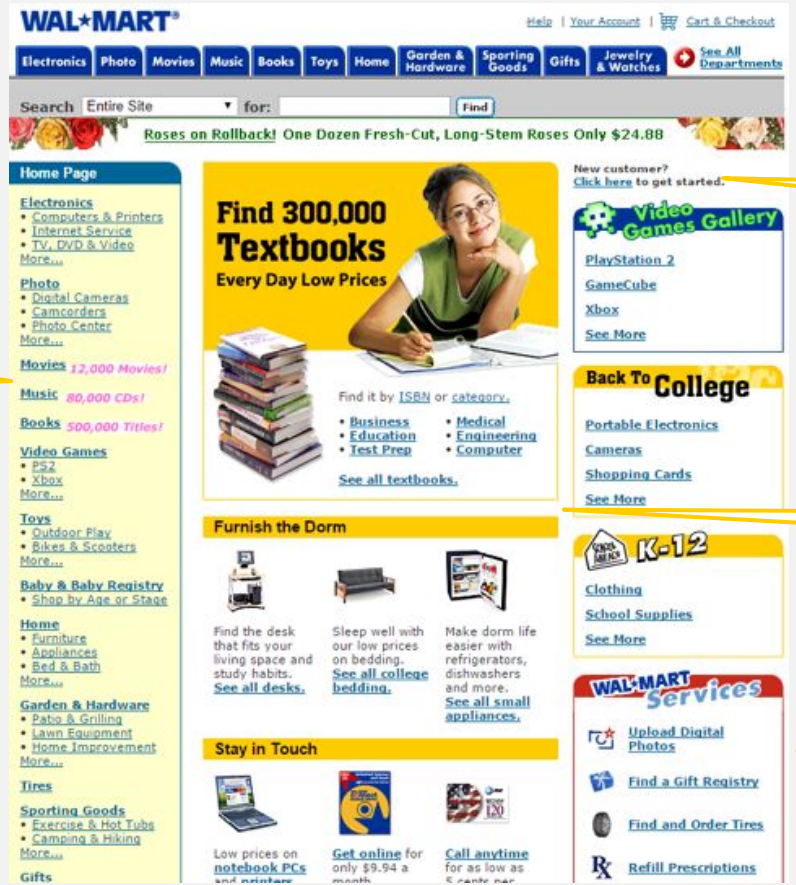
qaware.de

QA|WARE
SOFTWARE ENGINEERING

Cloud Native Software Architecture


Walmart
Supercentre
Enter
Exit
$2

2002



Shopping cart

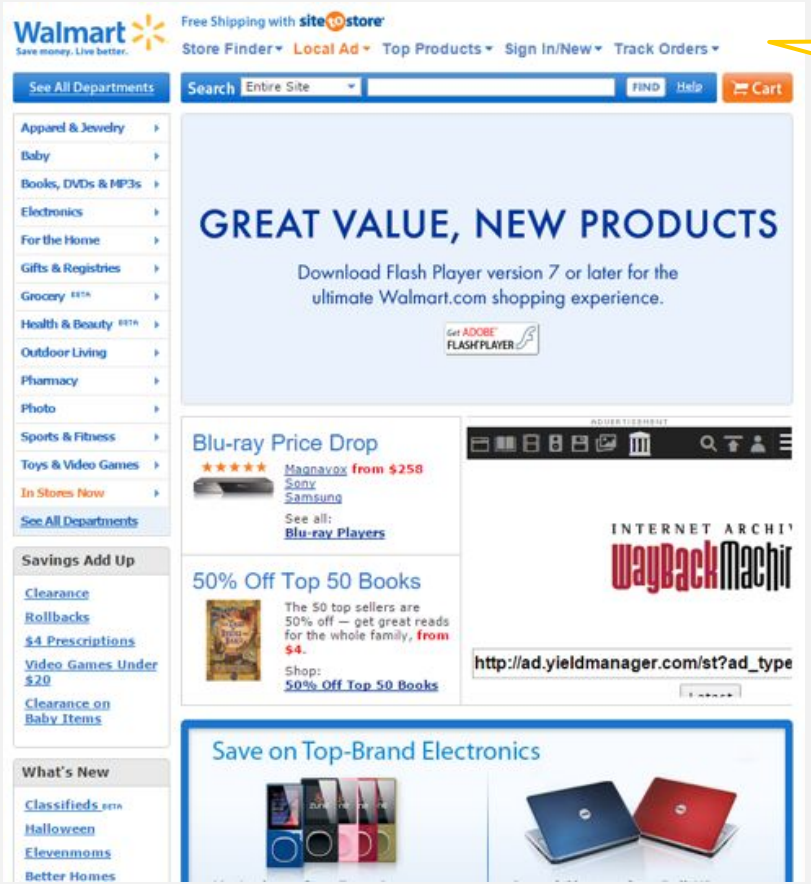
Registration

Categories

Products

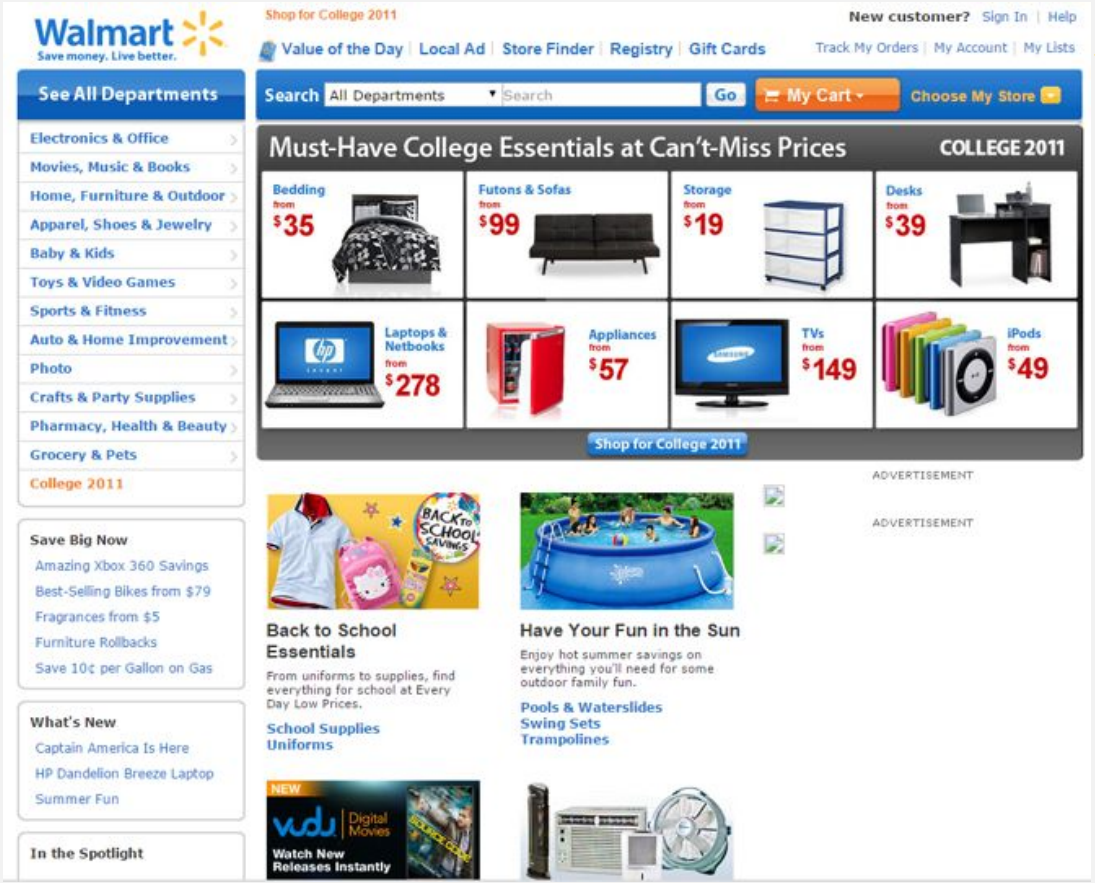
Services

2008



Additional services

2011



Additional services (partly personalized)

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."

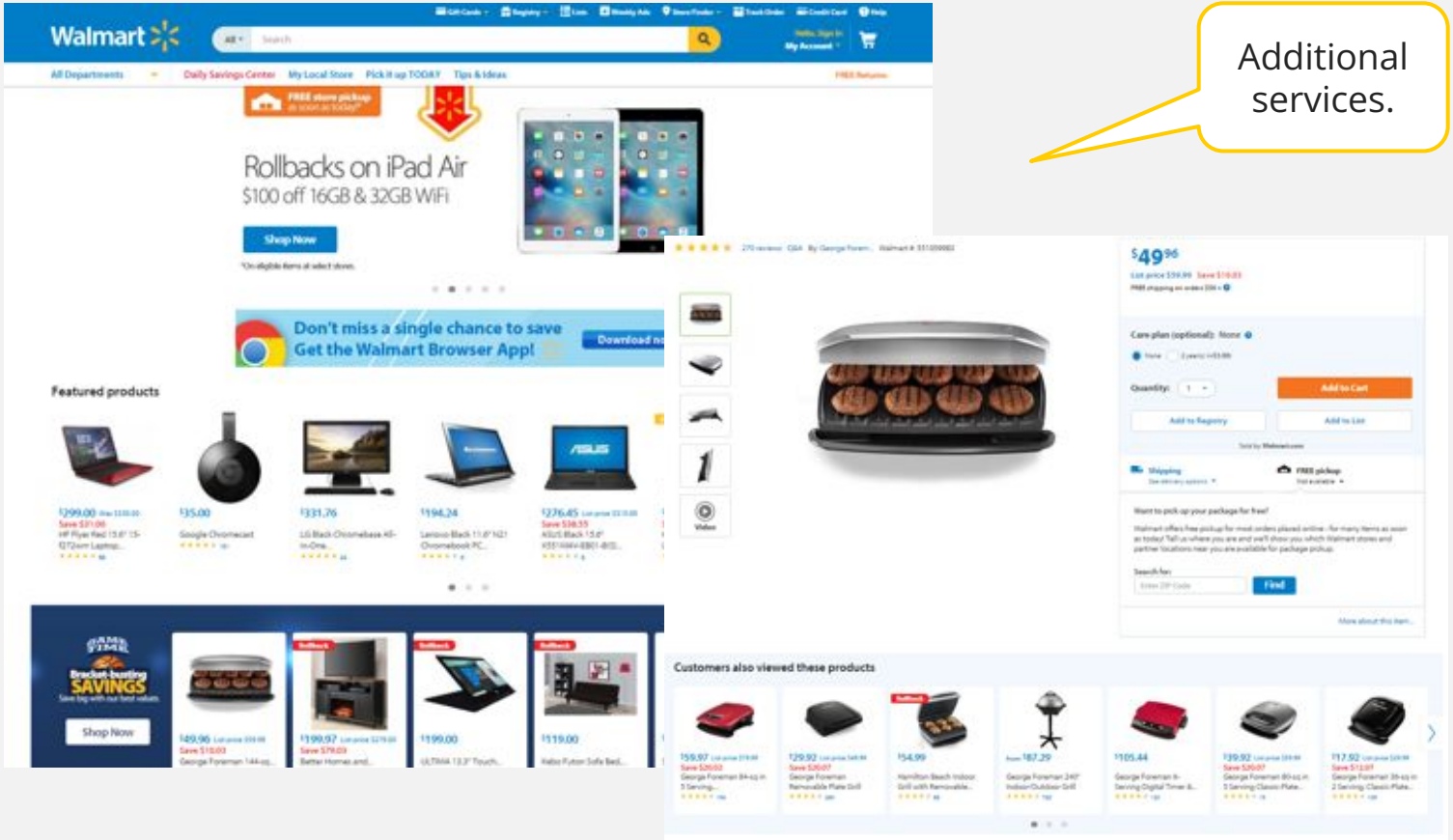
https://blog.risingstack.com/how-enterprises-benefit-from-microservices-architectures

Long cycles from dev to prod.

