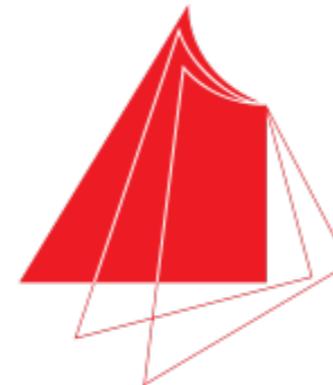


# Verteilte Systeme Master Lab

[christian.zirpins@hs-karlsruhe.de](mailto:christian.zirpins@hs-karlsruhe.de)

Microservice Middleware



Hochschule Karlsruhe  
Technik und Wirtschaft

UNIVERSITY OF APPLIED SCIENCES



# Zeitplan

## VERTEILTE SYSTEME MASTER LABOR - SOMMERSEMESTER 2020

				Präsenztermine (Onlinetreffen) jeweils Mi 8:00 Uhr		
#	KW	Tag	Laboraufgabe	Termin	Form	Abgabe
1	12					
2	13					
3	14		<b>A1+A2 Analyse der Legacy Anwendung und Microservice Architekturentwurf</b>	<b>Seminar: Microservice Architekturen</b>	Mediacast	
4	15			<b>Seminar: Web Shop Basistechnologien</b>	Mediacast	
5	16			<b>Seminar: Microservice Basistechnologien</b>	Mediacast	
6	17	22.4.		<b>Lab: Analyse Legacy App / Microservice Entwurf</b>	Onlinetreffen	
7	18	29.4.		<b>Meetup: Abgabe von Analysen und Entwürfen</b>	Onlinetreffen	<u>System- und REST-Modelle</u>
8	19		<b>A3 Microservice Implementierung</b>	<b>Seminar: Microservice Middleware</b>	Mediacast	
9	20	13.5.		<b>Lab: Web Shop REST Microservices</b>	Onlinetreffen	
10	21	20.5.		<b>Lab: Microservice Infrastruktur mit Netflix OSS</b>	Onlinetreffen	
11	22	27.5.		<b>Meetup: Web Shop Microservices</b>	Onlinetreffen	<u>Web Shop V2</u>
	23			<i>Pfingsten</i>		
12	24		<b>A4 Web Shop Implementierung</b>	<b>Seminar: Microservice Security</b>	Mediacast	
13	25	17.6.		<b>Lab: Microservice Security mit OAuth</b>	Onlinetreffen	
14	26	24.6.		<b>Lab: Frontend integration</b>	Onlinetreffen	
15	27	1.7.		<b>Meetup: Web Shop V3</b>	Onlinetreffen	<u>Web Shop V3</u>

# Struktur — Microservice Middleware

## Microservice-basierte Systeme

- Charakteristik von Microservice-basierten Systemen
- Herausforderungen im Betrieb
- Microservice Middleware

## Spring Cloud Framework

- Spring (Cloud (Netflix))
- Netflix-basierte Systemarchitektur

## Laboraufgabe

# Microservice Refactoring: Implikationen

*Wie verändert sich eine Systemlandschaft, wenn wir monolithische Systeme durch eine (große) Menge von Microservices ersetzen?*

**Die Systemlandschaft beinhaltet *mehr Systeme*.**

- Deployment und Management betrifft jeden einzelnen Microservice.
- Microservices werden alle einzeln installiert, entfernt oder ersetzt.

**Microservices wirken als *kooperative SOA* zusammen.**

- Sie stellen vernetzte Basis-, Vermittler- oder Prozessdienste bereit.
- Sie sind gleichzeitig Serviceprovider und Clients anderer Microservices.

**Die Systemlandschaft entwickelt *mehr Dynamik*.**

- Für jeden Microservice werden diverse Instanzen verwaltet.

# Microservice Refactoring: Implikationen (2)

**Die Systemlandschaft zeigt eine *erhöhte Fehlerrate*.**

- Bei mehr Komponenten kommt es öfter vor, dass eine fehlschlägt.
- Die *Mean Time between Failure (MTBF)* sinkt.

**API Gateways bilden *öffentliche APIs*.**

- Sie schirmen die SOA als Public Enterprise Services ab.

