Microservices und Sicherheit

Aufsatz zum Seminar Komponenten, Agenten und Workflows in verteilten Systemen

Felix Ortmann und Konstantin Möllers
{0ortmann,1kmoelle}@informatik.uni-hamburg.de

Universität Hamburg
Fachbereich Informatik
Vogt-Kölln-Straße 30
22527 Hamburg

16. Januar 2016

Gliederung des Vortrags

Gliederung des Vortrags

Eigenschaften von Microservices

„gleiches zu gleichem!“

(Fowler & Lewis, 2014)

Eigenschaften von Microservices

„gleiches zu gleichem!“

„le Service c’est moi!“

(Fowler & Lewis, 2014)

Eigenschaften von Microservices

„gleiches zu gleichem!“

„le Service c’est moi!“

„toll, ein anderer macht’s!“

(Fowler & Lewis, 2014)

Abgrenzung

- **SOA**
  fehlt: gekapseltes Frontend.
  hier: klare Trennung der Systeme, generische Schnittstelle durch das Web.
- **MAS**
  fehlt: soziologische Theorie.
  hier: soziotechnische Perspektive.
- **MOrgaS** (Wester-Ebbinghaus, 2010)
  fehlt: Schachtelbarkeit und Hierarchie.
  hier: Föderalismus, dezentralisierte Governance.

Für und Wider

Pro

+ Skalierbarkeit auf Enterprise-Ebene
+ höhere Entscheidungsbefugnis für Entwickler
+ *Rapid Prototyping* von Anwendungen möglich
+ schnellerer Zugang zu neuen Technologien und Programmiersprachen

Kontra

– hoher technischer Aufwand
– ausführliche Absprache und gute Tests obligatorisch
– Latenz und Overhead durch Kommunikation
– übergreifendes System-Refactoring schwer bis unlösbar

Motivation: Microservices + Sicherheit

- unabhängige, kleine Teams
  - ⇒ bessere, leichtere Kommunikation
  - ⇒ effizientes Debugging und Pair-Programming
- Unabhängigkeit der Ausführung von Services
  - ⇒ Hoheit über eigene Software
  - ⇒ rasches, häufiges Deployment und schließen von Sicherheitslücken
- einfache, generelle Schnittstelle
  - ⇒ viele gut getestete Tools für die Kommunikation
  - ⇒ Skalierung leichter und schneller möglich

Gliederung des Vortrags

Technologien zur Realisierung von MS

- Köhasion, Autonomie, Kooperation wahren und unterstützen
- softwaretechnisch: *Isolation* & *Kapselung*
- System formen aus einzelnen „Bausteinen“
- virtuelle Maschinen & Container