Lack of scalability.

Lack of elasticity.

2016


Walmart
Rollbacks on iPad Air
$100 off 16GB & 32GB WiFi
Shop Now
Don't miss a single chance to save
Get the Walmart Browser App!
Featured products
Bracket-busting SAVINGS
Shop Now
Customers also viewed these products
Additional services.

Good News

1000 deployments per day …

… by the developers

~ 100% availability

New business models, applications and devices (IoT, mobile, APIs)

Resource efficiency

Elastic scalability

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."

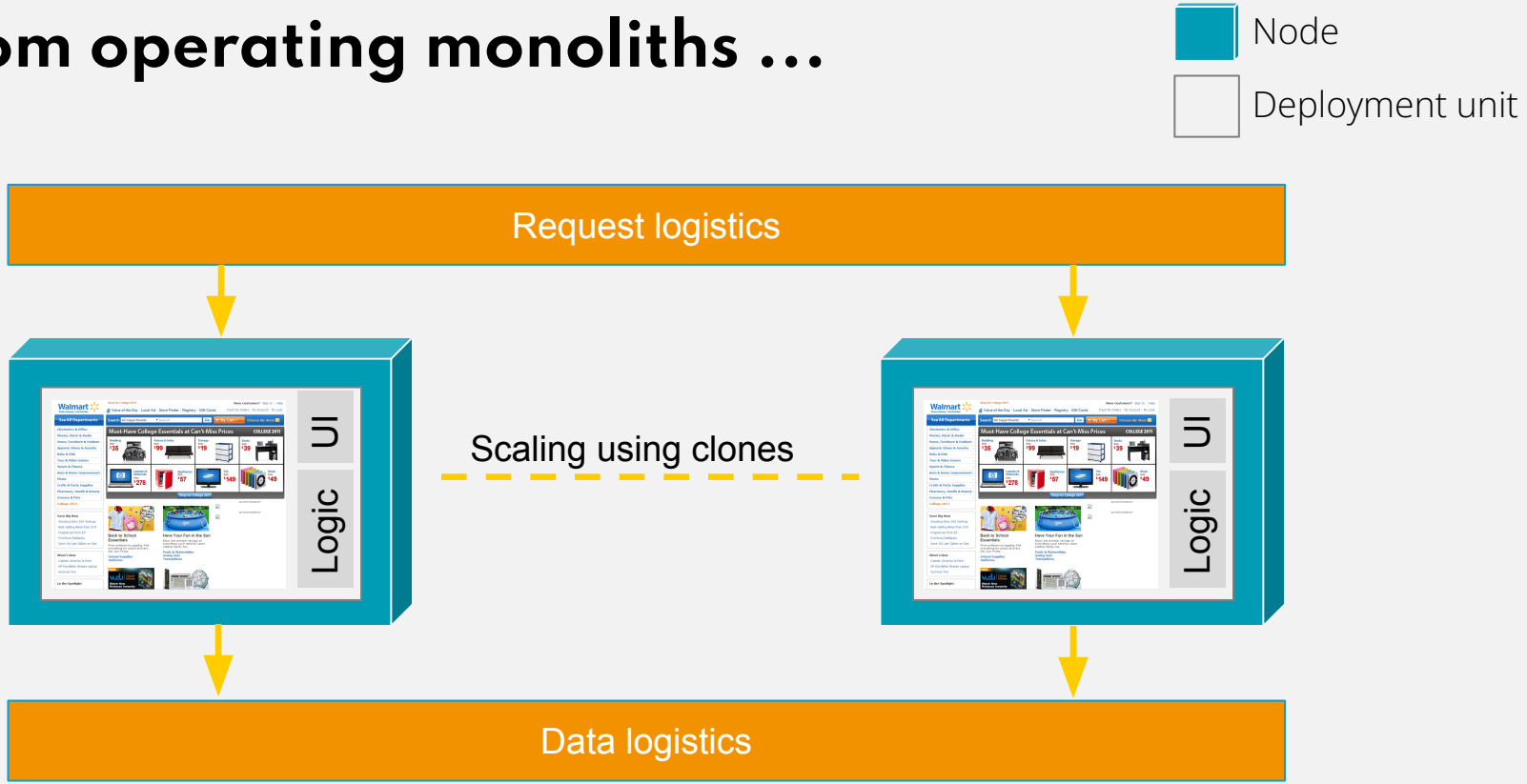
http://www.baselinemag.com/enterprise-apps/walmart-embraces-microservices-to-get-more-agile.html

- "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."

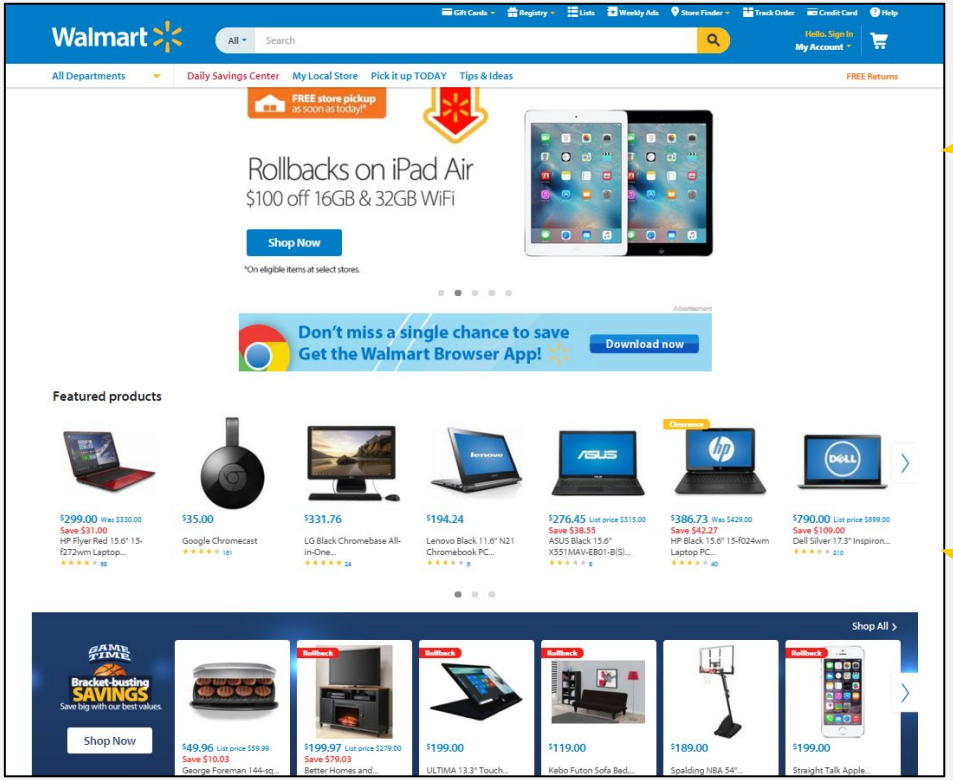
http://techcrunch.com/2014/12/02/walmart-com-reports-biggest-cyber-monday-in-history-mobile-traffic-at-70-over-the-holidays

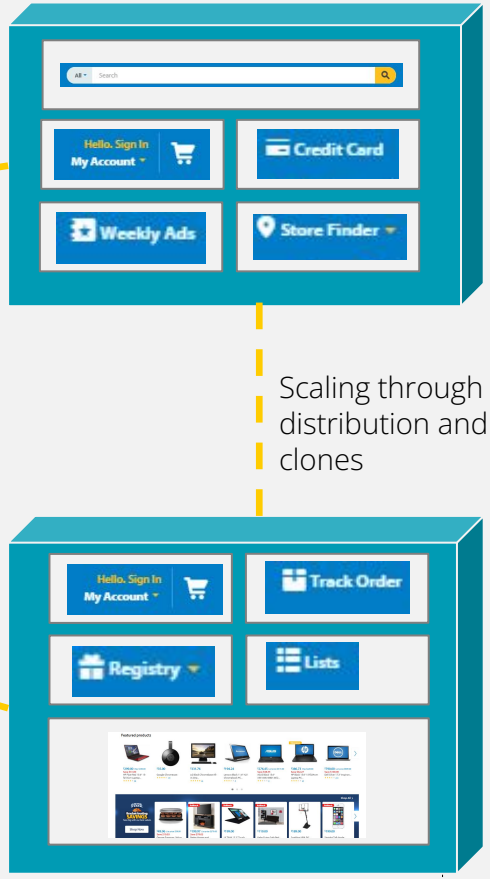
WHAT DID THEY DO?

From operating monoliths …



… to operating components





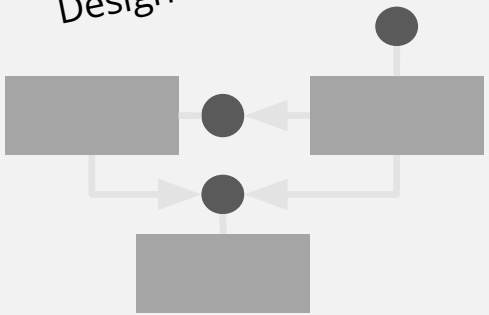
Cloud Native Application Development: Components All Along the Software Lifecycle

QA|WARE

DESIGN → BUILD → RUN

NEW!

