OBJEKTspektrum

# Cloud-native Apps 2.0

TECHNISCHE HOCHSCHULE LÜBECK

CoSA Center of Excellence

Inhaltliche Entwicklung: Prof. Dr. Nane Kratzke, Technische Hochschule Lübeck, Mönkhofer Weg 239, 23562 Lübeck
unter Mitwirkung von: Dr. Adersberger, QAware

Definition und Erläuterungen

## Definition

„Eine Cloud-native Applikation ist ein verteiltes, elastisches und horizontal skalierbares Softwaresystem, welches aus Service-orientierten Komponenten (Diensten) besteht, die Zustände in einem Minimum an zustandsbehafteten Komponenten isolieren. Die Applikation und ihre Komponenten werden gemäß Cloud-spezifischer Entwurfsmuster entworfen und auf einer elastischen Plattform betrieben."

(N. Kratzke, P.-C. Quint, Understanding Cloud-native Applications after 10 Years of Cloud Computing, Journal of Software and Systems, 2017)

## Cloud-native Applikationen (CNA): Softwaresysteme, die explizit für die Cloud entwickelt werden

- CNA unterscheiden sich damit von „klassischen" Multi-Tier Applikationen vor allem hinsichtlich des Grades ihrer **Verteilung** und ihrer **Elastizität**.
- CNA sind elastisch, d.h. sie können ihren **Ressourcenbedarf** dem zu verarbeitenden Workload (idealerweise vollautomatisch) **anpassen**, um so das Pay-as-you-go Prinzip und die **Kostenassoziativität** des Cloud Computing als Kostenvorteil zu nutzen.
- CNA sind insbesondere für Anwendungen interessant, deren **Workload schwer prognostizierbar** ist, deren Nutzung **Peak-Loads** unterliegt oder für die **exponentielles Wachstum** angestrebt wird.
- CNA werden häufig auf elastischen **Self-Service-Plattformen** betrieben und basieren auf **Software-Defined-Infrastruktur** Prinzipien.
- Architekturen von CNA sind meist Service-orientiert und folgen zunehmend der pragmatischen **Microservice**-Interpretation dieses Ansatzes.
- CNA Entwicklungsmethodiken beruhen oft auf Cloud-spezifischen Softwareentwicklungsmustern und **DevOps** Ansätzen.

**Weiterführende Links**:
- www.qaware.de/news/cloud-ready
- www.sigs-datacom.de/wissen

Beispiel für **Kostenassoziativität**: Der Betrieb einer vituellen Maschine für 100 Std. oder 100 virtueller Maschinen für 1 Std. kostet (fast) dasselbe.

## Prinzipien von CNA:

**I Isolated State**
Um horizontale Skalierbarkeit zu optimieren, versucht man zustandsbehaftete Komponenten und deren Skalierungskomplexität zu isolieren.

**D Distributed**
CNA sind verteilte Systeme (web-scale), die aus unabhängig voneinander austauschbaren Dienstenkomponiert werden.

**E Elastic**
CNA sind elastisch, d.h. sie fordern Ressourcen (Compute, Storage, Network) abhängig von einer Last an (wenig Last → wenig Ressourcen, hohe Last → mehr Ressourcen). Die Skalierung erfolgt dabei meist horizontal (mehr Ressourcen) und nicht vertikal (stärkere Ressourcen).

**A Automated**
CNA sollten meist auf automatisierten Plattformen betrieben werden die wenig bis keinen Operatoreingriff erfordern.

**L Loosely coupled**
Die Dienste rufen sich untereinander idealerweise nicht direkt auf, sondern interagieren indirekt über entkoppelnde Messaging-Lösungen.

In Anlehnung an: Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P., „Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications", Springer, 2014

**Design for Failure**
**(„Everthing fails all the time"):**
Mittels Resilienz schützen sich Cloud-native Applikationen z.B. per Circuit Breaker vor einer fehlerhaften und langsamen Umgebung. Einzelne Serviceausfälle sollten nie die gesamte Cloud-native Applikation beeinträchtigen.

## Stateless Services + Service Orchestration + Data (Stateful Services)

Managed Services | CNCF Projects | Microservice | Serverless | Container | Isolated State

**SaaS**

**Applikationen** werden aus Services komponiert.

**Frameworks**
Zur Serviceentwicklung existieren Frameworks für Microservices, für Nanoservices (FaaS) und für die Datenanalyse mittels Machine Learning oder BigData-Frameworks.

### Microservice Frameworks
- **Spring (Boot)**
- **Akka**
- **Ballerina**
- **Lagom**
- **Node.js**
- **Jolie (research)**
- **MicroProfile**
- **Dropwizard**
- ...

