PS2

. Outdoor Play

Furniture

· Appliances

· Bed & Bath

More

. Bikes & Scooters More...

Baby & Baby Registry

. Shop by Age or Stage

Xbox

More...

Toys

Sporting Goods . Exercise & Hot Tubs

Garden & Hardware

· Home Improvement

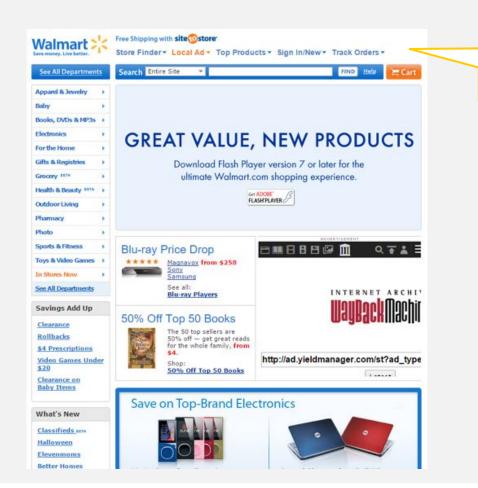
· Patio & Grilling Lawn Equipment

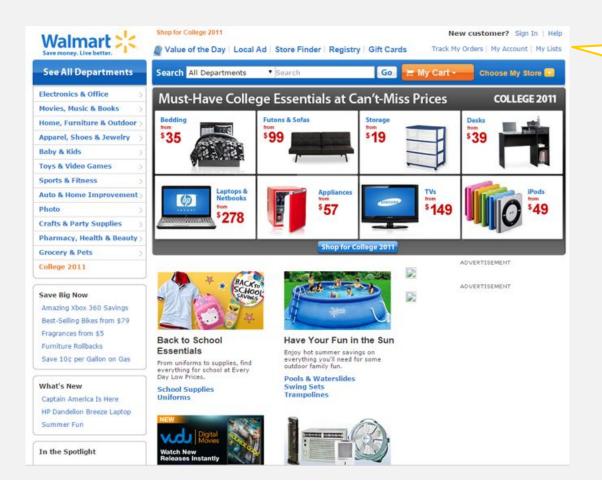
- · Camping & Hiking More...
- Gifts



Zusätzliche

Dienste





Zusätzliche Dienste (z.T. personalisiert)

Bad News



CHALLENGE

Four years ago, the Walmart Global eCommerce system was a monolithic application, deployed once every 2 months. This represented an unsustainable rate of innovation given the competitive climate. Walmart recognized the need to fundamentally transform technology delivery to compete in the Digital Economy.

Walmart auf http://www.oneops.com

- "[...] it was unable to scale for 6 million pageviews per minute and was down for most of the day during peak events."
- "This is the multi-million dollar question which the IT Department of Walmart Canada had to address after they were failing to provide to their users on Black Fridays for two years in a row."

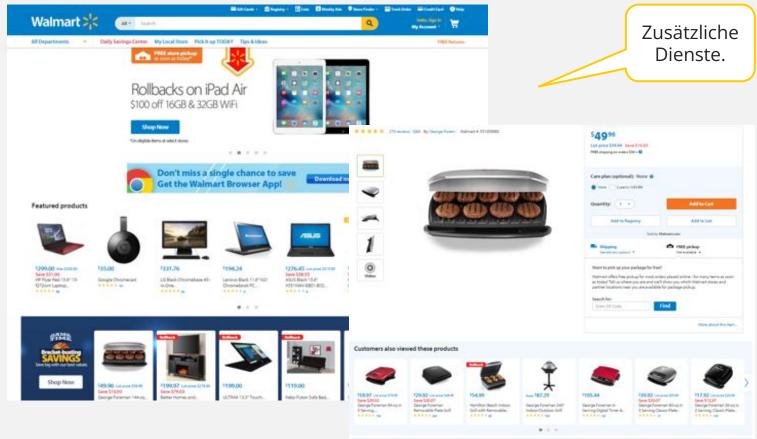
https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures

Lange Zyklen von Dev-to-Prod.