Design Components

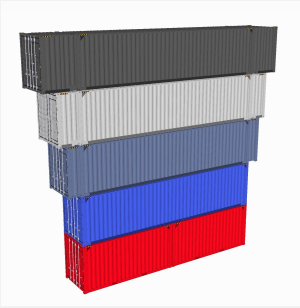


1:1

Dev Components

n:1

Ops Components



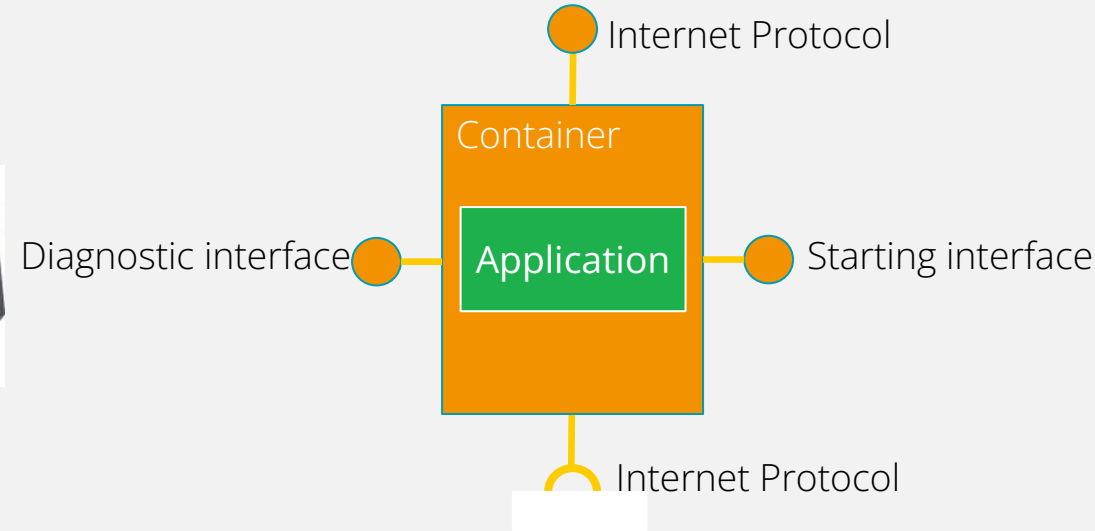
- Complexity unit
- Data integrity unit
- Coherent and cohesive features unit
- Decoupled unit

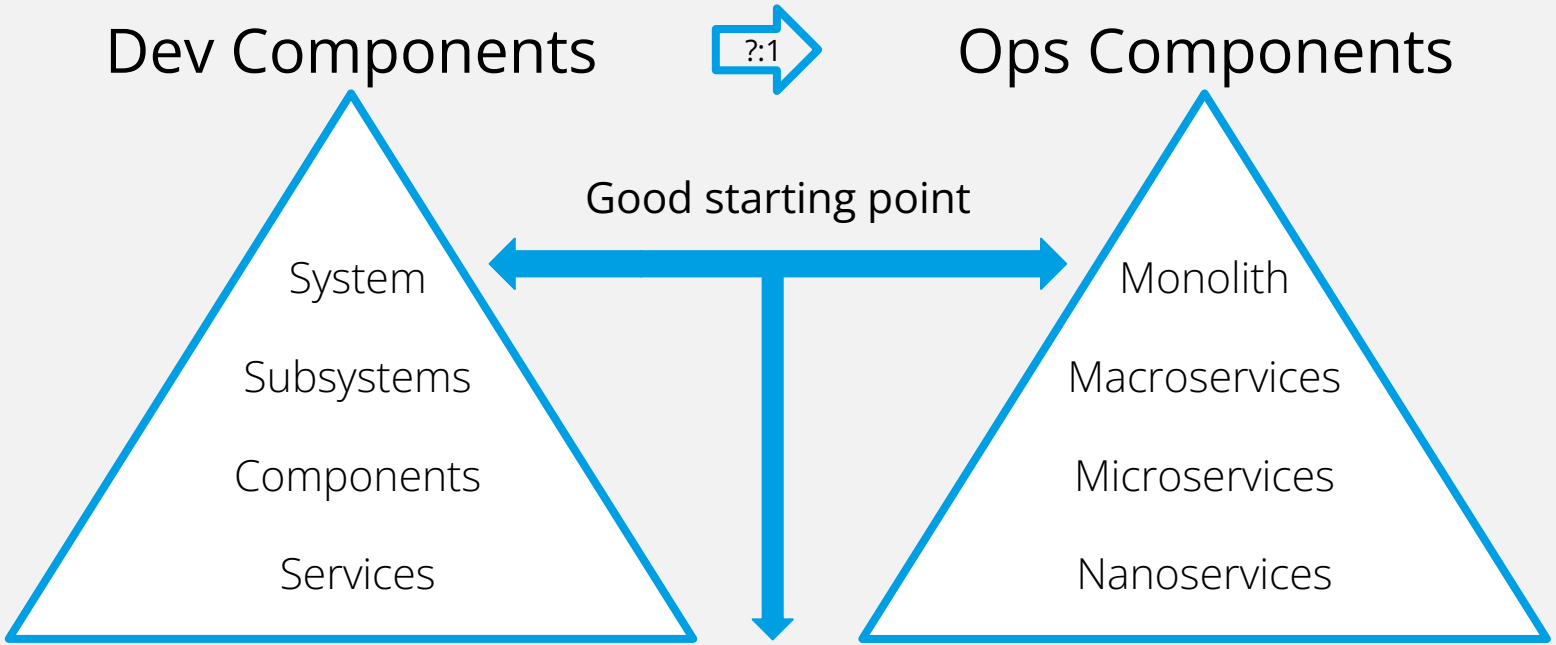
- Planning unit
- Team assignment unit
- Knowledge unit
- Development unit
- Integration unit

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

The anatomy of an operation component

QA|WARE





QA|WARE

Decomposition Trade-Offs

+ More flexible to scale	− Distribution debt: Latency
+ Runtime isolation (crash, slow-down, ...)	− Increasing infrastructure complexity
+ Independent releases, deployments, teams	− Increasing troubleshooting complexity
+ Higher utilization possible	− Increasing integration complexity

Rule #1 for operating services in the cloud

QA|WARE

"Everything fails all the time."

— Werner Vogels, CTO of Amazon

Rule #2 for operating services in the cloud

QA|WARE

If you want the sky to be the limit, then only horizontal scaling works.

Rule #3 for operating services in the cloud

If you want to be in the cloud, you should speak cloud.