VMs & Container

Virtuelle Maschine

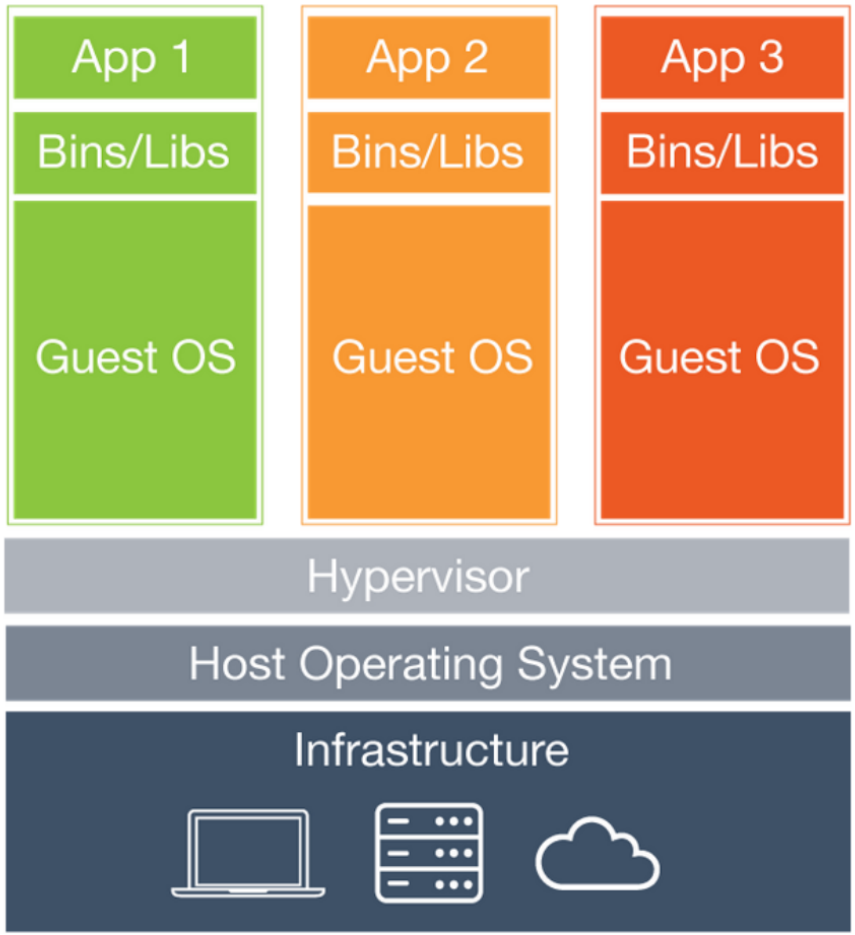
- isoliert Anwendung(en)
- „sandboxed“ Execution Environment
- Anwendung $\approx$ VM-Image $\rightarrow$ Replizierbar
- „schwer“
  - $\Rightarrow$ beinhaltet eigene Libs und Gast-Betriebssystem
  - $\Rightarrow$ Images meist groß

VMs & Container

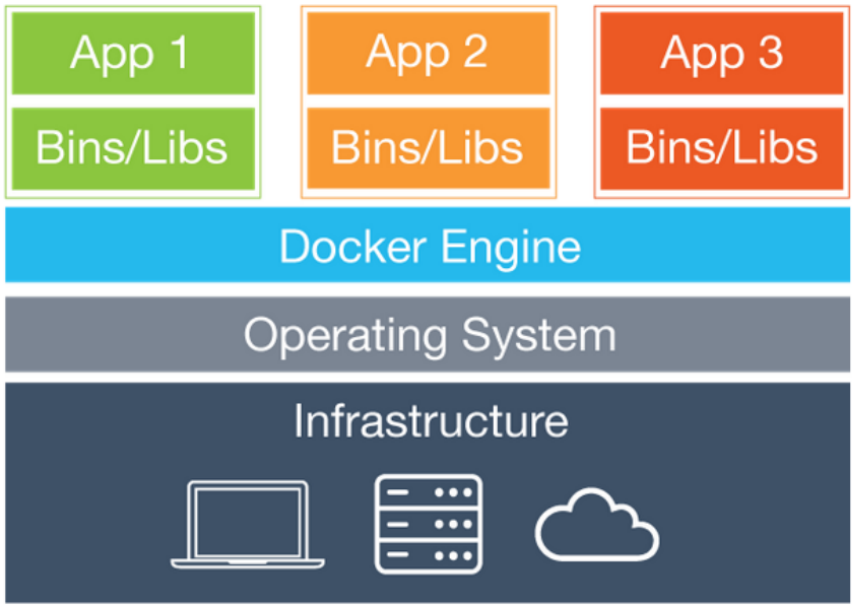
Container (insb. Docker)

- isoliert genau eine Anwendung
- *Namespaces*, *Network Interfaces*, *Cgroups*, *Chroots*
- Replikation via Dockerfile oder Image
- „leicht“
  - $\Rightarrow$ beinhaltet nur App und Abhängigkeiten
  - $\Rightarrow$ Images klein
  - $\Rightarrow$ Ausführung: 1 Prozess

VMs & Container



Virtual Machines



Containers

Microservice Systeme

- Zusammenspiel vieler Microservices
- „viele bewegliche Teile“ $\rightarrow$ neue Probleme:

1. Service Discovery
2. Erreichbarkeitsüberwachung
3. Skalierbarkeit
4. Load Balancing
5. Deployment
6. Replizier- & Portier & Ersetzbarkeit
7. Kommunikationssicherheit
8. Verhaltensüberwachung (Logging)

- „Orchestration“: *K8s*, *Mesos*, *ECS*, *Azure* ...