Mangelnde Skalierbarkeit.

Mangelnde Elastizität.





Good News



1000 Deployments pro Tag ...
... durch die Entwickler

~ 100% Verfügbarkeit

Neue Geschäftsmodelle, Anwendungen und Geräte (IoT, mobile, APIs)

Ressourcen Sparsamkeit

Elastische Skalierbarkeit

RESULTS

Today the Walmart eCommerce platform is hosted in some of the largest OpenStack clouds and is managed exclusively via OneOps. On a typical day there are now over 1,000 deployments, executed on-demand by development teams, each taking only minutes on average to complete.

Walmart auf http://www.oneops.com

"They wanted to prepare for the world by 2020, with 4 billion people connected, 25+ million apps available, and 5.200 GB of data for each person on Earth. Walmart replatformed [...] with the intention of achieving close to 100% availability with reasonable costs."

https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures

- "In fact, the organization reports that some 3,000 engineers [...] drive 30,000 changes per month to Walmart software."
- "Those new applications, which span everything from mobile devices to the Internet of things (IoT), are crucial weapons in a global e-commerce contest that pits Walmart against the likes of Amazon and Alibaba, as well as a host of other rivals that are emerging as the cost of entry into the online retail sector continues to decline in the age of the API economy."

http://www.baselinemag.com/enterprise-apps/walmart-embraces-microservices-to-get-more-agile.html

- "The Walmart [...] servers [...] were able to handle all mobile Black Friday traffic with about 10 CPU cores and 28Gb RAM."
- "On Thanksgiving weekend, Walmart servers processed 1.5 billion requests per day. 70 percent of which were delivered through mobile."

http://techcrunch.com/2014/12/02/walmart-com-reports-biggest-cyber-monday-in-history-mobile-traffic-at-70-over-the-holidays

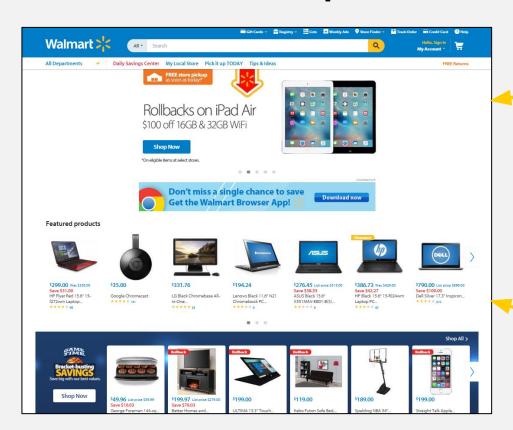


Von Betriebsmonolithen ...





... zu Betriebskomponenten





Cloud Native Application Development: Components All Along the Software Lifecycle





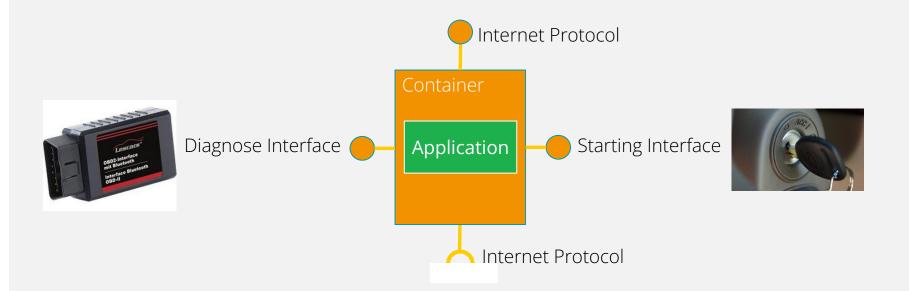
- Complexity unit
- Data integrity unit
- Coherent and cohesive features unit
- Decoupled unit

- Planning unit
- Team assignment unit
- Knowledge unit
- Development unit
- Integration unit

- Release unit
- Deployment unit
- Runtime unit (crash, slow-down, access)
- Scaling unit