TCP

HTTP

DHCP

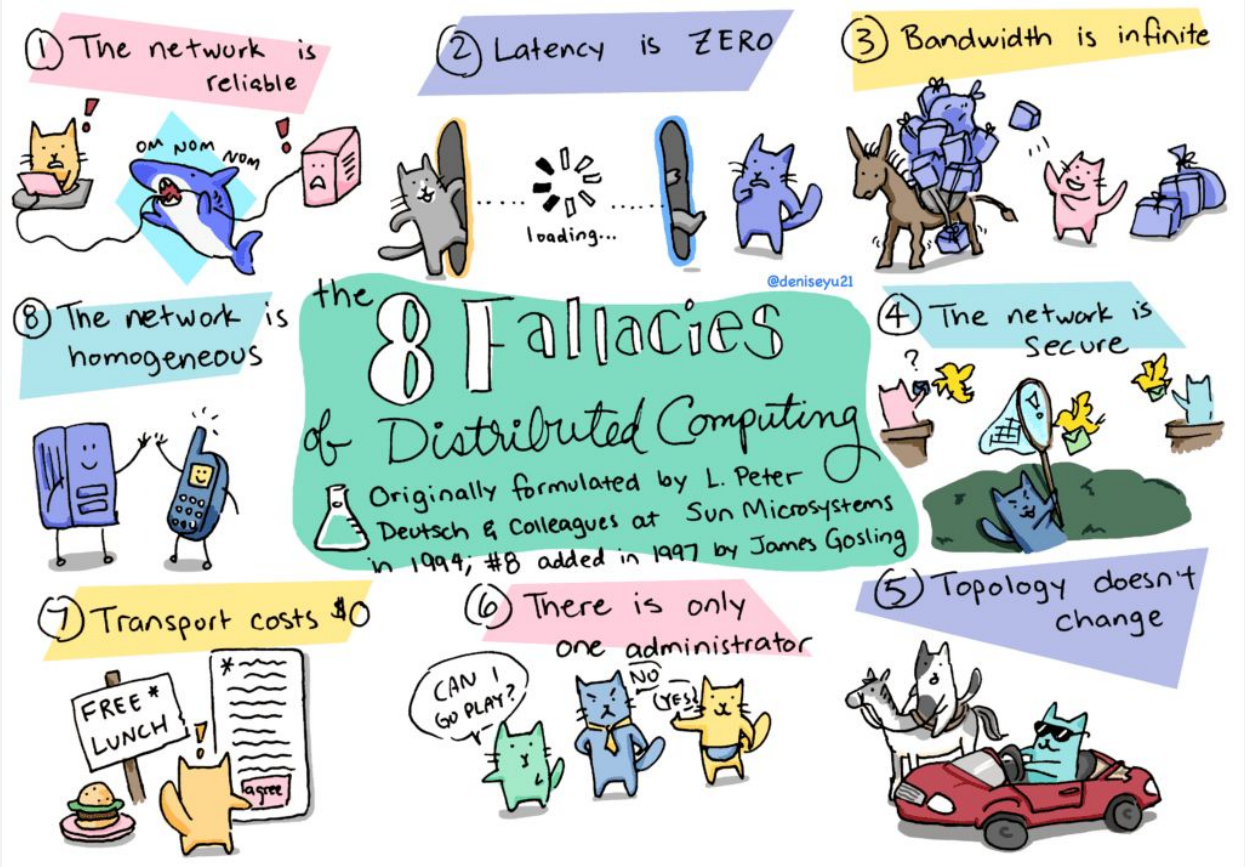
DNS

The 5 Commandments of the 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

Source: https://de.wikipedia.org/wiki/Zehn_Gebote

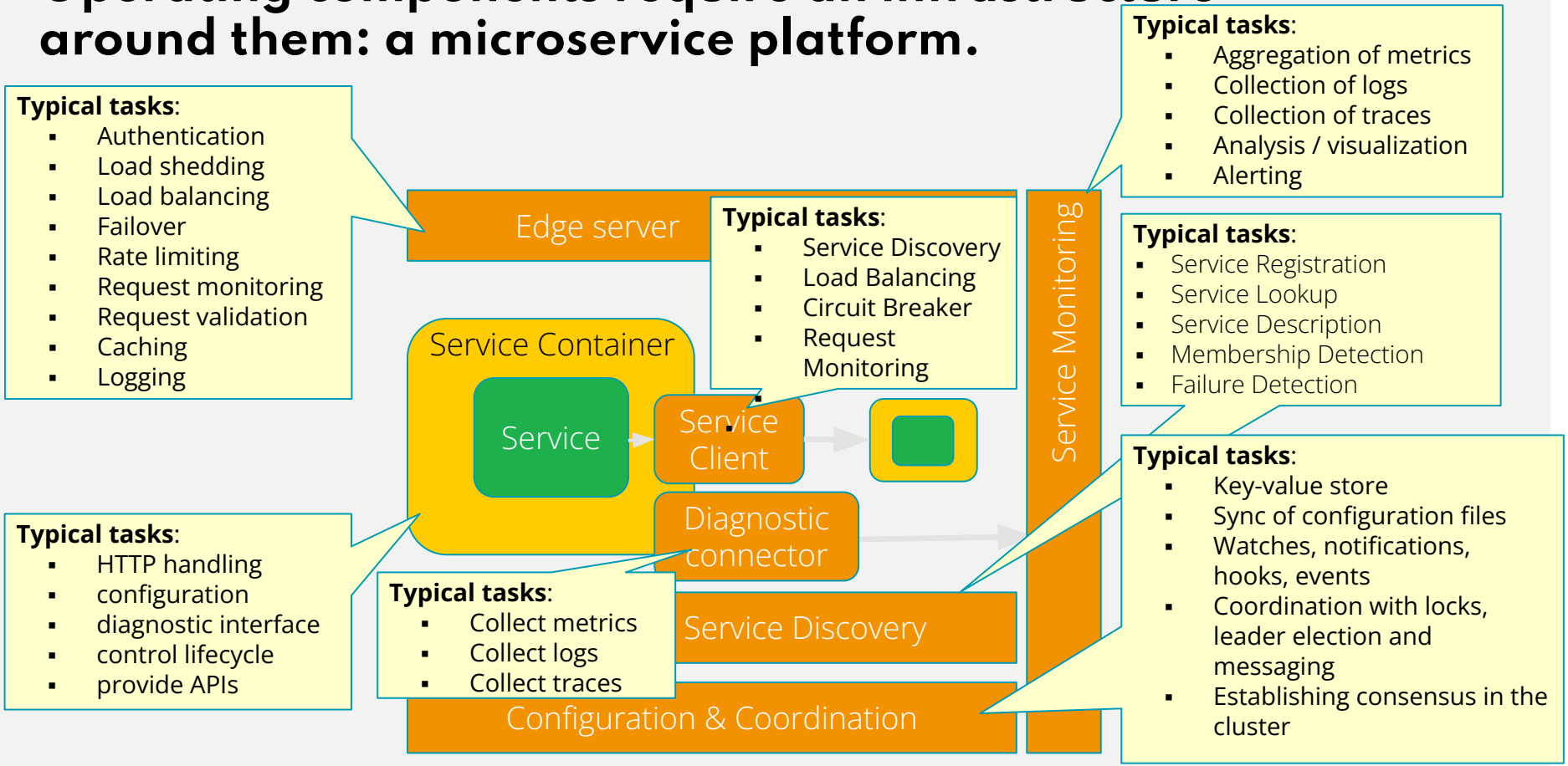
Eight fallacies of distributed computing



Cloud architecture from a software architecture perspective: Design for Failure.

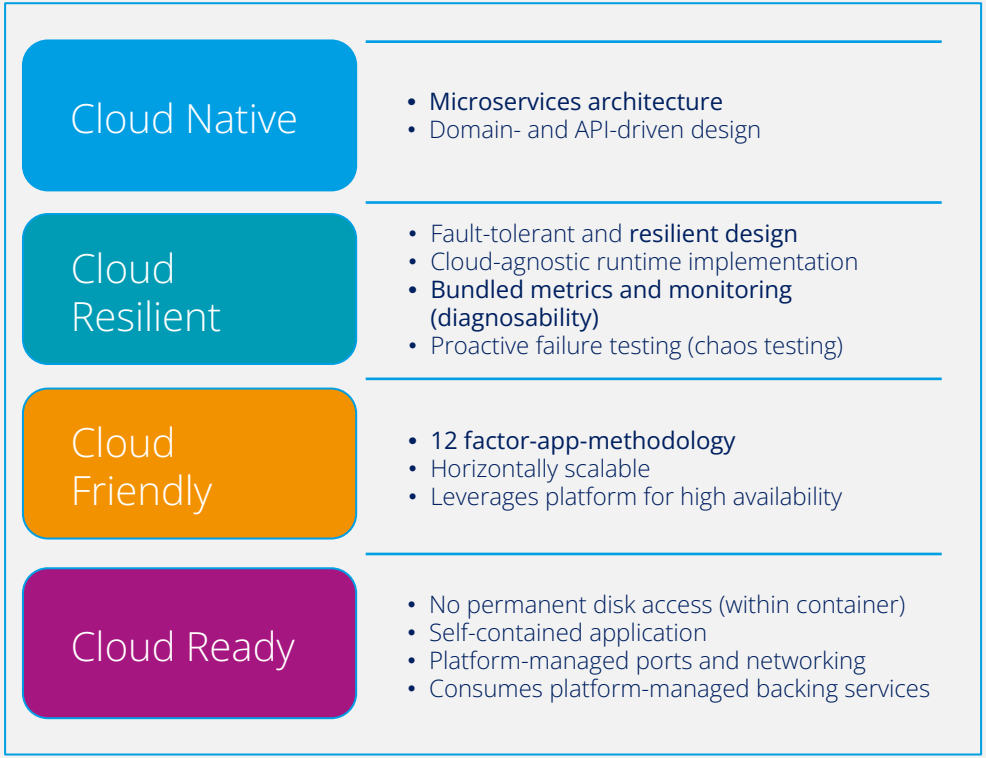
- Each component runs independently and in isolation.
- The operating components communicate with each other via internet protocols HTTP, UDP, …
- Each operating component can run in multiple instances, thus providing redundancy. There is no “common point of failure”. Cluster Orchestrator
- Each operating component has diagnostic interfaces to detect faulty behavior.
- Each microservice can be restarted at any time and deployed to another node. It has no state of its own.
- The implementation behind each microservice can be replaced without the users noticing.

Operating components require an infrastructure around them: a microservice platform.



The Cloud Native Application Maturity Model

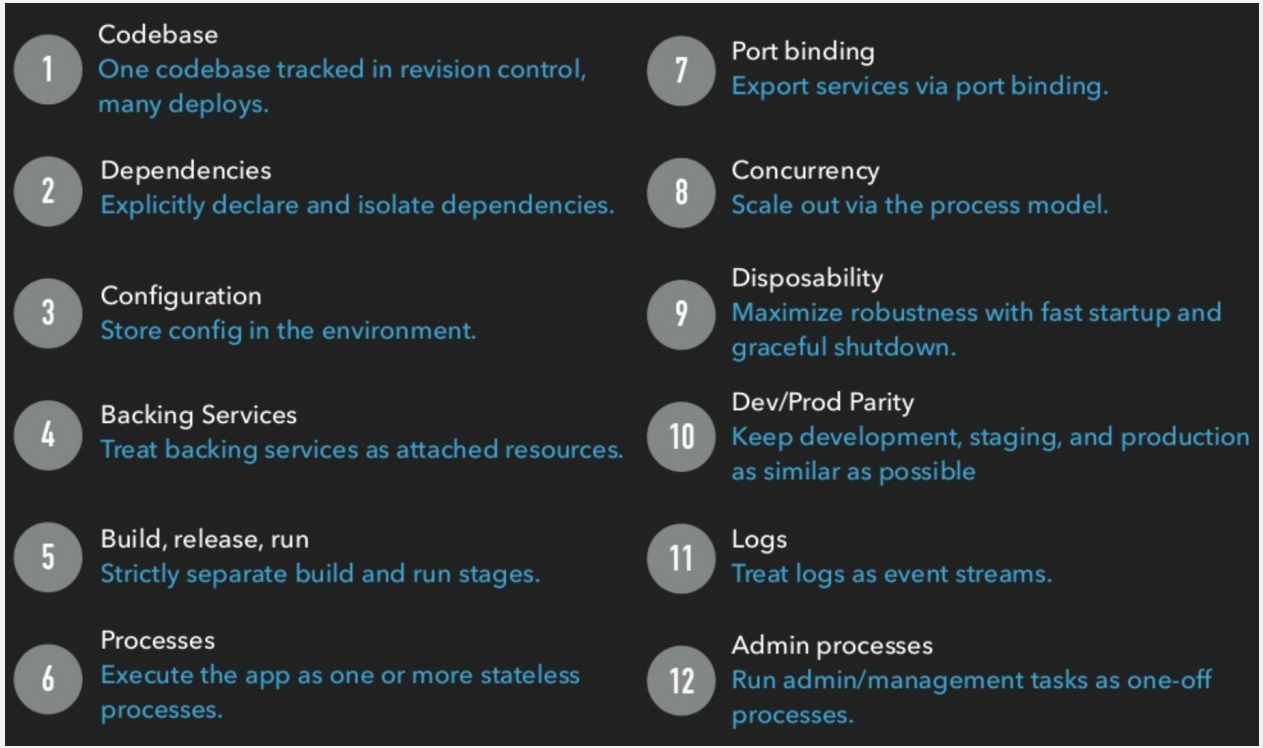
QA|WARE



Exercise 1 (20 min): Twelve Factor Apps

12 Factor App

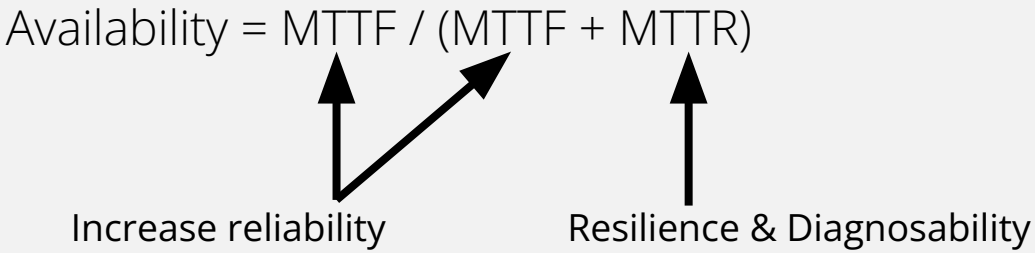
QA|WARE



https://12factor.net/de/
https://www.slideshare.net/Alicanakku1/12-factor-apps

Resilience

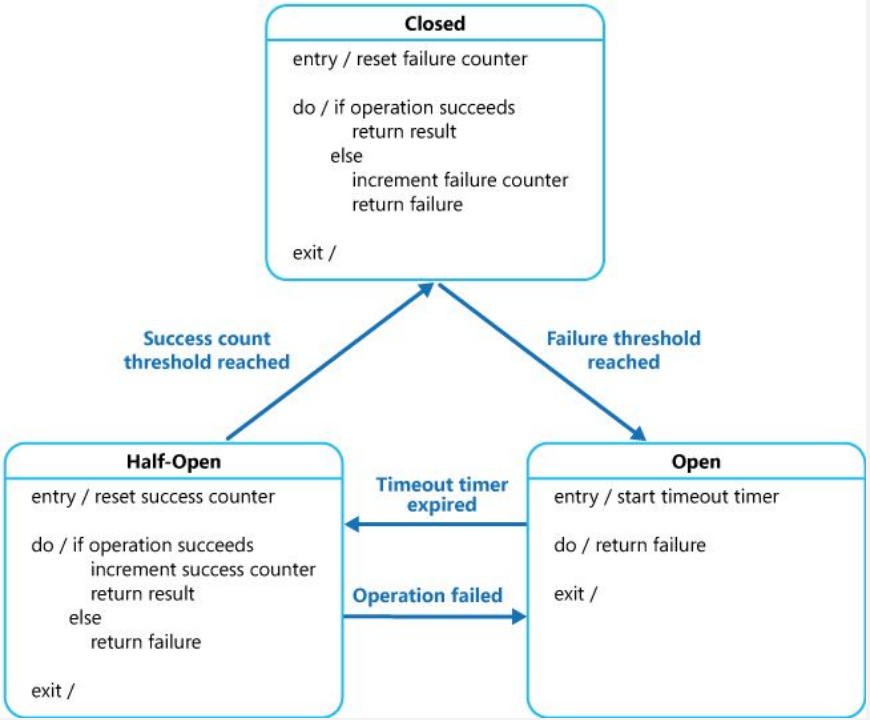
Resilience



Resilience: The ability of a system to deal with unexpected and faulty situations