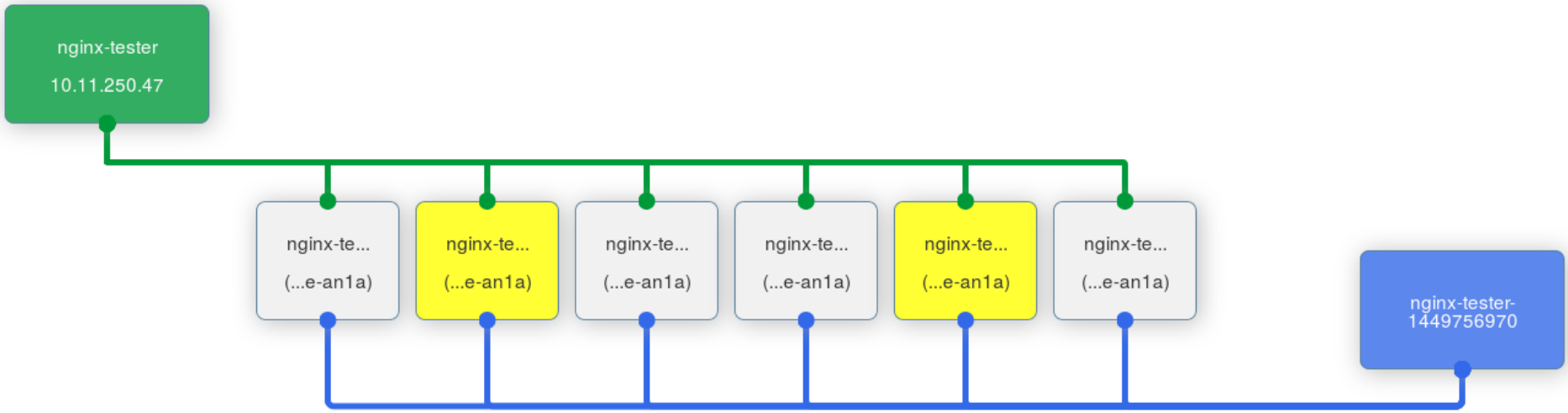
Kubernetes

„Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts.“(*Kubernetes Documentation*, o. J.)

- löst obige Problemstellungen 1-7
- abstrahiert Lösung, um mit Docker Containern MS Systeme zu realisieren

Kubernetes

- Pods
- Services
- Replication Controller



Gliederung des Vortrags

Kommunikationsbedarf

- „System“ $\rightarrow$ Vernetzung der Microservices
- *Orchestration Tools* versuchen, das Problem zu adressieren
- Kommunikation fast immer Event-basiert
- Technologien: REST & Message Queues
- Problem: Unsichere Kanäle

RESTful APIs

- „Representational State Transfer“
- Request / Response (HTTP)
- synchron – Gesprächspartner bekannt
- abhörbar, manipulierbar, offene APIs
- HTTPS $\rightarrow$ ~~abhörbar, manipulierbar~~ *offene APIs*
- Authentifizierung an API + HTTPS – Mindestsicherheit nach (Newman, 2015)

Message Queues

- asynchron & indirekt – Empfänger evtl. unbekannt
- bessere Skalierbarkeit, Worker-Chains
- Transport von/zu Queue unsicher
- *RabbitMQ, ZeroMQ*

System Sandboxing

- Gesamtsystem hinter Firewall
- Subnetze verwenden
- keine Schnittstelle zum WWW
- verlagert den Angriffsvektor, eliminiert ihn nicht!

