

Cloud Architektur

24.11.2021



WHAT DID THEY DO?



Betriebsmonolithen



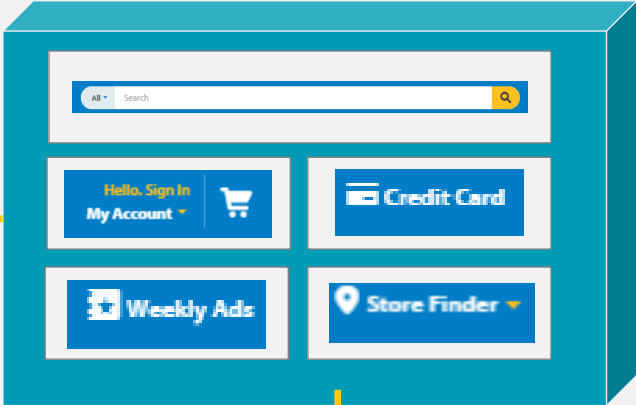
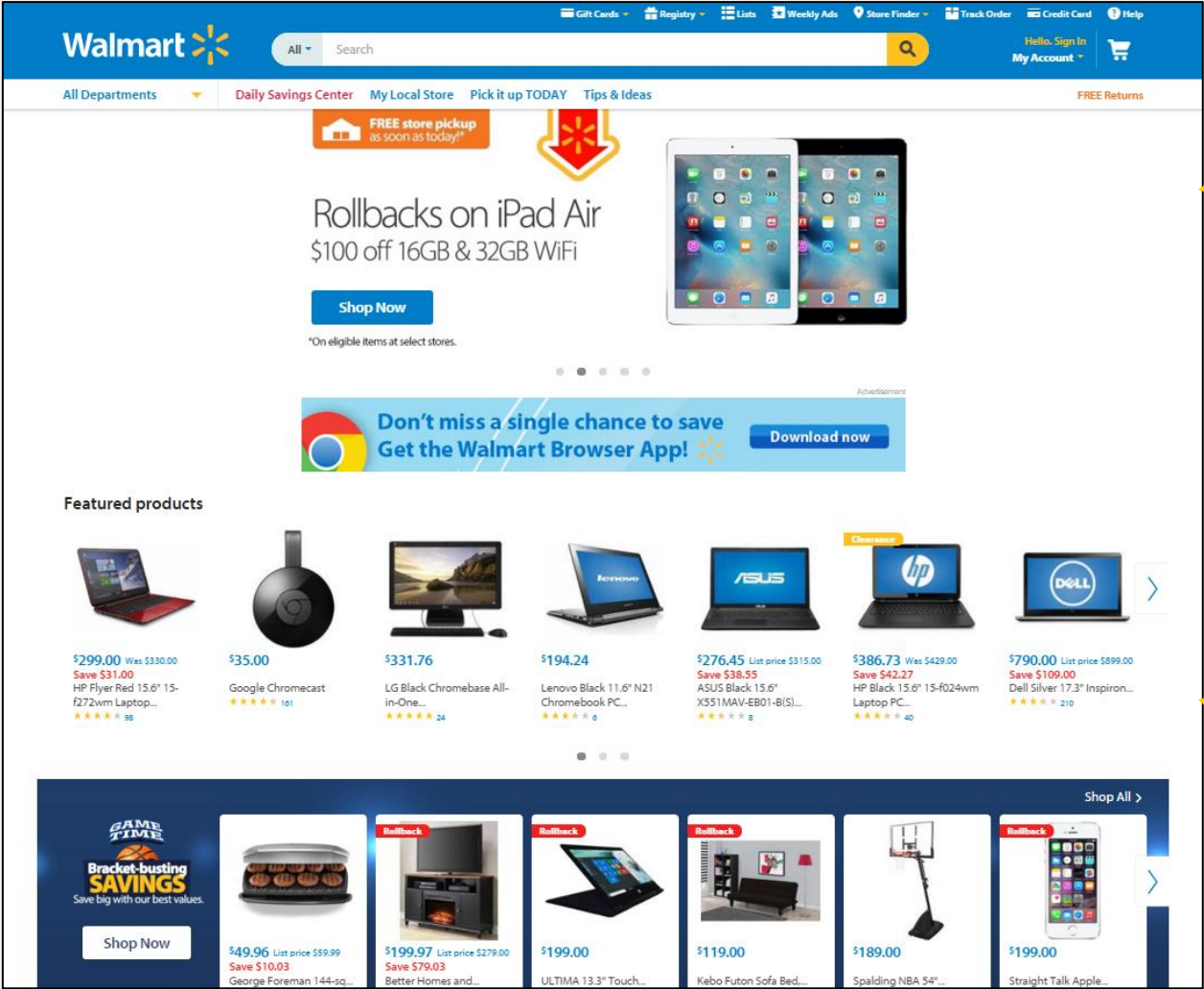
Knoten