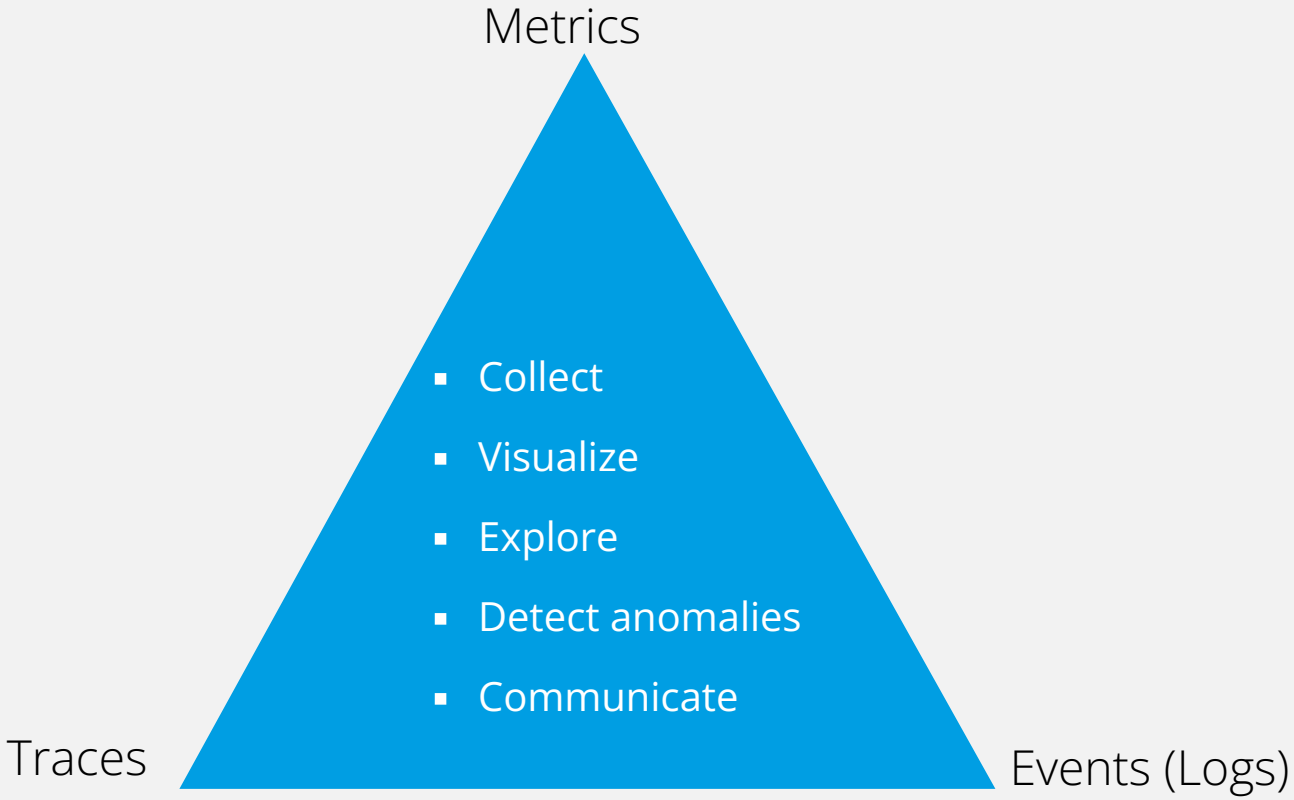
- Without the user noticing (best case)
- With a “graceful degradation” of the service (worst case)

Resilience pattern: circuit breaker



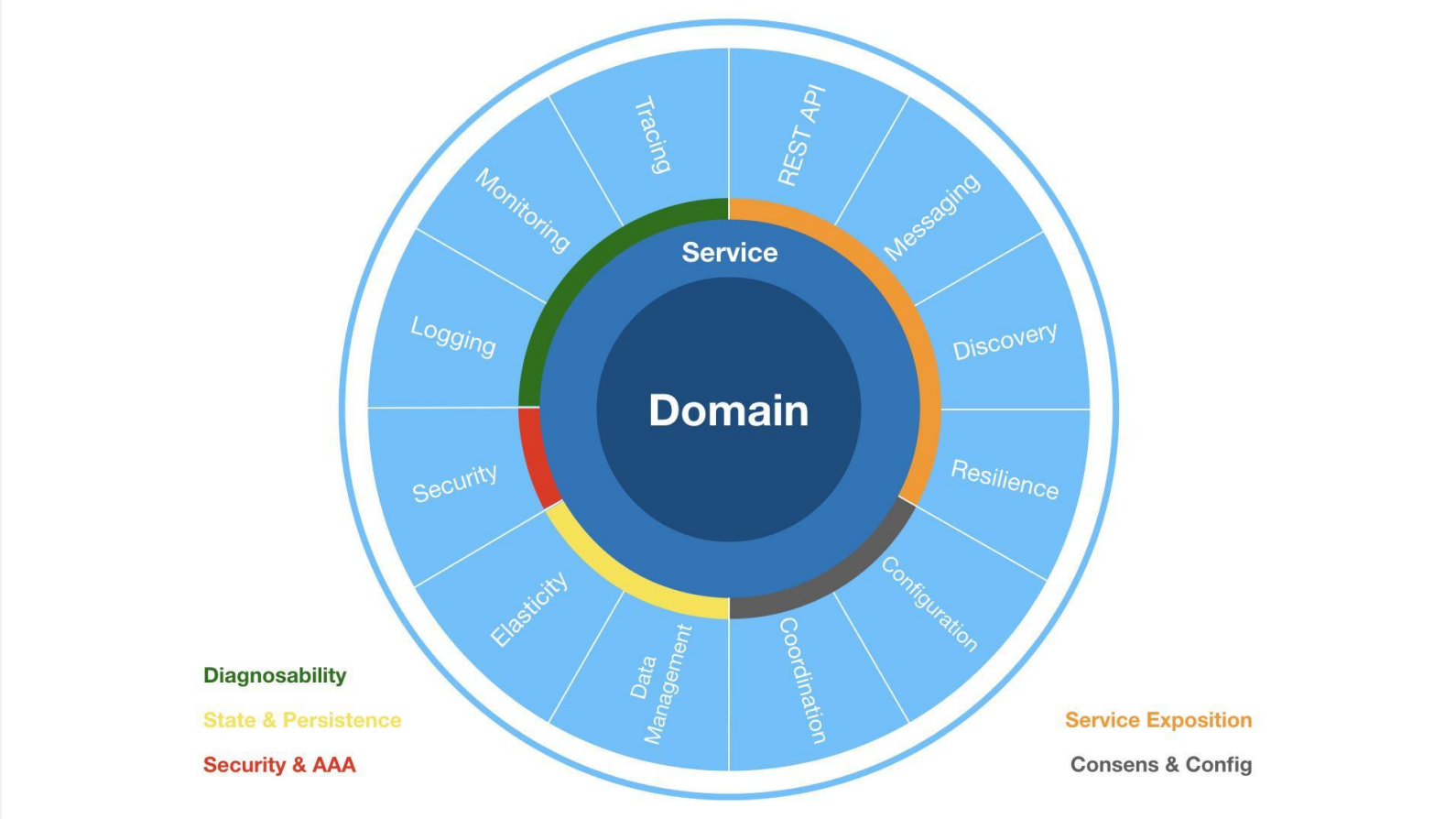
Source: https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker
Further patterns: https://learn.microsoft.com/en-us/azure/well-architected/reliability/design-patterns