Validierung und Verifikation

- Grobe Maßnamen:
  - ⇒ Geteiltes JSON Schema
  - ⇒ Hashing des Inhalts
  - ⇒ Identifikation des Absenders
- Im Detail:
  - ⇒ Pro Anwendungsfall Schutzziele definieren
  - ⇒ Passende Technologie pro Schutzziel
  - ⇒ Erhöhter Implementationsaufwand wenn richtig gemacht

Gliederung des Vortrags

Vertraulichkeit

- Informationen nur Befugten zugänglich
- Informationsflüsse festlegen und kontrollieren

Vorgestellte Verfahren:

- *OAuth*
- *JSON Web Tokens*

Vertraulichkeit

- Informationen nur Befugten zugänglich
- Informationsflüsse festlegen und kontrollieren

Vorgestellte Verfahren:

- *OAuth*
- *JSON Web Tokens*

OAuth 2

- RFC 6749 (Hardt, 2012)
- ermöglicht *Single Sign On*
- basiert auf Web-Technologie (vor allem HTTP)
- vielfach genutzt, z.B. von Google, Facebook, …

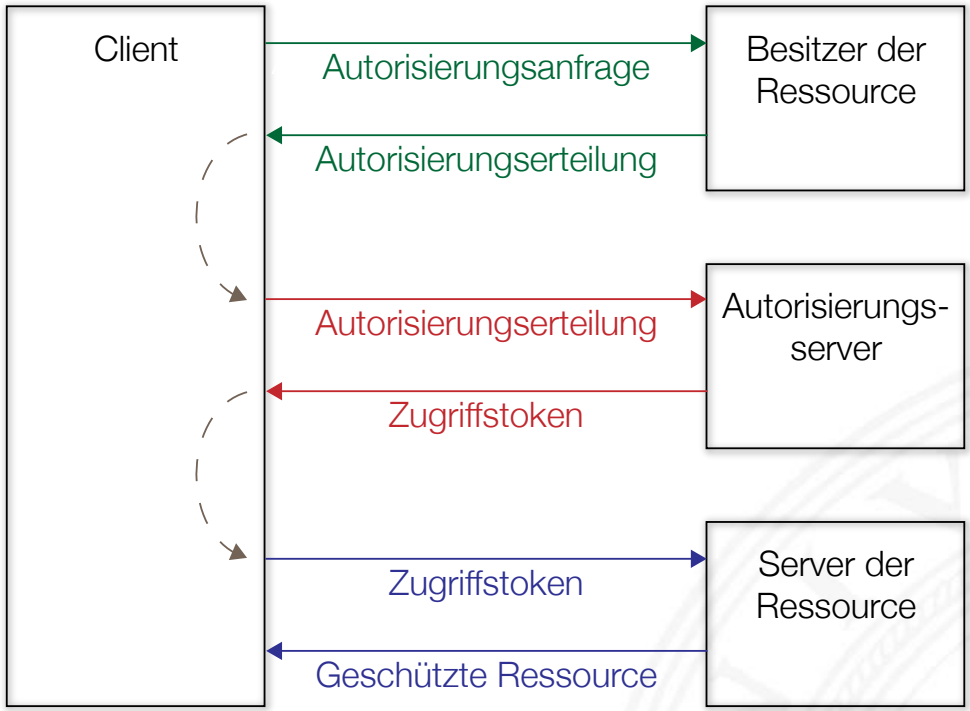
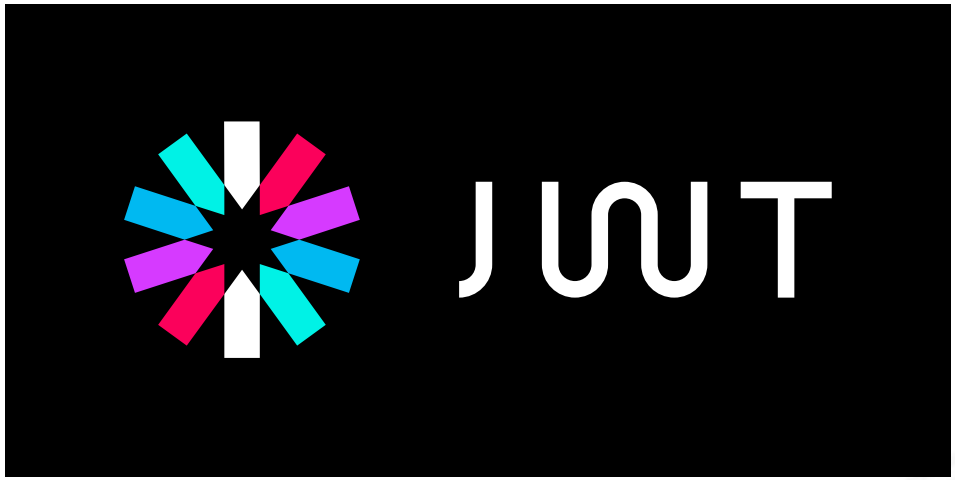