Die Anatomie einer Betriebs-Komponente



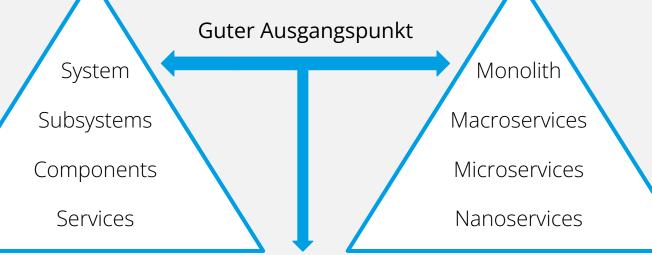


Dev Components



Ops Components





Decomposition Trade-Offs

- + More flexible to scale
- + Runtime isolation (crash, slow-down, ...)
- + Independent releases, deployments, teams
- + Higher utilization possible

- Distribution debt: Latency
- Increasing infrastructure complexity
- Increasing troubleshooting complexity
- Increasing integration complexity

Regel #1 für den Betrieb in der Cloud



"Everything fails all the time." — Werner Vogels, CTO of Amazon

Regel #2 für den Betrieb in der Cloud



Soll nur der Himmel die Grenze sein, dann funktioniert nur horizontale Skalierung.



Regel 3 für den Betrieb in der Cloud



Wer in die Cloud will, der sollte Cloud sprechen.

Die 5 Gebote der Cloud

- Everything fails all the time
- Focus on MTTR, not on MTTF
- Respect the eight fallacies of distributed computing
- Scale out, not up
- Treat resources as cattles, not as pets



Quelle: https://de.wikipedia.org/wiki/Zehn Gebote

Eight fallacies of distributed computing



Cloud-Architektur aus Sicht der Softwarearchitektur: Design for Failure.



- Jede Komponente läuft eigenständig und isoliert Betriebskomponenten
- Die Betriebskomponenten kommunizieren untereinander über Internet-Protokolle □ HTTP, UDP, ...
- Jede Betriebskomponente kann in mehreren Instanzen laufen und bietet damit Redundanz. Es gibt keinen "Common Point of Failure". Cluster Orchestrator
- Jede Betriebskomponente besitzt Diagnoseschnittstellen um ein fehlerhaftes Verhalten erkennen zu können
- Jeder Microservice kann zu jeder Zeit neu gestartet und auf einem anderen Knoten in Betrieb genommen werden. Er besitzt keinen eigenen Zustand.
- Die Implementierung hinter einem jeden Microservice kann ausgetauscht werden, ohne dass die Nutzer davon etwas bemerken.

Betriebskomponenten benötigen eine Infrast<u>ruktur um</u>

sie herum: Eine Micro-Service-Plattform.

Typische Aufgaben:

- Authentifizierung
- Load Shedding
- Load Balancing
- Failover
- Rate Limiting
- Request Monitoring
- Request Validierung
- Caching
- Logging

Typische Aufgaben:

- HTTP Handling
- Konfiguration
- Diagnoseschnittstelle
- Lebenszyklus steuern
- APIs bereitstellen

Edge Server

Service Container

Service

Typische Aufgaben:

- Service Discovery
- Load Balancing
- Circuit Breaker
- Request Monitoring

Service Client

Diagnosestecker

Typische Aufgaben:

- Metriken sammeln
- Logs sammeln
- Trace sammeln

Service Discovery

Configuration & Coordination

Typische Aufgaben:

- Aggregation von Metriken
- Sammlung von Logs
- Sammlung von Traces
- Analyse / Visualisierung
- Alerting

Typische Aufgaben:

- Service Registration
- Service Lookup
- Service Description
- Membership Detection
- Failure Detection

Typische Aufgaben:

- Key-Value-Store (oft in Baumstruktur. Teilw. mit Ephemeral Nodes)
- Sync von Konfigurationsdateien
- Watches, Notifications, Hooks, Events
- Koordination mit Locks, Leader Election und Messaging
- Konsens im Cluster herstellen

ce Monitoring

Das Cloud Native Application Reifegradmodell



Cloud Native

- Microservices architecture
- Domain- and API-driven design

Cloud Resilient

- Fault-tolerant and resilient design
- Cloud-agnostic runtime implementation
- Bundled metrics and monitoring (diagnosability)
- Proactive failure testing (chaos testing)

Cloud Friendly

- 12 factor-app-methodology
- Horizontally scalable
- Leverages platform for high availability

Cloud Ready

- No permanent disk access (within container)
- Self-contained application
- · Platform-managed ports and networking
- Consumes platform-managed backing services



Übung 1 (20 min): Twelve Factor Apps

12 Factor App



- Codebase
 One codebase tracked in revision control, many deploys.
- Port binding
 Export services via port binding.
- Dependencies
 Explicitly declare and isolate dependencies.
- 8 Concurrency
 Scale out via the process model.

Configuration
Store config in the environment.

- Disposability

 Maximize robustness with fast startup and graceful shutdown.
- Backing Services
 Treat backing services as attached resources.
- Dev/Prod Parity
 Keep development, staging, and production as similar as possible

- Build, release, run
 Strictly separate build and run stages.
- Logs
 Treat logs as event streams.
- Processes

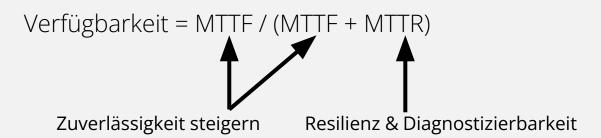
 Execute the app as one or more stateless processes.
- Admin processes
 Run admin/management tasks as one-off processes.



Resilienz

Resilienz



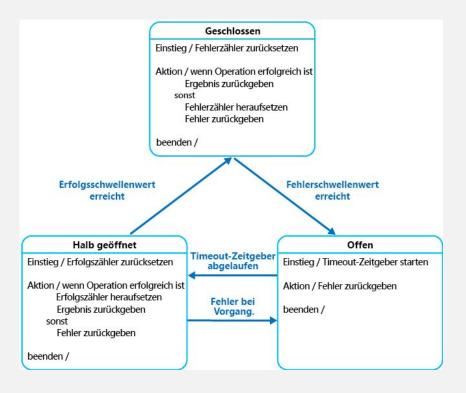


Resilienz: Die Fähigkeit eines Systems mit unerwarteten und fehlerhaften Situationen umzugehen

- Ohne dass es der Nutzer merkt (Bestfall)
- Mit ein einer "graceful degradation" des Services (schlechtester Fall)