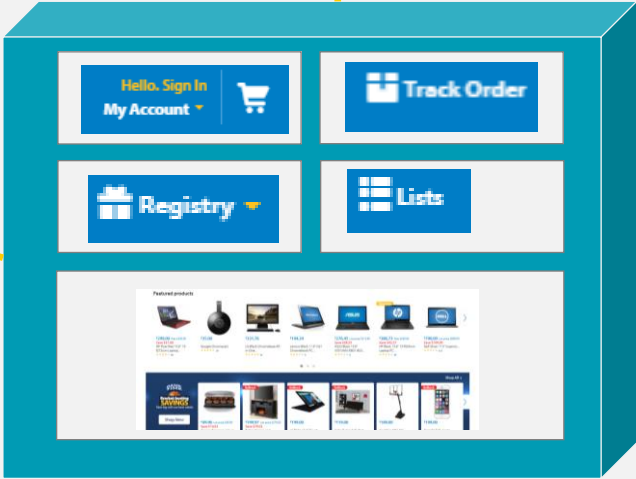
Deployment-Einheit



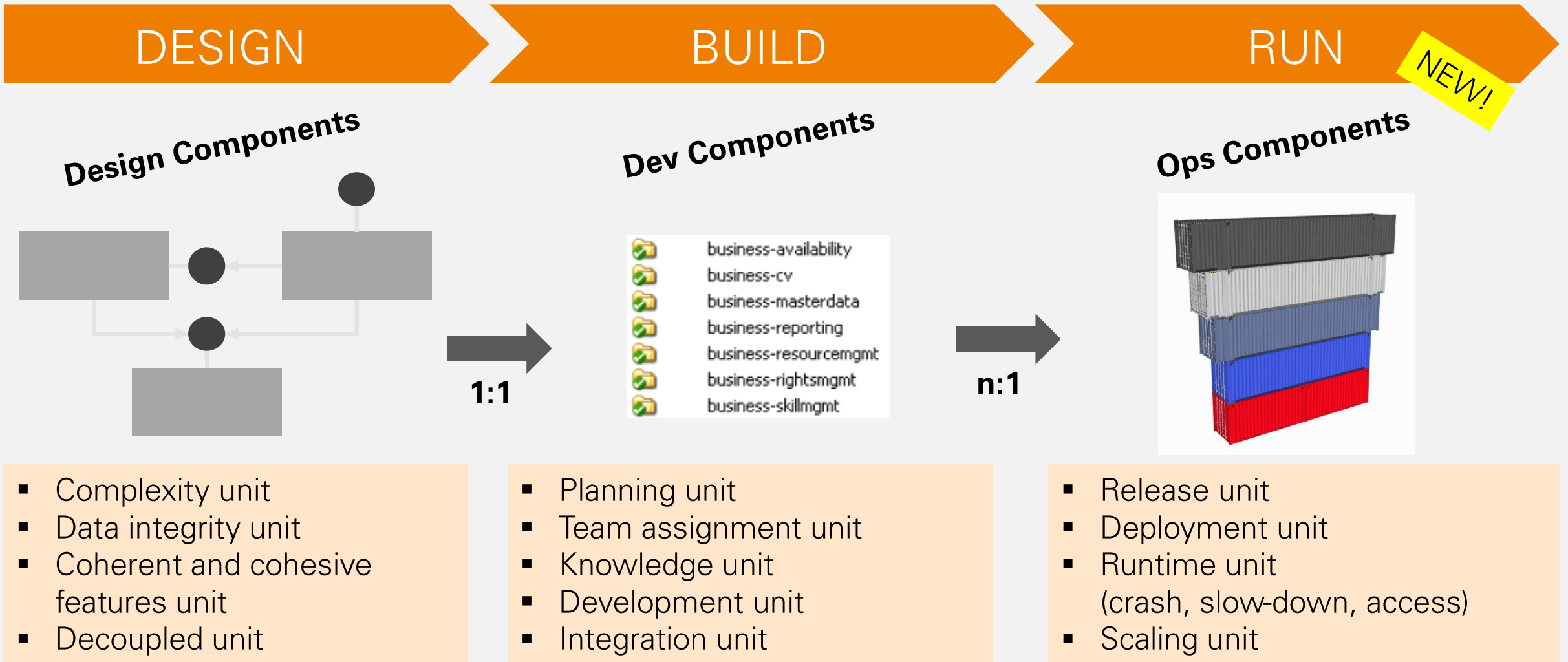
Betriebskomponenten



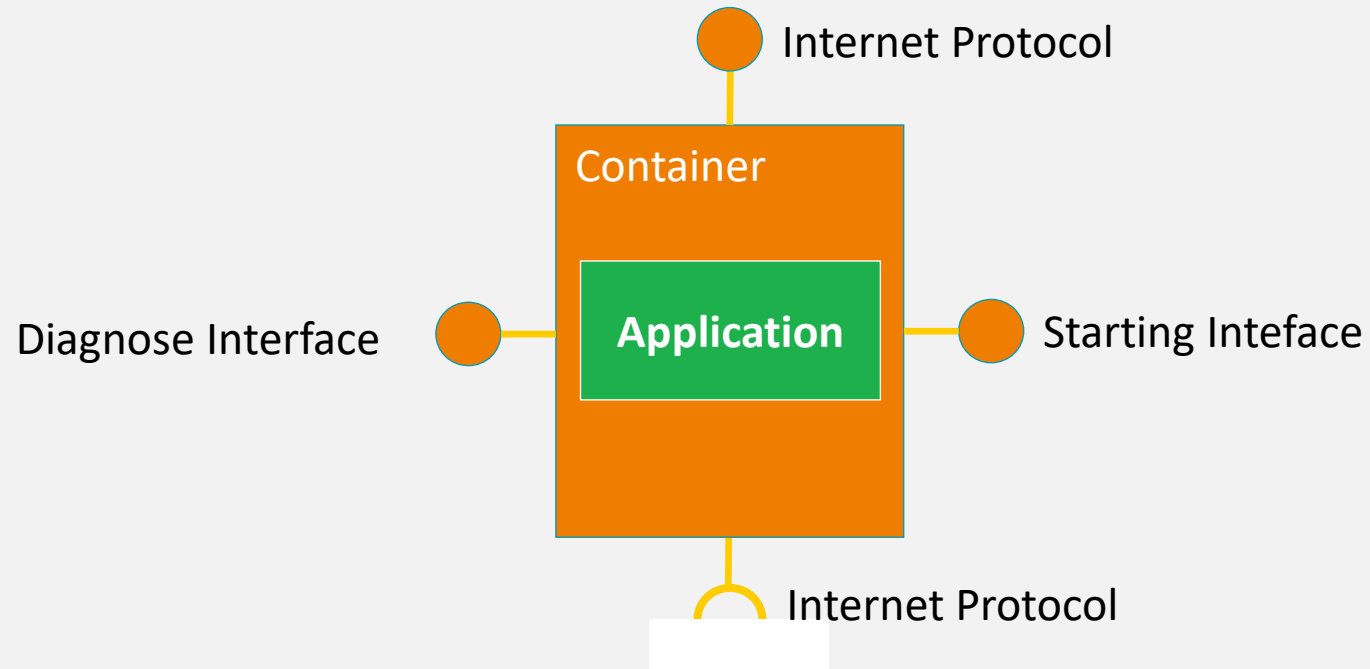
Skalieren durch
Verteilung und Klone



Cloud Native Application Development: Components All Along the Software Lifecycle.



Die Anatomie einer Betriebs-Komponente.



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.

TCP

HTTP

DHCP

DNS

Cloud-Architektur aus Sicht der Softwarearchitektur:

Design for Failure.

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

Betriebskomponenten benötigen eine Infrastruktur um 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

Typische Aufgaben:

- Metriken sammeln
- Logs sammeln
- Trace sammeln

Typische Aufgaben:

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

Typische Aufgaben:

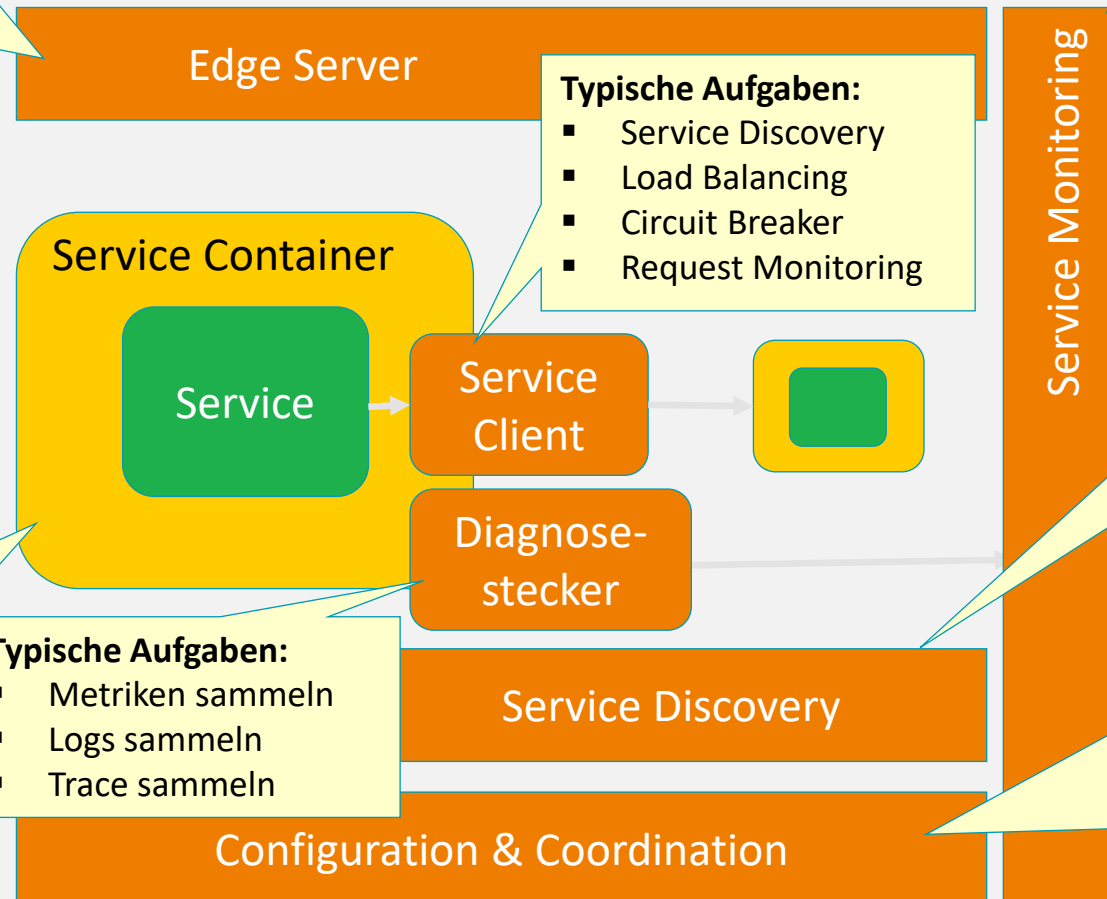
- Aggregation von Metriken
- Sammlung von Logs
- Sammlung von Traces
- Analyse / Visualisierung
- Alerting

Typische Aufgaben:

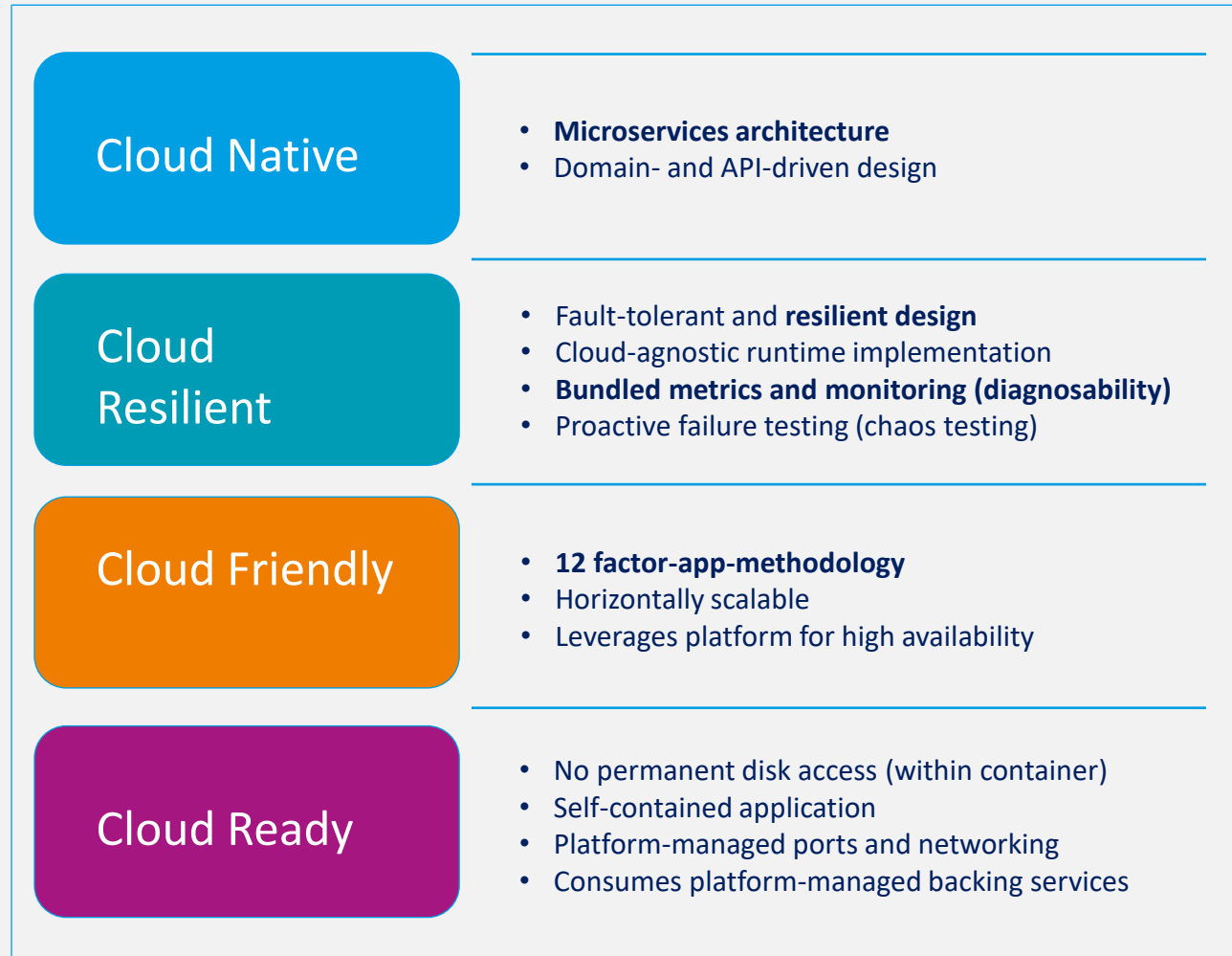
- Service Registration
- Service Lookup
- Service Description
- Membership Detection
- Failure Detection

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



Das Cloud Native Application Reifegradmodell



12 Factor App

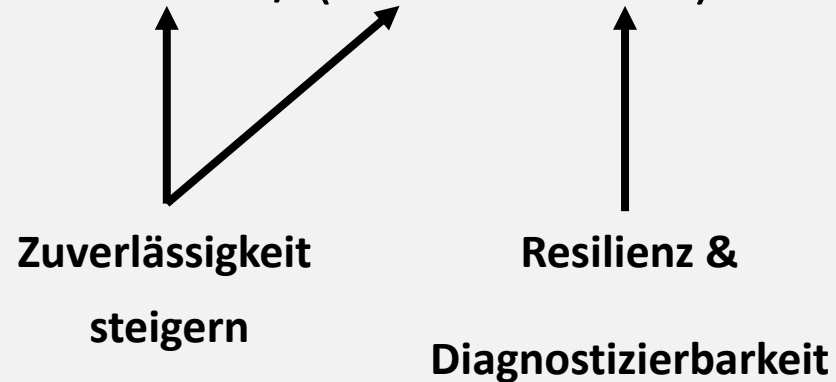
- | | | | |
|---|---|----|---|
| 1 | Codebase
One codebase tracked in revision control, many deploys. | 7 | Port binding
Export services via port binding. |
| 2 | Dependencies
Explicitly declare and isolate dependencies. | 8 | Concurrency
Scale out via the process model. |
| 3 | Configuration
Store config in the environment. | 9 | Disposability
Maximize robustness with fast startup and graceful shutdown. |
| 4 | Backing Services
Treat backing services as attached resources. | 10 | Dev/Prod Parity
Keep development, staging, and production as similar as possible |
| 5 | Build, release, run
Strictly separate build and run stages. | 11 | Logs
Treat logs as event streams. |
| 6 | Processes
Execute the app as one or more stateless processes. | 12 | Admin processes
Run admin/management tasks as one-off processes. |

<https://12factor.net/de>

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

Resilienz

$$\text{Verfügbarkeit} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$$



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)

Dignostizierbarkeit