Abbildung: Protokollablauf bei OAuth 2 (übersetzt aus RFC 6749)

JSON Web Tokens



- RFC 7519 (Jones, Bradley & Sakimura, 2015)
- simpler gegenüber *OAuth*
- jeder Client kann verifizieren
- klein, daher via HTTP übertragbar
- kann auch von Browsern geparsed werden

JSON Web Tokens

```
{
    "alg": "RS256", // Verschlüsselungsalgorithmus
    "typ": "JWT" // „JSON Web Token", offen für weitere
}
```

Abbildung: JWT-Header

```
{
    "thema": "Microservices und Sicherheit",
    "modul": "Entwicklung verteilter Systeme",
    "kursnr": "64-400"
}
```

Abbildung: JWT-Payload

JSON Web Tokens

Abbildung: Vollständiges *JSON Web Token*

Integrität

- **Datenintegrität**:
  Vollständigkeit & Korrektheit der Daten
- **Systemintegrität**:
  korrekte Funktionsweise des Systems

Vorgestellte Verfahren:

- Verschlüsselung
- Integrationstests

Integrität

- **Datenintegrität**:
  Vollständigkeit & Korrektheit der Daten
- **Systemintegrität**:
  korrekte Funktionsweise des Systems

Vorgestellte Verfahren:

- Verschlüsselung
- Integrationstests

Verschlüsselte Protokolle

- **HTTPS** – *HyperText Transport Protocol Secure*
  - RFC 2818
  - sichere Übertragung von Daten + Informationen
- **WSS** – *WebSockets Secure*
  - RFC 6455
  - sichere und schnelle Übertragung von Daten in einem Socket
- **SSH** – *Secure Shell*
  - RFC 42{50-56}
  - sichere entfernte Ausführung via RSA und AES
- **SFTP** – *SSH File Transfer Protocol*
  - (Working draft)
  - sichere Übertragung von Dateien