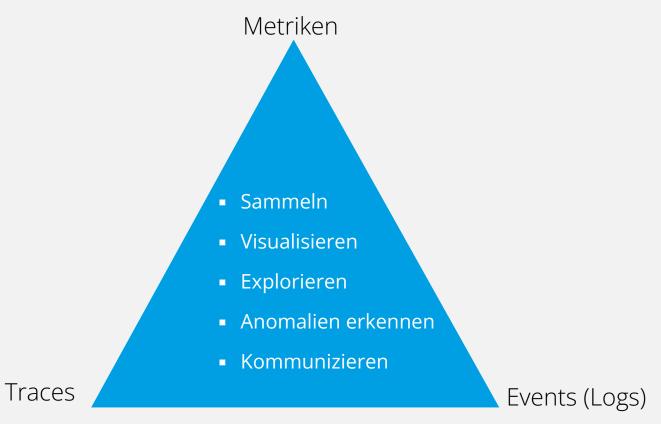
Resilienz-Pattern: Circuit Breaker





Diagnostizierbarkeit

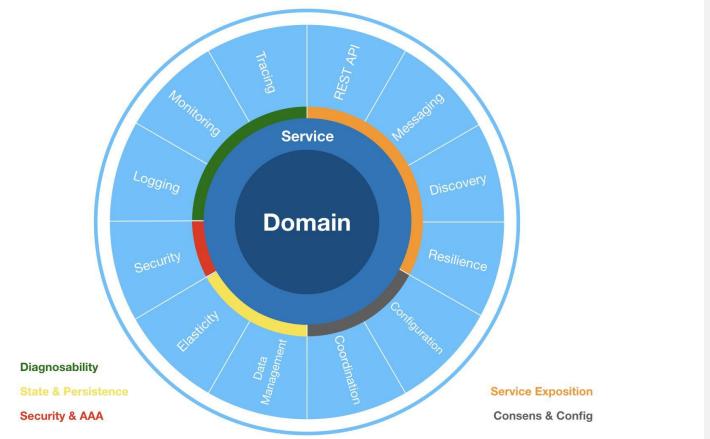




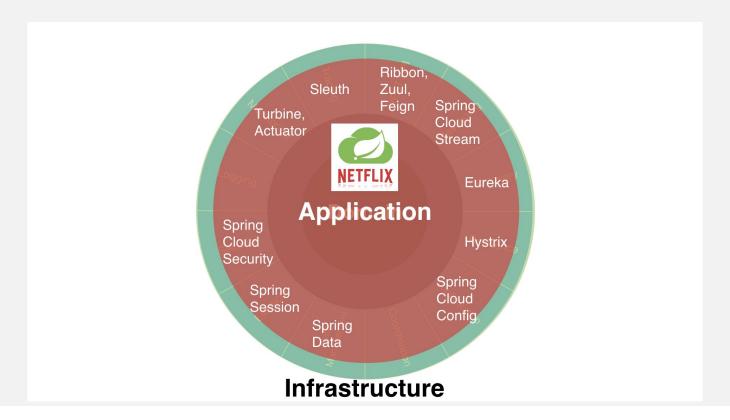


Technische Aspekte von Microservices

Technische Aspekte von Microservices



Technische Aspekte von Microservices: Die Library-Lösung

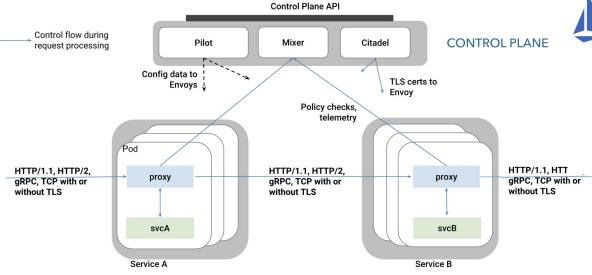


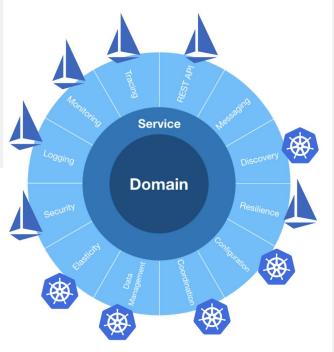
Technische Aspekte von Microservices: Die Infrastruktur-Lösung