Diagnosability

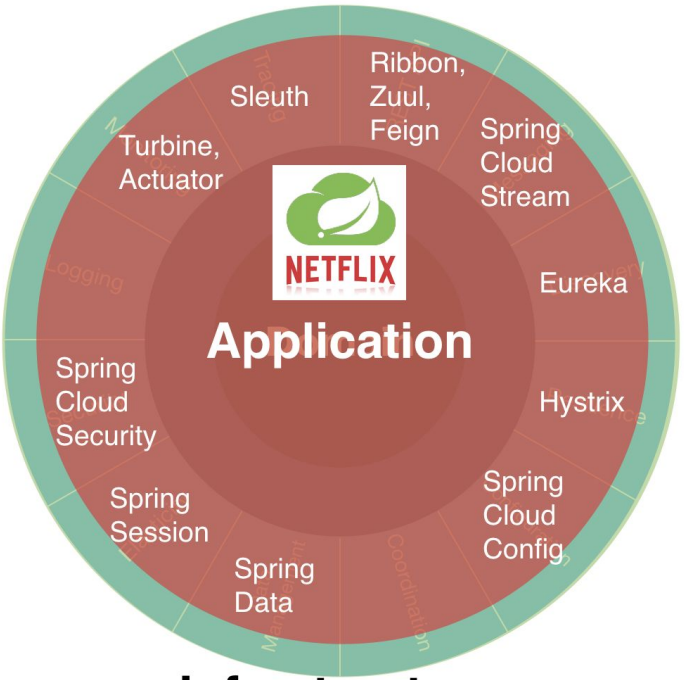


Technical aspects of microservices

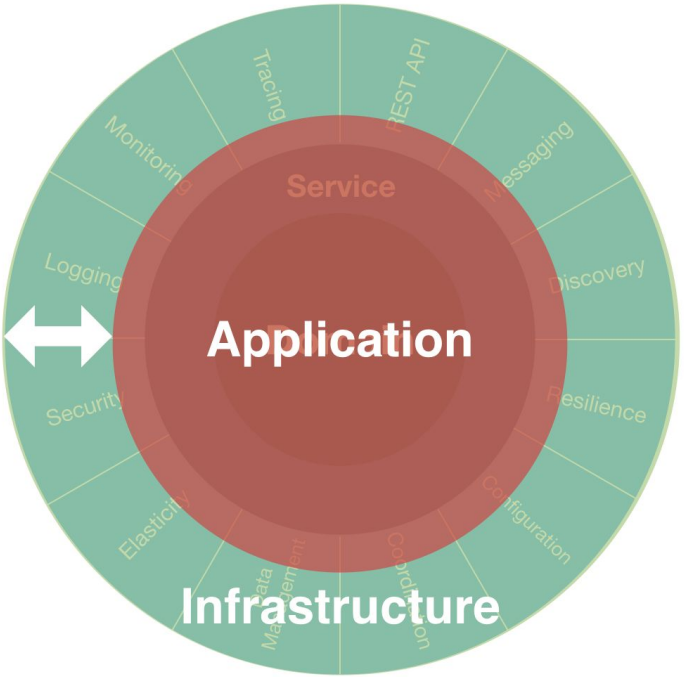
Technical aspects of microservices



Technical aspects of microservices: The library solution

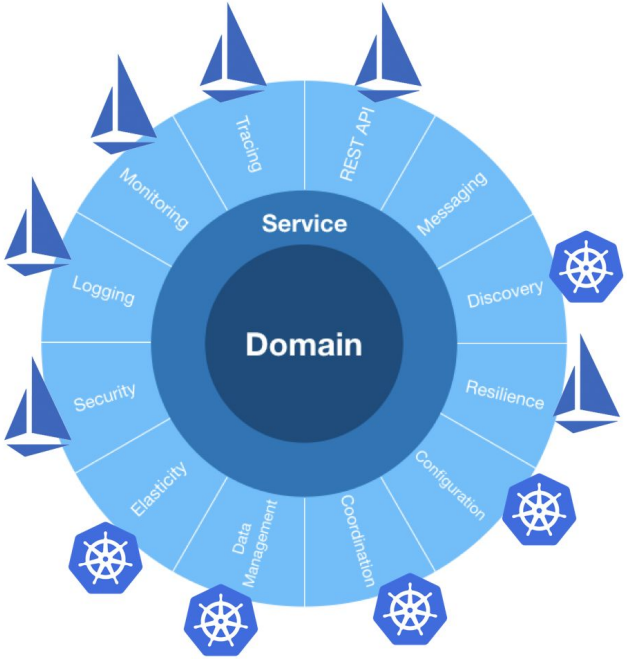


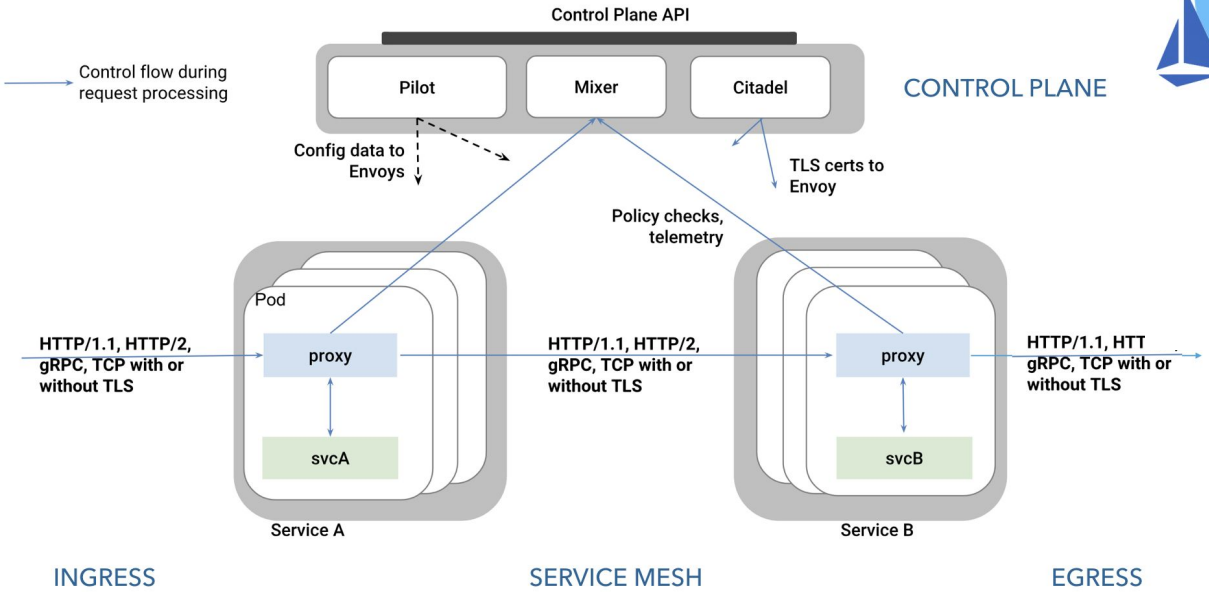
Technical aspects of microservices: The infrastructure solution

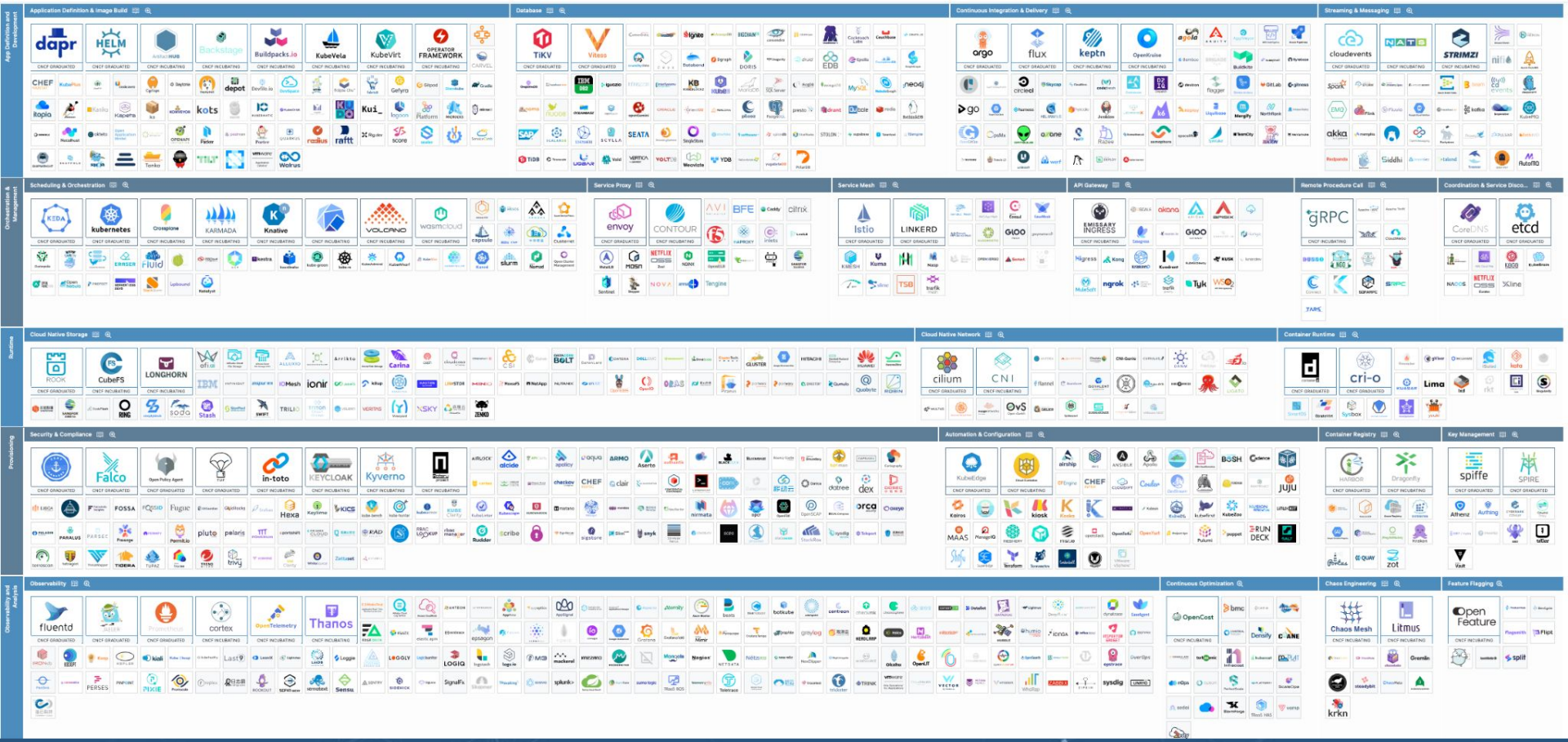


Istio Service Mesh

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





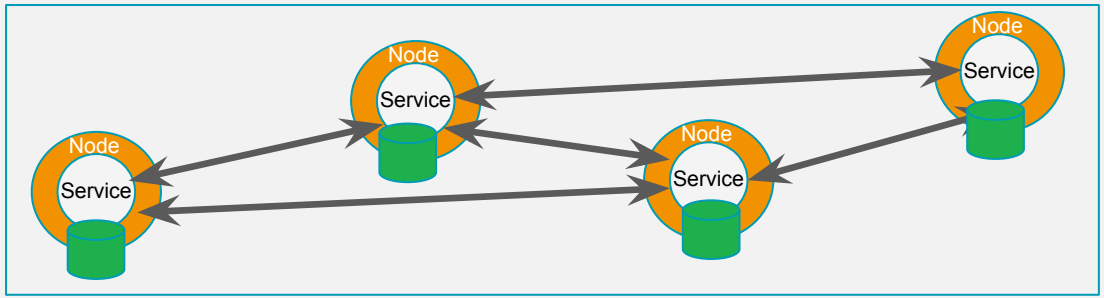


https://landscape.cncf.io/

https://github.com/cncf/landscape

Configuration & Coordination: Distributed State and Consensus

A distributed configuration memory



How is the state of the configuration memory synchronized in the cluster?