# Herausforderungen im Betrieb

**Wie sind die Microservices *konfiguriert* und ist das korrekt?**

- Viele Property Dateien/Tabellen sind über diverse Rechner verteilt.
- Die konsistente Wartung der Konfigurationen ist aufwändig.

**Welche Microservices sind *ausgerollt* (bzw. "deployed") und wo?**

- Hosts und Ports aller Serviceinstanzen müssen nachverfolgt werden.
- Die Orte der Services ändern sich praktisch kontinuierlich.

**Wie kann das *Routing* aktuell gehalten werden?**

- API Requests erfordern Routing (Reverse Proxies, Client Konfiguration)
- Die manuelle Pflege von Routingtabellen ist langsam und fehleranfällig.

# Herausforderungen im Betrieb (2)

## Wie lassen sich *Fehlerkaskaden* vermeiden?

- Durch die Abhängigkeiten zwischen Services werden Fehler propagiert.
- Einzelne Ausfälle können sich auf große Systembereiche ausbreiten.

## Wie wird *kontrolliert*, ob alle Services gestartet sind / korrekt laufen?

- Die Nachverfolgung aller Systemzustände von Services ist aufwändig.
- Die Zustände sind einzeln und in ihrer Gesamtheit zu bewerten.

## Wie können *Nachrichten zwischen Services* verfolgt werden?

- Zur Fehleranalyse müssen Abläufe im System nachzuvollziehen sein, um die Fehlerquelle zu identifizieren und Fehler zu beheben.
- Möglicherweise können "hängende" Prozesse fortgesetzt werden.

# Herausforderungen im Betrieb (3)

Wie lässt sich sicherstellen, dass *nur API Gateways öffentlich* sind.

- Externer Zugriff auf interne Microservices muss verhindert werden.

Wie können *API Services sicher* gemacht werden?

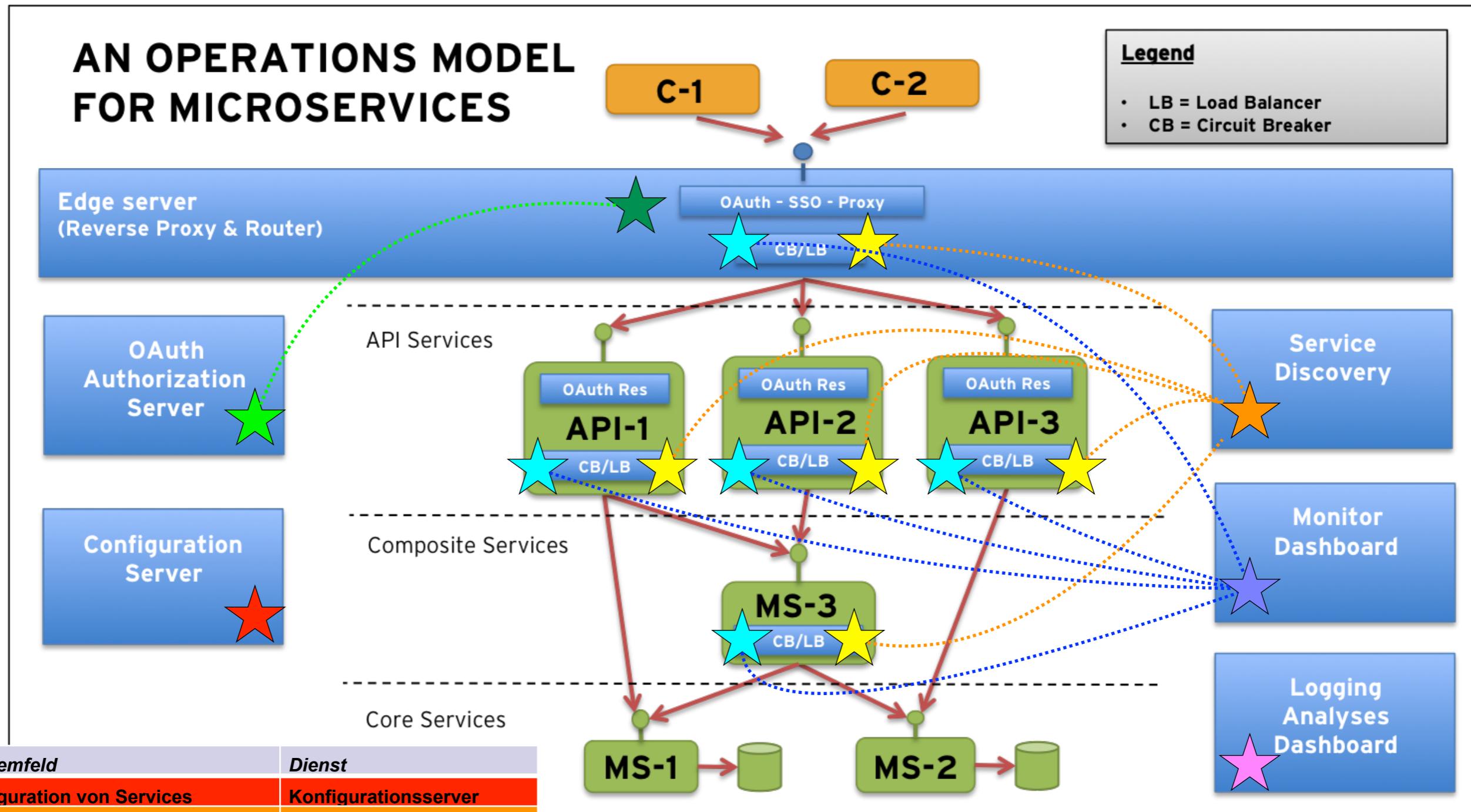
- API Services sind Public Enterprise Services.
- Es gelten die Sicherheitsanforderungen für externen Zugriff.