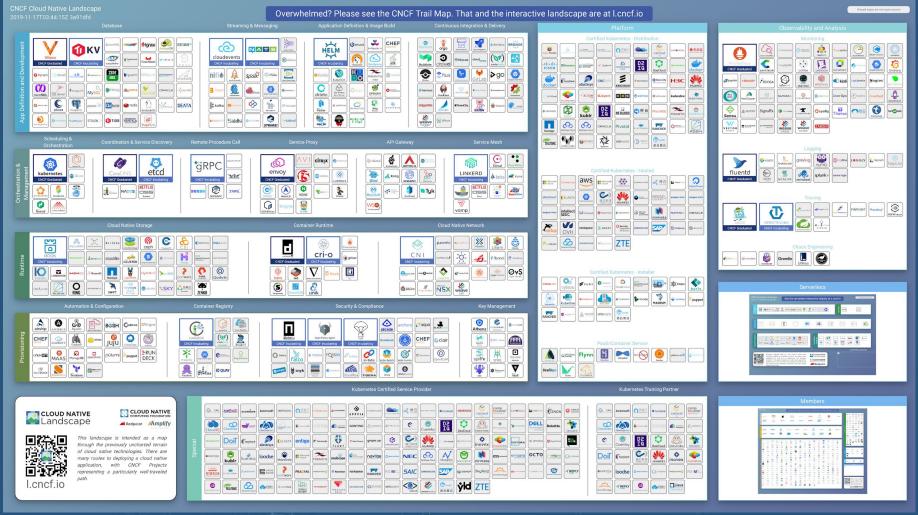
Istio Service Mesh

- Open-Source-Projekt von Google, IBM, Lyft, RedHat u.A.
- Aufsatz auf Kubernetes, der wichtige technische Aspekte
- auf Infrastruktur-Seite ergänzt





INGRESS SERVICE MESH EGRESS 33

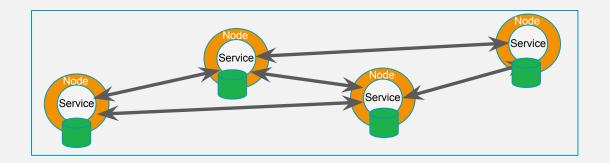




Konfiguration & Koordination: Verteilter Zustand und Konsens

Ein verteilter Konfigurationsspeicher





Wie wird der Zustand des Konfigurationsspeichers im Cluster synchronisiert?

Das CAP-Theorem



Theorem von Brewer für Eigenschaften von zustandsbehafteten verteilten Systemen – mittlerweile auch formal bewiesen. *

Es gibt drei wesentliche Eigenschaften, von denen ein verteiltes System nur zwei gleichzeitig haben kann:

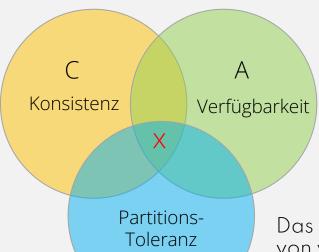
- Konsistenz (C)
- Verfügbarkeit (A)
- Partitionstoleranz (P)

^{*} Brewer, Eric A. "Towards robust distributed systems." PODC. 2000.

Das CAP-Theorem



Alle Knoten sehen dieselben Daten zur selben Zeit. Alle Kopien sind stets gleich.



Das System läuft auch, wenn einzelne Knoten ausfallen. Ausfälle von Knoten und Kanälen halten die überlebenden Knoten nicht von ihrer Funktion ab.

Im Falle einer Netzwerk-Partition muss man sich zwischen Konsistenz und Verfügbarkeit entscheiden, beides geht nicht.

Das System funktioniert auch im Fall von verlorenen Nachrichten. Das System kann dabei damit umgehen, dass sich das Netzwerk an Knoten in mehrere Partitionen aufteilt, die nicht miteinander kommunizieren.

Gossip-Protokolle für Hoch-Verfügbarkeit

Grundlage: Ein Netzwerk an Agenten mit eigenem Zustand

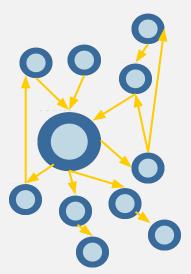
- Agenten verteilen einen Gossip-Strom
- Nachricht: Quelle, Inhalt / Zustand, Zeitstempel
- Nachrichten werden in einem festen Takt periodisch versendet
- an eine bestimmte Anzahl anderer Knoten (Fanout)

Virale Verbreitung des Gossip-Stroms

- Knoten, die mit mir in einer Gruppe sind, bekommen auf jeden Fall eine Nachricht
- Die Top x% an Knoten, die mir Nachrichten schicken bekommen eine Nachricht

Nachrichten, denen vertraut wird, werden in den lokalen Zustand übernommen, wenn

- die gleiche Nachricht von mehreren Seiten gehört wurde
- die Nachricht von Knoten stammt, denen der Agent vertraut
- keine aktuellere Nachricht vorhanden sind



Gossip-Protokolle für Hoch-Verfügbarkeit

Vorteile:

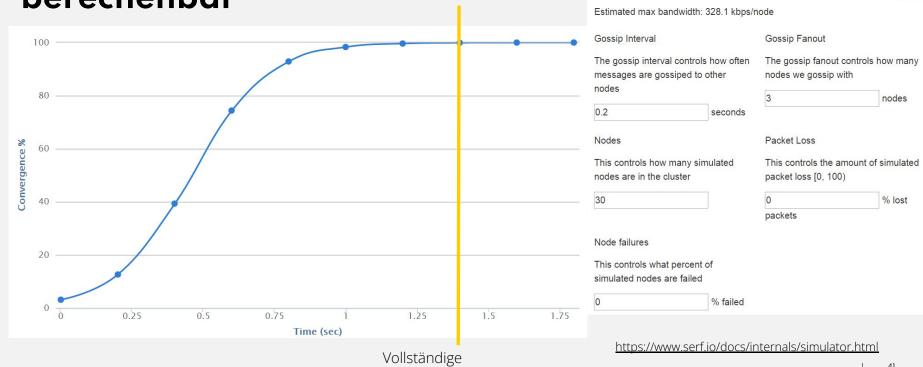
- Keine zentralen Einheiten notwendig.
- Fehlerhafte Partitionen im Netzwerk werden umschifft. Die Kommunikation muss nicht verlässlich sein.

Nachteile:

- Der Zustand ist potenziell inkonsistent verteilt (konvergiert aber mit der Zeit)
- Overhead durch redundante Nachrichten.

Die Konvergenz der Daten und damit der Zeitpunkt der vollständigen Konsistenz ist berechenbar





Konsistenz

Protokolle für verteilten Konsens: im Gegensatz zu Gossip-Protokollen konsistent, aber nicht hoch-verfügbar



Grundlage: Netzwerk an Agenten

Prinzip: Es reicht, wenn der Zustand auf einer einfachen Mehrheit der Knoten konsistent ist und die restlichen Knoten ihre Inkonsistenz erkennen.

Verfahren:

- Das Netzwerk einigt sich per einfacher Mehrheit auf einen Leader-Agenten initial und falls der Leader-Agent nicht erreichbar ist. Eine Partition in der Minderheit kann keinen Leader-Agenten wählen.
- Alle Änderungen laufen über den Leader-Agenten. Dieser verteilt per Multicast Änderungsnachrichten periodisch im festen Takt an alle weiteren Agenten.
- Quittiert die einfache Mehrheit an Agenten die Änderungsnachricht, so wird die Änderung im Leader und (per Nachricht) auch in den Agenten aktiv, die quittiert haben. Ansonsten wird der Zustand als inkonsistent angenommen.

Protokolle für verteilten Konsens: im Gegensatz zu Gossip-Protokollen konsistent, aber nicht hoch-verfügbar



Vorteile:

- Fehlerhafte Partitionen im Netzwerk werden toleriert und nach Behebung des Fehlers wieder automatisch konsistent.
- Streng konsistente Daten.

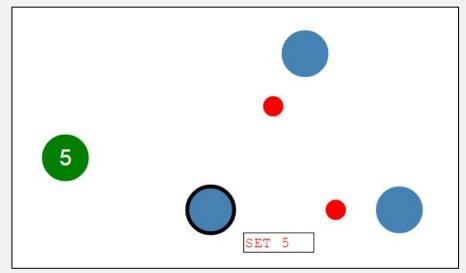
Nachteile:

- Der zentrale Leader-Agent limitiert den Durchsatz an Änderungen.
- Nicht hoch-verfügbar: Bei einer Netzwerk-Partition kann die kleinere Partition nicht weiterarbeiten. Ist die Mehrheit in keiner Partition, so kann insgesamt nicht weiter gearbeitet werden.