### Continuous Delivery
- **Drone.io**
- **Spinnaker**
- **JenkinsX**
- **GO-CD**
- **Concourse**
- **Knative**

**Circle CI, Codeship | Travis CI, AWS Code Pipeline | Azure Continuous Delivery | Google Cloud Continuous Delivery**

### Nanoservice Frameworks (FaaS)
**Serverless | Squeezer ...**

**Kubeless | Knative | OpenFaaS | OpenWhisk | nuclio | Fn project | Fission | Spring Functions ...**

- **AWS** Lambda
- **Azure** Functions
- **Google** Functions
- ...

### (Big) Data + Machine Learning
**Spark | Flink | Storm | Caffe | CNTK | Torch | Keras | ...**

- **AWS** {Glue | Athena | EMR, Transcribe | Translate | Comprehend | Lex | Deep Learning | ...}
- **Azure** {Databricks | Machine Learning | Stream Analytics | Data Lake Analytics | ...}
- **Google** {Dataflow | Dataproc | Vision | Speech | Translation | ...}

**PaaS**

**Service Operation**
Services sind lose gekoppelt und unabhängig voneinander auf CaaS Plattformen deployabar. Die Kopplung erfolgt entweder Event-basiert (Messaging) oder mittels Stateful Services (NoSQL DB)

### Service Discovery
- **CoreDNS**
- **Consul**
- **Eureka**
- ...

### Service Meshs
- **Istio**
- **Linkerd**
- **Consul Connect**
- ...

### Service Proxies
- **Envoy**
- **Prana**
- ...

### Fault Tolerant Communication
- **Finagle**
- **Hystrix**
- **Proxygen**
- **Resilience4j**
- ...

### Monitoring
- **Metrics: Prometheus | Sensu | ...**
- **Logs: ELF/EFK | Stack | fluentd | ...**
- **Traces: Jaeger | OpenTracing | ZipKin | ...**

- **Loggly, Datadog,** ...
- **AWS** { ES, CloudWatch }
- **Azure** Log Analytics
- **Google** Stackdriver Logging

### Messaging/Streaming
- **Active MQ Artemis**
- **Kafka**
- **NATS**
- ...

**AWS** SQS | **Azure** Event Grid | **Google** Task Queue | ...

### Database Solutions and Services
**MySQL | Maria DB | Postgres | Cockroach DB | Vitess | CitusDB | Mongo DB | Couch DB | Cassandra | Neo4j | ...**

- **AWS** {RDS | DynamoDB | SimpleDB | ... }
- **Azure** {SQL | DocumentDB | ... }
- **Google** {SQL | BigTable | Spanner | Memory Store | ... }
- ...

**CaaS**
Elastische Container Platt formen integrieren IaaS Ressourcen zu einer verteilten Cloud Runtime Environment für Services.

### Container Platforms (Ausführung standardisierter Deployment Units, Container as a Service)
- **Kubernetes**
- **Docker Swarm**
- **DC/OS (Mesos)**
- **Cloud Foundry**
- **OpenShift**
- **Nomad**

**AWS** ECS | EKS | **Azure** Kubernetes Service | **Google** Container Engine

### Clustered Storage Solutions
- **Commercial: Quobyte | Portworx**
- **Open Source: Ceph | FS | IPFS | HDFS | Objective FS | XtreemFS | Rook | ...**

- **AWS** {S3, EBS, Glacier, ... }
- **Azure** {Files, BLOBS, Data Lake, ... }
- **Google** { Filestore, Datastore, ... }

**IaaS**

**Infrastructure**
IaaS Provider stellen Computing, Networking und Storage Ressourcen bereit.

aws | Alibaba Cloud | Azure | Google Cloud | openstack. | vmware

## Cloud Application Maturity Level

| Level | Maturity |
|---|---|
| 3 \| Adaptive | **Cloud Native**<br>› Applikation skaliert elastisch (lastabhängig)<br>› Migration auf andere Infrastrukturen ohne Service Downtime |
| 2 \| Abstracted | **Cloud Resilient**<br>› Dienste sind stateless<br>› Applikation ist designed for failure<br>› Applikation ist Infrastruktur agnostisch (runs anywhere) |
| 1 \| Loosely Coupled | **Cloud Friendly**<br>› Applikation ist aus lose gekoppelten Diensten komponiert<br>› Dienste sind per Namen adressierbar<br>› Applikation berücksichtigt 12-Factor App Prinzipien<br>› Applikation trennt Compute und Storage Dienste<br>› Applikation basiert auf Compute-, Storage- und Netzwerkdiensten |
| 0 \| Virtualized | **Cloud Ready**<br>› Applikation läuft auf virtualisierter Infrastruktur<br>› Applikation kann automatisiert ausgeprägt werden |