# Microservice Middleware

- Problemfelder im Betrieb von Microservice-basierten Systemen lassen sich größtenteils durch zusätzliche Systemkomponenten (bzw. Dienste) auf Ebene der Middleware adressieren.

<b>Problemfeld</b>	<b>Dienst</b>
Konfiguration von Services	Konfigurationsserver
Finden von Serviceinstanzen	Discovery Server
Routing von Requests	Router / Load Balancer
Fehlerketten	Circuit Breaker
Kontrolle von Serviceinstanzen	Service Monitor
Nachrichtenverfolgung	Log Server
Öffentliche APIs	Edge Server
Sichere APIs	Autorisierungsserver

# Microservice Middleware Architektur



Problemfeld	Dienst
Konfiguration von Services	Konfigurationsserver
Finden von Serviceinstanzen	Discovery Server
Routing von Requests	Router / Load Balancer
Fehlerketten	Circuit Breaker
Kontrolle von Serviceinstanzen	Service Monitor
Nachrichtenverfolgung	Log Server
Öffentliche APIs	Edge Server
Sichere APIs	Autorisierungsserver

(Larson 2015a)

# Microservice Middleware Dienste

## ■ Zentraler Konfigurationsserver

- Der Server verteilt eine *zentral administrierte Basiskonfiguration*.
- Microservices nutzen eine API um *Konfigurationen zu lesen*.

## ■ Service Discovery Server

- Der Server betreibt eine *Registry für Microservices* und deren Endpunkte.
- Microservices nutzen eine API um sich beim Start zu *registrieren*.

## ■ Dynamisches Routing und Load Balancer

- Routing Mechanismen *finden Microservices* über die Discovery API.
- Load Balancer *leiten Anfragen* zu alternativen Serviceinstanzen.

# Microservice Middleware Dienste

- **Circuit Breaker Mechanismus ("Schutzschalter")**
  - Dieser Client Mechanismus *unterbindet Anfragen an defekte Services.*
  - Für genutzte Services werden die *Vitalfunktionen überwacht.*
- **Monitoring Dashboard**
  - Dashboards *sammeln und präsentieren Vitaldaten* der Circuit Breaker.
  - Sie zeigen den *Gesundheitszustand* und können *Alarme* auslösen.
- **Zentraler Log Server**
  - Der Log Server *sammelt und kombiniert Logdateien* von Microservices.
  - Eine zentrale Datenbank ermöglicht *Suche und Dashboardfunktionen.*

# Microservice Middleware Dienste

## ■ Edge Server

- Edge Server verarbeiten alle externen Anfragen als *Zugriffsschutz und Façade* für interne Microservices.
- Als *Reverse Proxies* nutzen sie dynamisches Routing und Load Balancing.