The CAP Theorem

Brewer's theorem for properties of stateful distributed systems – now also formally proven. [1]

There are three essential properties that a distributed system can only have two of at a time:

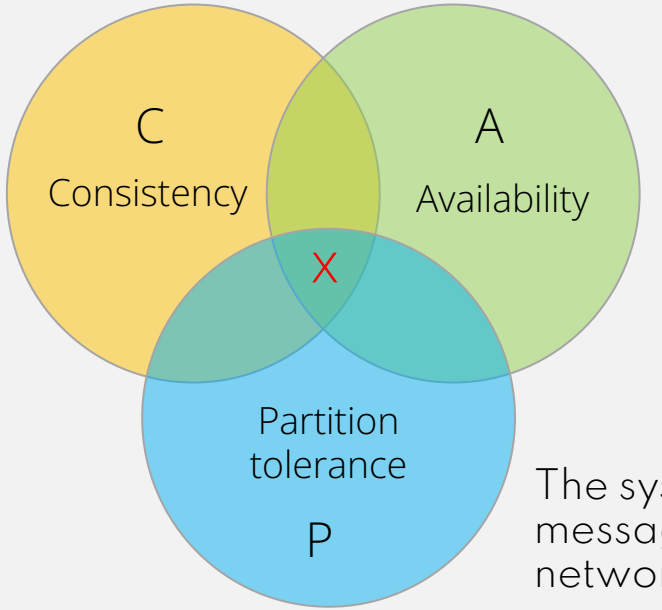
- Consistency **(C)**
- Availability **(A)**
- Partition tolerance **(P)**

[1] Brewer, Eric A. "Towards robust distributed systems." PODC. 2000.

The CAP Theorem

QA|WARE

All nodes see the same data at the same time. All copies are always the same.



The system will continue to operate even if individual nodes fail. Failures of nodes and channels do not prevent the surviving nodes from functioning.

The system also works in the case of lost messages. The system can handle the network being divided into nodes in several partitions that do not communicate with each other.

In the case of a network partition, you have to choose between consistency and availability; you can't have both.

Gossip protocols for high availability

Basis: A network of agents with their own state

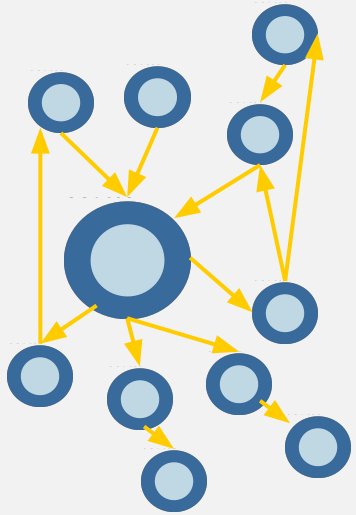
- Agents distribute a gossip stream
- Message: Source, content / state, timestamp
- Messages are sent periodically at fixed intervals
- to a certain number of other nodes (fanout)

Viral spread of the gossip stream

- Nodes that are in a group with me will definitely receive a message
- The top x% of nodes that send me messages will receive a message

Messages that are trusted are adopted into the local state if

- the same message has been heard from several pages
- the message comes from nodes that the agent trusts
- no more current message is available



Gossip protocols for high availability

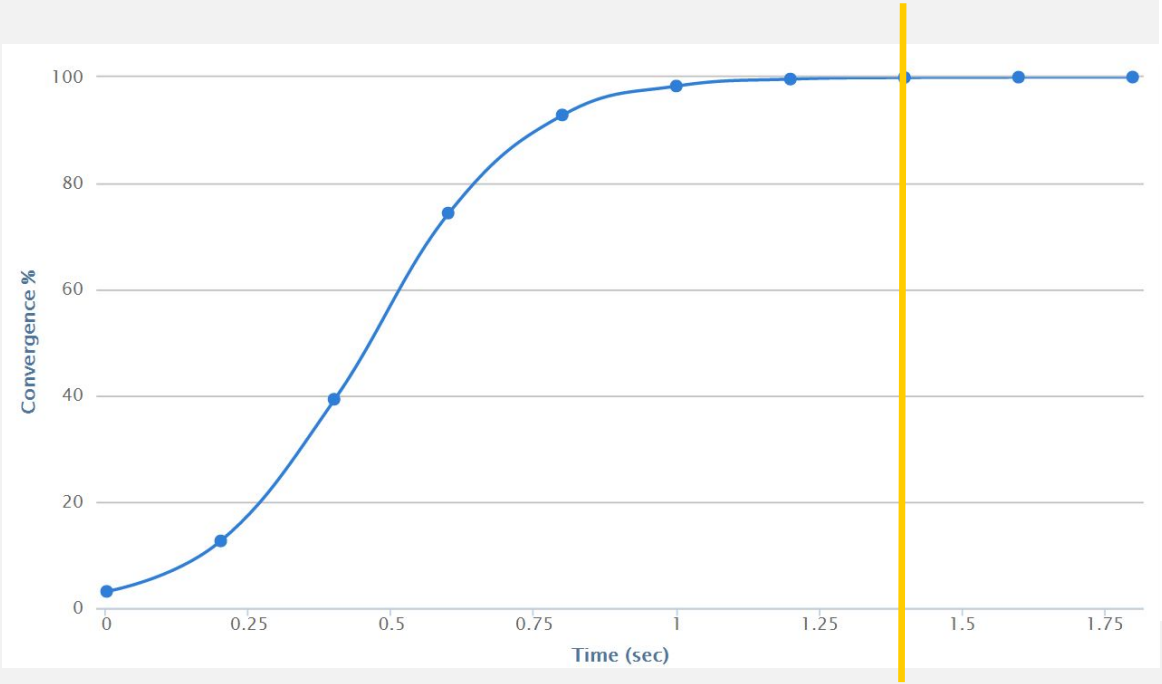
Advantages:

- No centralized units necessary.
- Defective partitions in the network are bypassed. Communication does not have to be reliable.

Disadvantages:

- The state is potentially inconsistently distributed (but converges over time)
- Overhead due to redundant messages.

The convergence of the data and thus the time of complete consistency can be calculated.



Vollständige Konsistenz

Estimated max bandwidth: 328.1 kbps/node

Gossip Interval

The gossip interval controls how often messages are gossiped to other nodes

0.2 seconds

Gossip Fanout

The gossip fanout controls how many nodes we gossip with

3 nodes

Nodes

This controls how many simulated nodes are in the cluster

30

Packet Loss

This controls the amount of simulated packet loss [0, 100)

0 % lost packets

Node failures

This controls what percent of simulated nodes are failed

0 % failed

https://www.serf.io/docs/internals/simulator.html

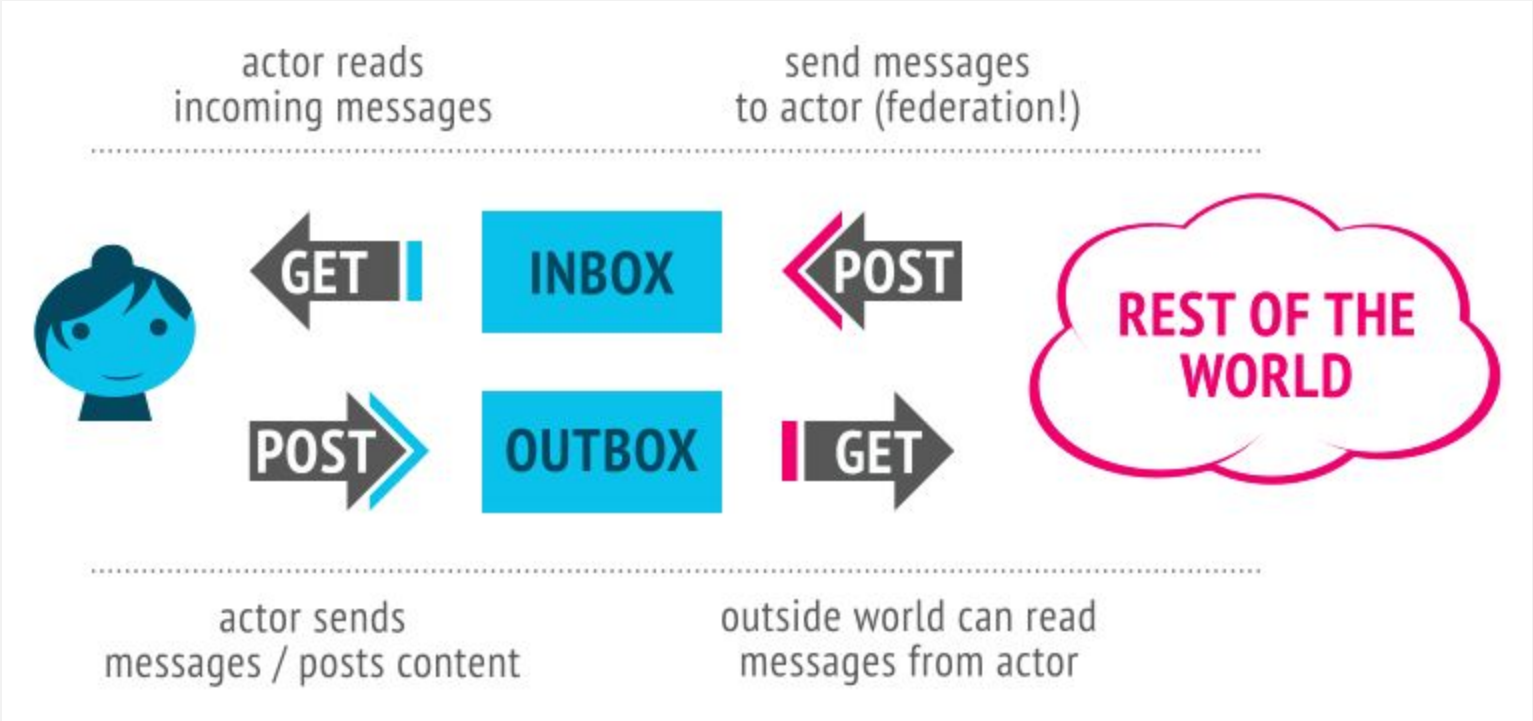
The Fediverse

QA|WARE

https://framatube.org/w/9dRFC6Ya11NCVeYKn8ZhiD

Example for a Gossip protocol: ActivityPub

QA|WARE



Source: https://www.w3.org/TR/activitypub/

Protocols for distributed consensus: consistent, but not highly available, in contrast to gossip protocols

Basis: network of agents

Principle: it is sufficient if the state is consistent on a simple majority of the nodes and the remaining nodes recognize their inconsistency.

Procedure:

- the network agrees on a leader agent by a simple majority – initially and if the leader agent is not accessible. A partition in the minority cannot elect a leader agent.
- All changes are routed through the leader agent. The leader agent periodically distributes change messages to all other agents via multicast.
- If a simple majority of agents acknowledge the change message, the change is activated in the leader and (via message) also in the agents that have acknowledged. Otherwise, the state is assumed to be inconsistent.

Specific consensus protocols: **Raft**, **Paxos**

Protocols for distributed consensus: consistent, but not highly available, in contrast to gossip protocols

Advantages:

- Faulty partitions in the network are tolerated and automatically consistent again after the error has been rectified.
- Strictly consistent data.