Integrationstests für Microservices

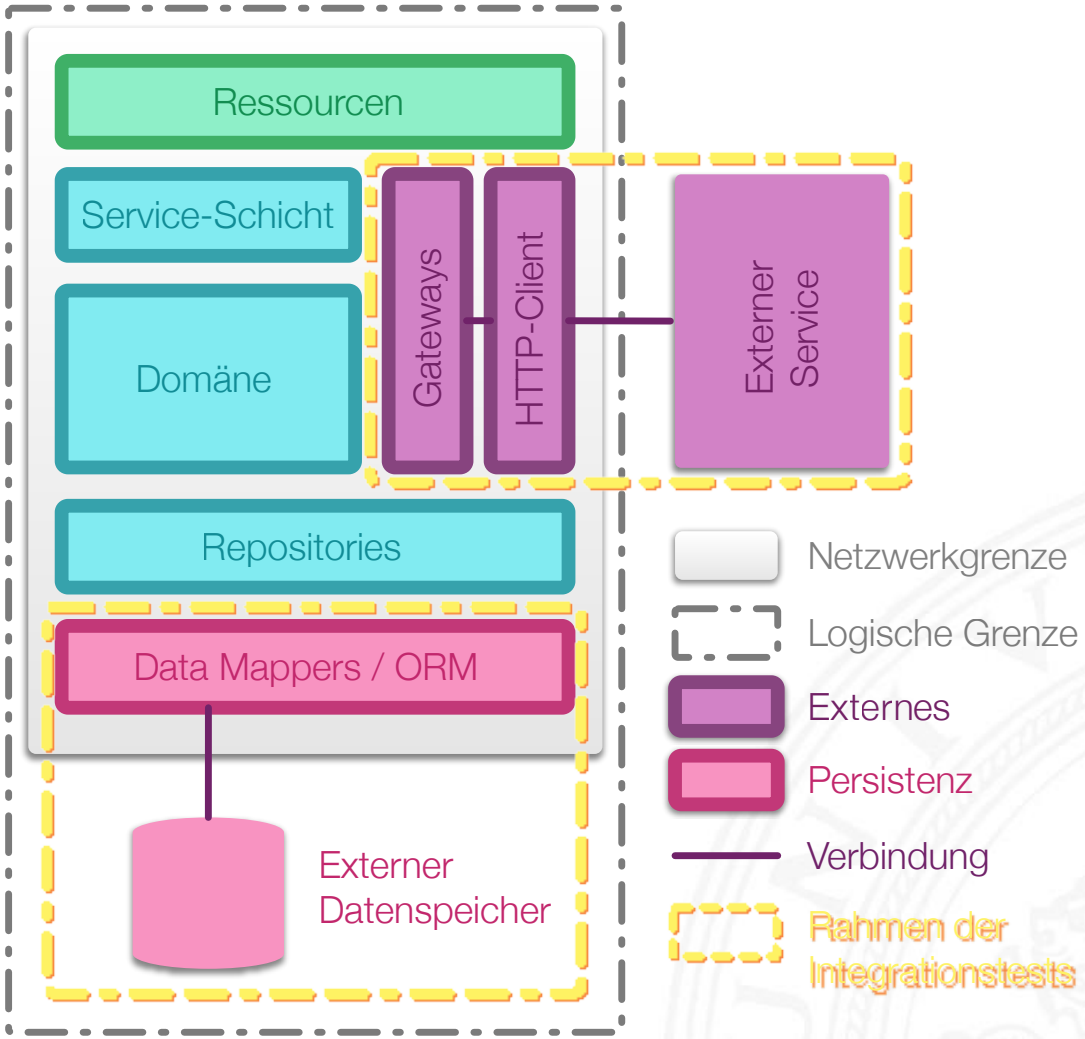


Abbildung: Integrationstests in einer Microservice-Architektur (Clemson, 2014)

Verfügbarkeit

- Systeme stets betriebsbereit
- zeitgerechter Zugriff auf Daten jederzeit möglich

Vorgestellte Verfahren:

- horizontale Skalierung & Lastverteilung
- *Baker Street*

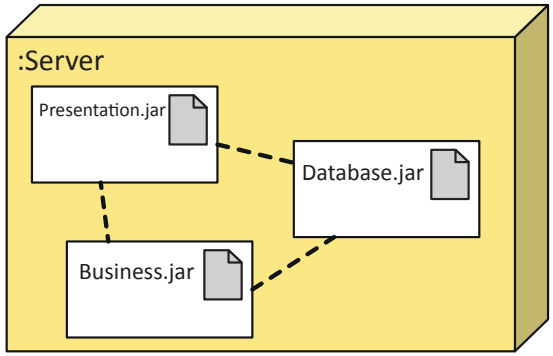
Verfügbarkeit

- Systeme stets betriebsbereit
- zeitgerechter Zugriff auf Daten jederzeit möglich

Vorgestellte Verfahren:

- horizontale Skalierung & Lastverteilung
- *Baker Street*

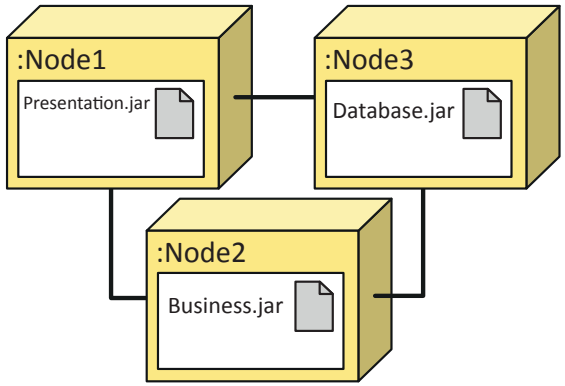
Horizontale Skalierung & Lastverteilung



Scale Up

„größere Box“

- Hardwarekapazität erweitern (CPU, RAM, …)
- einfach & günstig
- *Single Point of Failure*
- unterliegende Software muss auch skalieren



Scale Out

„mehr Boxen“

- mehr Server hinzufügen und Lastverteilung
- schützt vor Hardwarefehlern
- leicht für die Microservice-Architektur!