Das Raft-Konsens-Protokoll



Ongaro, Diego; Ousterhout, John (2013). "In Search of an Understandable Consensus Algorithm".



http://thesecretlivesofdata.com/raft

https://raft.github.io/



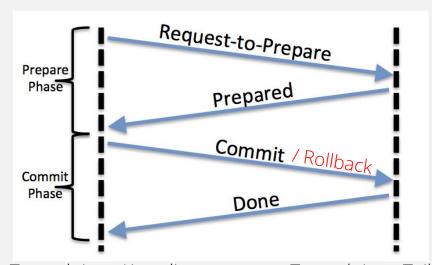
Übung 2 & 3 (30 min):

Das Raft-Konsens-Protokoll etcd

Ist strenge Konsistenz über alle Knoten notwendig, so verbleibt das 2-Phase-Commit Protokoll (2PC)



Ein Transaktionskoordinator verteilt die Änderungen und aktiviert diese erst bei Zustimmung aller. Ansonsten werden die Änderungen rückgängig gemacht.



Transaktions-Koordinator

Transaktions-Teilnehmer

Vorteil:

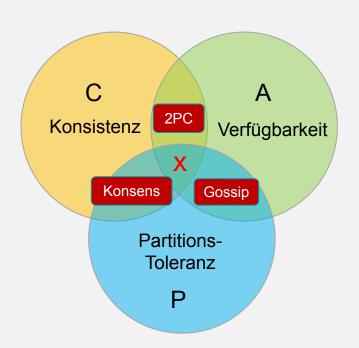
Alle Knoten sind konsistent zueinander.

Nachteile:

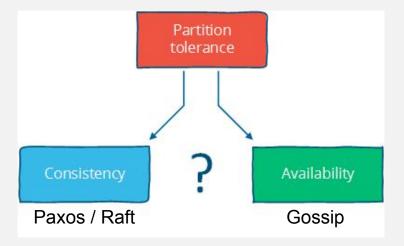
- Zeitintensiv, da stets alle Knoten zustimmen müssen.
- Das System funktioniert nicht mehr, sobald das Netzwerk partitioniert ist.

Die vorgestellten Protokolle und das CAP Theorem





In der Cloud müssen Partitionen angenommen werden. Damit ist die Entscheidung binär: Zwischen Konsistenz und Verfügbarkeit.





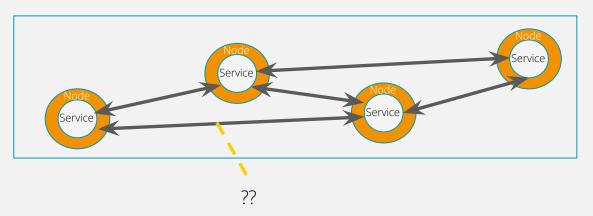
Address ipAndPort)



Service Discovery

Die Probleme einer klassischen Verknüpfung von Services in der Cloud





Probleme:

- Mangelnde Redundanz: Jeder Service wird direkt genutzt. Er kann nicht unmittelbar in mehreren Instanzen laufen, die Redundanz schaffen.
- Mangelnde Flexibilität: Die Services können nicht ohne Seiteneffekt neu gestartet oder auf einem anderen Knoten in Betrieb genommen werden – oder sogar durch eine andere Service-Implementierung ausgetauscht werden.

Lösungen:

- Dynamischer DNS
- Ambassador
- Dynamischer Konfigurationsdateien und Umgebungsvariablen

Das Ambassador Pattern

Ein Ambassador-Knoten für jede Knoten-Art (z.B. Webserver)

Node Service Ambassador Node Service

Service Registration:

- Beobachtet das Cluster und erkennt neue und kranke/tote Knoten in seiner Gruppe.
- Hinterlegt die aktuell aktiven Knoten im Konfigurationsspeicher.

Service Discovery: Der Client kommuniziert mit dem Ambassador-Knoten, der die Anfragen aber möglichst effizient an einen Knoten der Gruppe weiterreicht.

Der Ambassador-Knoten kann dabei eine Reihe an Zusatzdienste erweisen bei der Verbindung zum Service (Service Binding):

- Load Balancing inklusive Failover
- Service Monitoring
- Circuit Breaker Pattern
- Throttling



Übung 4: Consul & Traefik