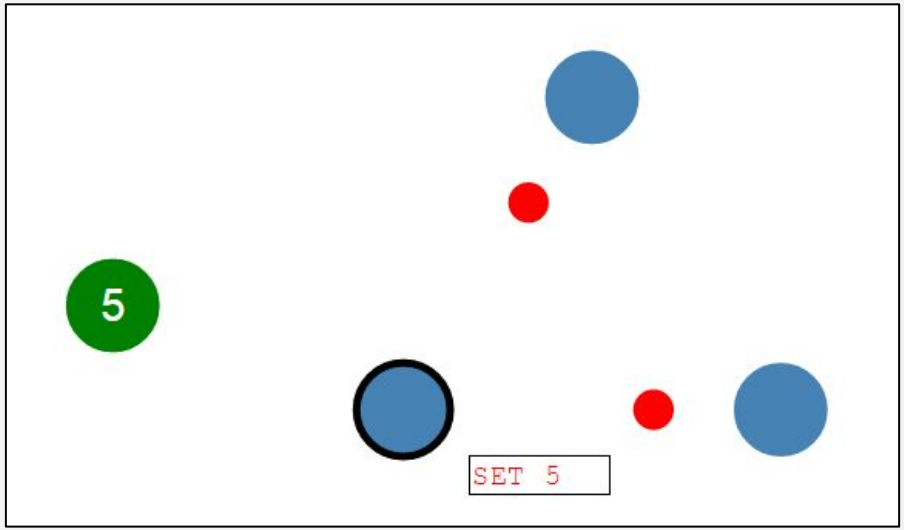
Disadvantages:

- The central leader agent limits the throughput of changes.
- Not highly available: in the case of a network partition, the smaller partition cannot continue to work. If the majority is not in any partition, then overall work cannot continue.

The Raft Consensus Protocol

QA|WARE

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



http://thesecretlivesofdata.com/raft

https://raft.github.io/

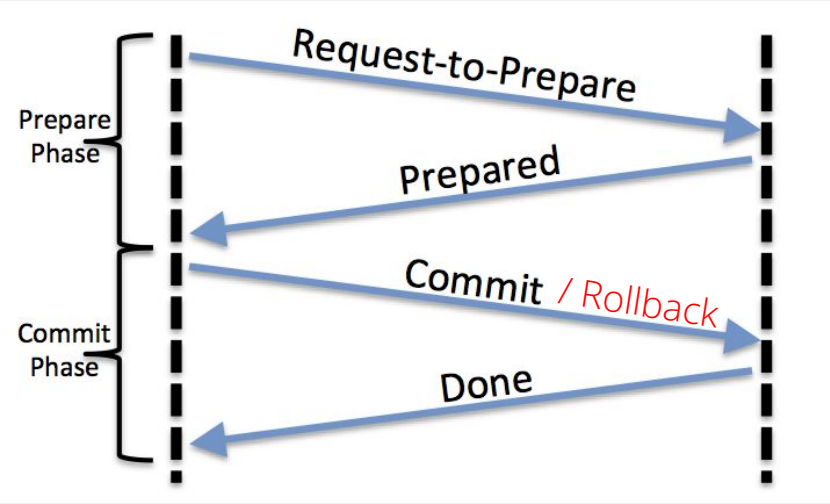
Exercise 2 & 3 (30 min):

The Raft Consensus Protocol
etcd

If strict consistency across all nodes is necessary, the 2-phase commit protocol (2PC) remains.

QA|WARE

A transaction coordinator distributes the changes and activates them only when all have been approved. Otherwise, the changes are undone.



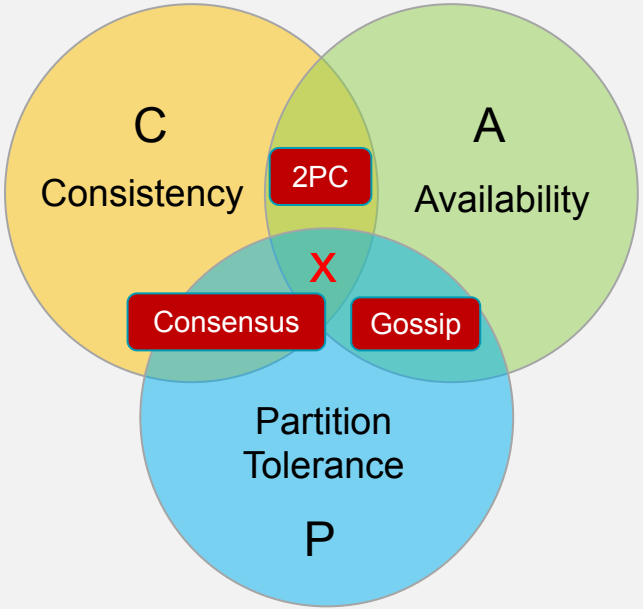
Advantage:

- All nodes are consistent with each other.

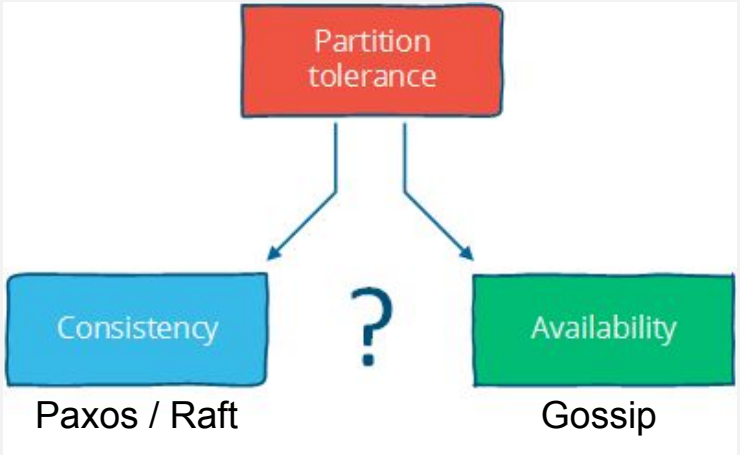
Disadvantages:

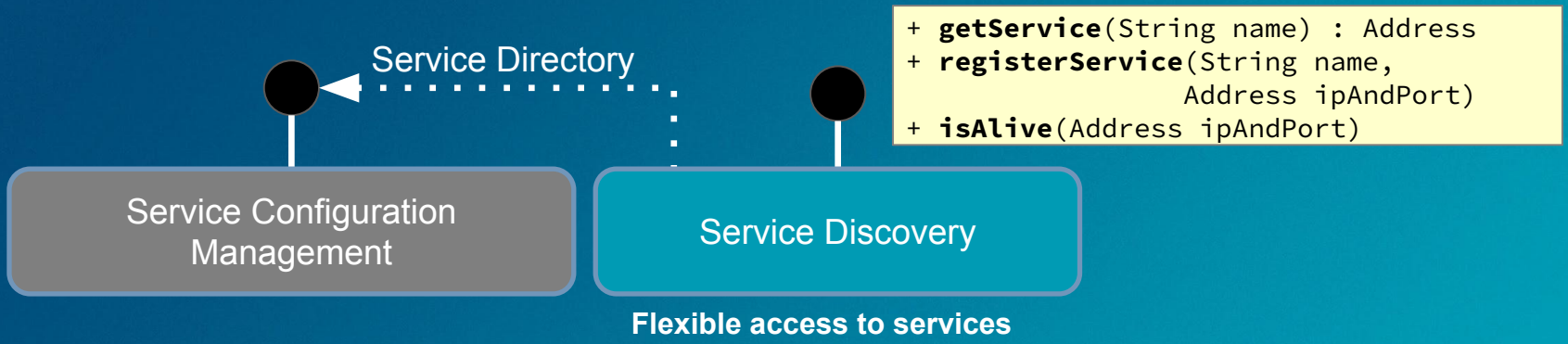
- Time-consuming, since all nodes must always agree.
- The system no longer works once the network is partitioned.

The presented protocols and the CAP theorem



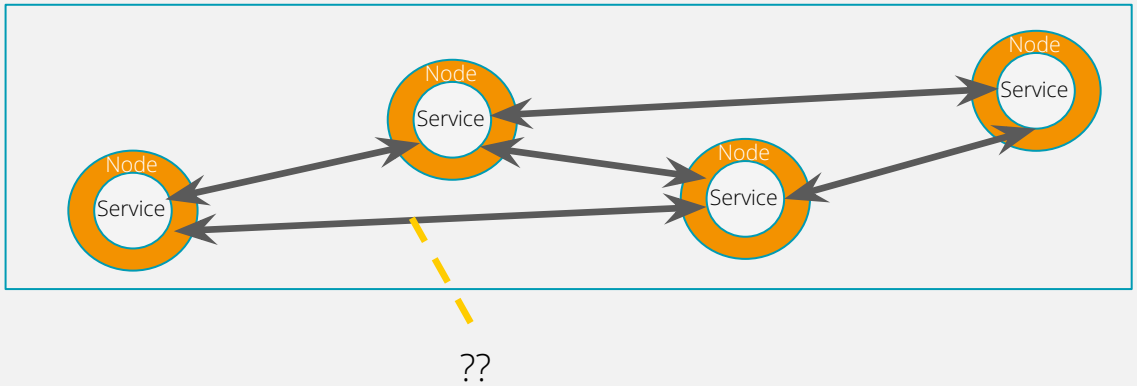
In the cloud, partitions have to be assumed.
This makes the decision binary:
Between consistency and availability.





Service Discovery

The problems of a classic linking of services in the cloud



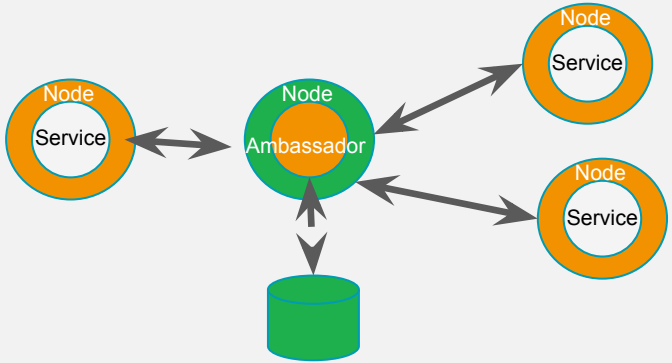
Problems:

- Lack of redundancy: Each service is used directly. It cannot run in multiple instances to provide redundancy.
- Lack of flexibility: The services cannot be restarted or deployed on a different node without side effects – or even replaced with a different service implementation.

Solutions:

- Dynamic DNS
- Ambassador
- Dynamic configuration files
- and environment variables

The Ambassador Pattern



One ambassador node for each node type (e.g. web server)

Service Registration:

- Monitors the cluster and recognizes new and sick/dead nodes in its group.
- Stores the currently active nodes in the configuration memory.

Service Discovery: The client communicates with the ambassador node, which then passes the requests on to a node in the group as efficiently as possible.

The ambassador node can provide a range of additional services when connecting to the service (service binding):

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

Exercise 4:
Consul & Traefik