(Meier et al., 2008)

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Arten von Lastverteilung

- **Zufall**
  - einen zufälligen Knoten wählen
  - sehr simpel, aber asymmetrisch
- **Sharding**
  - Knoten auf Basis eines Indexes wählen
  - bei guter Schlüsselwahl symmetrisch
- **Auslastung**
  - Knoten mit geringster Auslastung wählen
  - zusätzliche Belastung der Knoten

Baker Street

- Open-Source-Projekt von 2015
- `http://bakerstreet.io/`

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an **Sherlock**

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an Sherlock weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an Sherlock weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an Sherlock weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an Sherlock weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von Watson an Sherlock weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an **Sherlock** weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an **Sherlock** weiter

(Li, 2015)

Baker Street

- **Sherlock**
  - *HAProxy*-basierter *Load Balancer*
  - lokal, findet andere Knoten
- **Watson**
  - *Health Checker*, prüft Zustand des Microservices
  - übernimmt Anmeldung
- **Datawire-Verzeichnis**
  - globale Diensterkennung
  - leitet Nachrichten von **Watson** an **Sherlock** weiter

(Li, 2015)

Baker Street

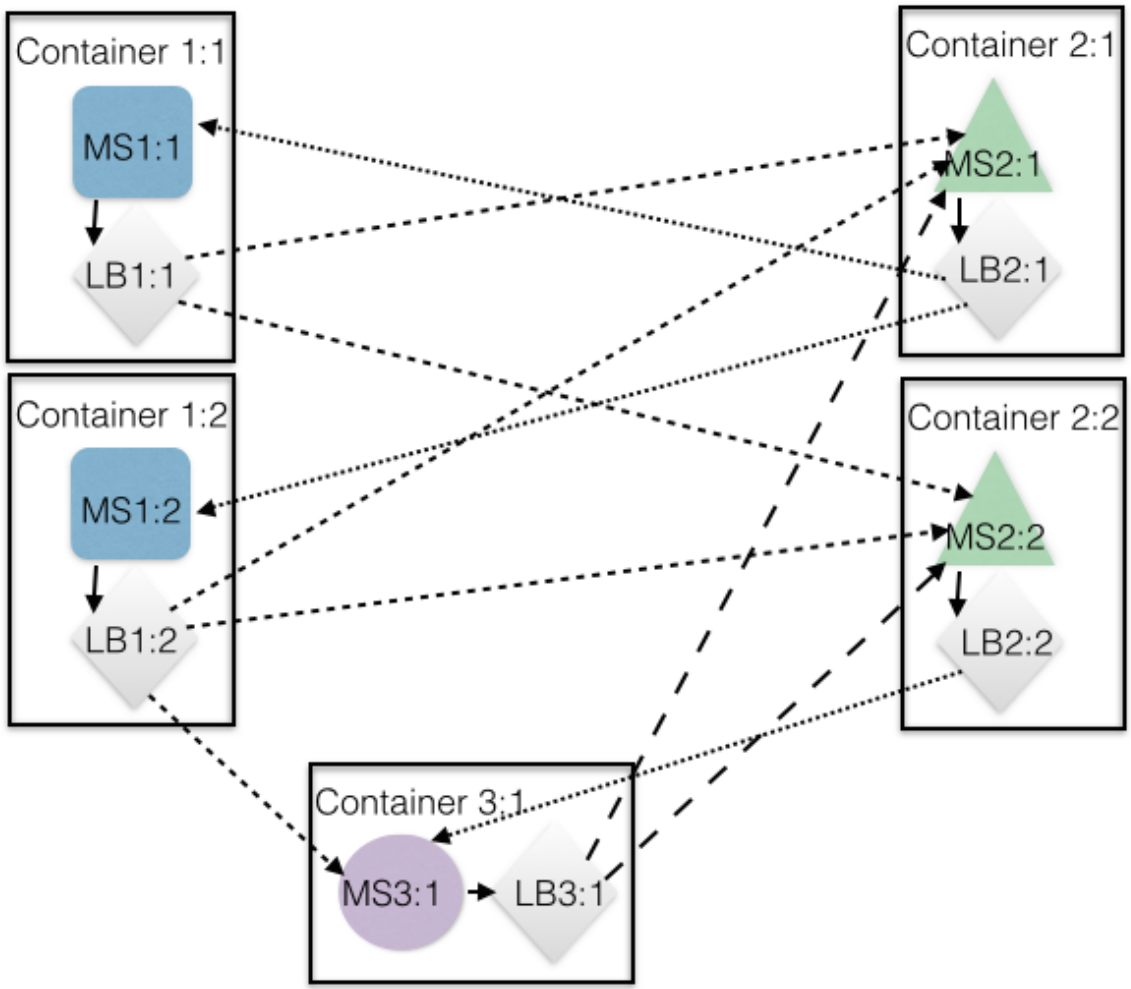


Abbildung: Schematische Darstellung von clientseitiger Lastverteilung (aus Li, 2015)

Gliederung des Vortrags

Zusammenfassung

- Konzepte wie Kohäsion, Autonomie und Kooperation
- Container & VMs
- Herausforderung: Zusammenspiel von MS
- Kommunikation via REST & Queues
- Vertraulichkeit: *OAuth 2* & JWT
- Integrität: Verschlüsselung & Tests
- Verfügbarkeit: *Load Balancing* & *Baker Street*

Gliederung des Vortrags

Gliederung des Vortrags

Literaturverzeichnis I

Auth0 Inc. (o. J.). *Introduction to JSON Web Tokens.* Webartikel. Zugriff am 2015-12-30 auf `http://jwt.io/introduction/`

Bedner, M. & Ackermann, T. (2010). Schutzziele der IT-Sicherheit. *Datenschutz und Datensicherheit*, *5*, 323 - 328.

Clemson, T. (2014, November). *Testing Strategies in a Microservice Architecture.* Präsentation. Zugriff am 2015-12-31 auf `http://martinfowler.com/articles/microservice-testing/`

Fowler, M. & Lewis, J. (2014, März). Microservices. A definition of this new architectural term.

Hardt, D. (2012, Oktober). *The OAuth 2.0 Authorization Framework* (RFC Nr. 6749). Internet Engineering Task Force. Internet Requests for Comments. Zugriff am 2015-12-29 auf `https://tools.ietf.org/html/rfc6749` doi: http://dx.doi.org/10.17487/RFC6749

Literaturverzeichnis II

Hohpe, G. & Woolf, B. (2003). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Jones, M., Bradley, J. & Sakimura, N. (2015, Mai). *JSON Web Token (JWT)* (RFC Nr. 7519). Internet Engineering Task Force. Internet Requests for Comments. Zugriff am 2015-12-30 auf `https://tools.ietf.org/html/rfc7519` doi: http://dx.doi.org/10.17487/RFC7519

Kubernetes documentation. (o. J.). `http://kubernetes.io/v1.1/docs/`.

Li, R. (2015, Oktober). Baker Street: Avoiding Bottlenecks with a Client-Side Load Balancer for Microservices. Zugriff am 2015-12-08 auf `http://thenewstack.io/baker-street-avoiding-bottlenecks-with-a-client-side-load-balancer-for-microservices/`

Marmol, V., Jnagal, R. & Hockin, T. (2015, Februar). Networking in containers and container clusters. Netdev 0.1, Ottawa, Canada.

Literaturverzeichnis III

Meier, Homer, Hill, Taylor, Bascode, Wall, … Bogawat (2008). *Chapter 5. Deployment Patterns.* Zugriff am 2014-12-07 at 19:15 auf `https://apparchguide.codeplex.com/wikipage?title=Chapter%205%20-%20Deployment%20Patterns`

Newman, S. (2015). *Building microservices*. O'Reilly Media, Incorporated.

Richardson, C. (o. J.). *A pattern laguage for microservices.* `http://microservices.io/patterns/`.

Webber, J., Parastatidis, S. & Robinson, I. (2010). *Rest in practice: Hypermedia and systems architecture*. O'Reilly Media.

Wester-Ebbinghaus, M. (2010). *Von Multiagentensystemen zu Multiorganisationssystemen* (Unveröffentlichte Dissertation). Universität Hamburg.