## ■ Sicherheitsmechanismen

- Die offengelegten *API Gateway Services müssen gesichert werden*.
- Edge Server koordinieren die Autorisierung von API Zugriffen und nutzen *3rd Party Dienste* wie z.B. Autorisierungsserver (z.B. OAuth 2.0).

# Struktur — Microservice Middleware

## Microservice-basierte Systeme

- Charakteristik von Microservice-basierten Systemen
- Herausforderungen im Betrieb
- Microservice Middleware

## Spring Cloud Framework

- Spring (Cloud (Netflix))
- Netflix-basierte Systemarchitektur

## Laboraufgabe

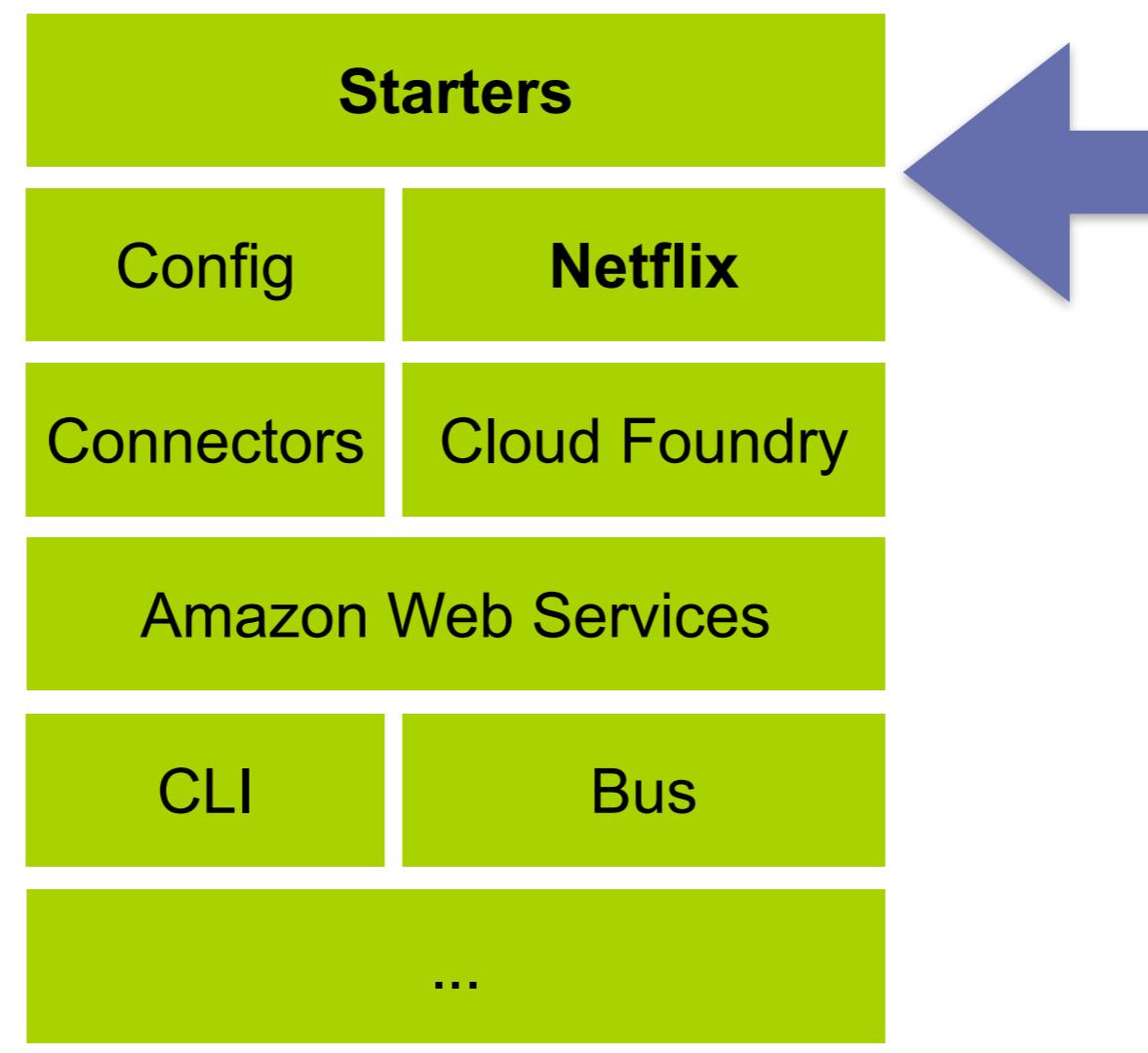
# Spring IO Platform



(Gierke 2015)

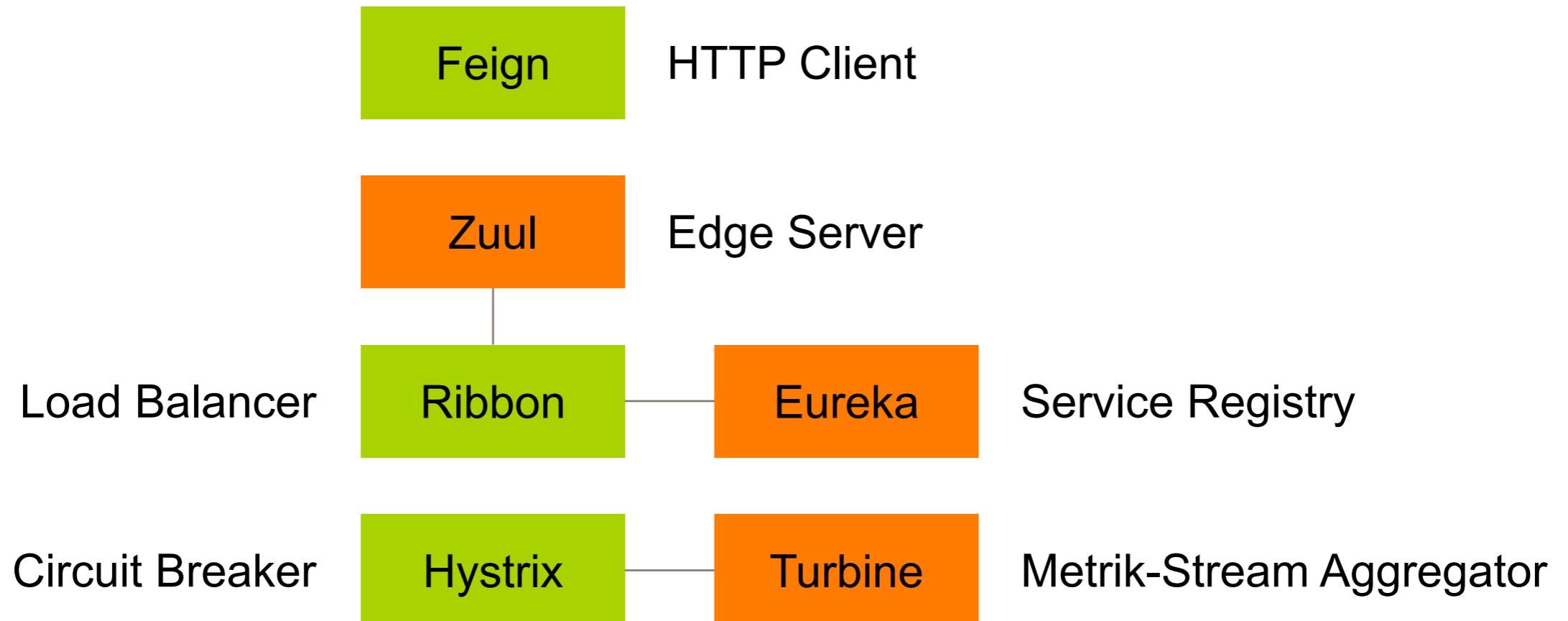
# Spring Cloud

- **Spring Cloud** ist ein Framework mit Werkzeugen zur *Unterstützung typischer Muster und Anwendungsfälle verteilter Systeme.*
  - Solche Muster finden sich z.B. bei Cloud- Anwendungen und/oder Microservice-basierten Systemen.

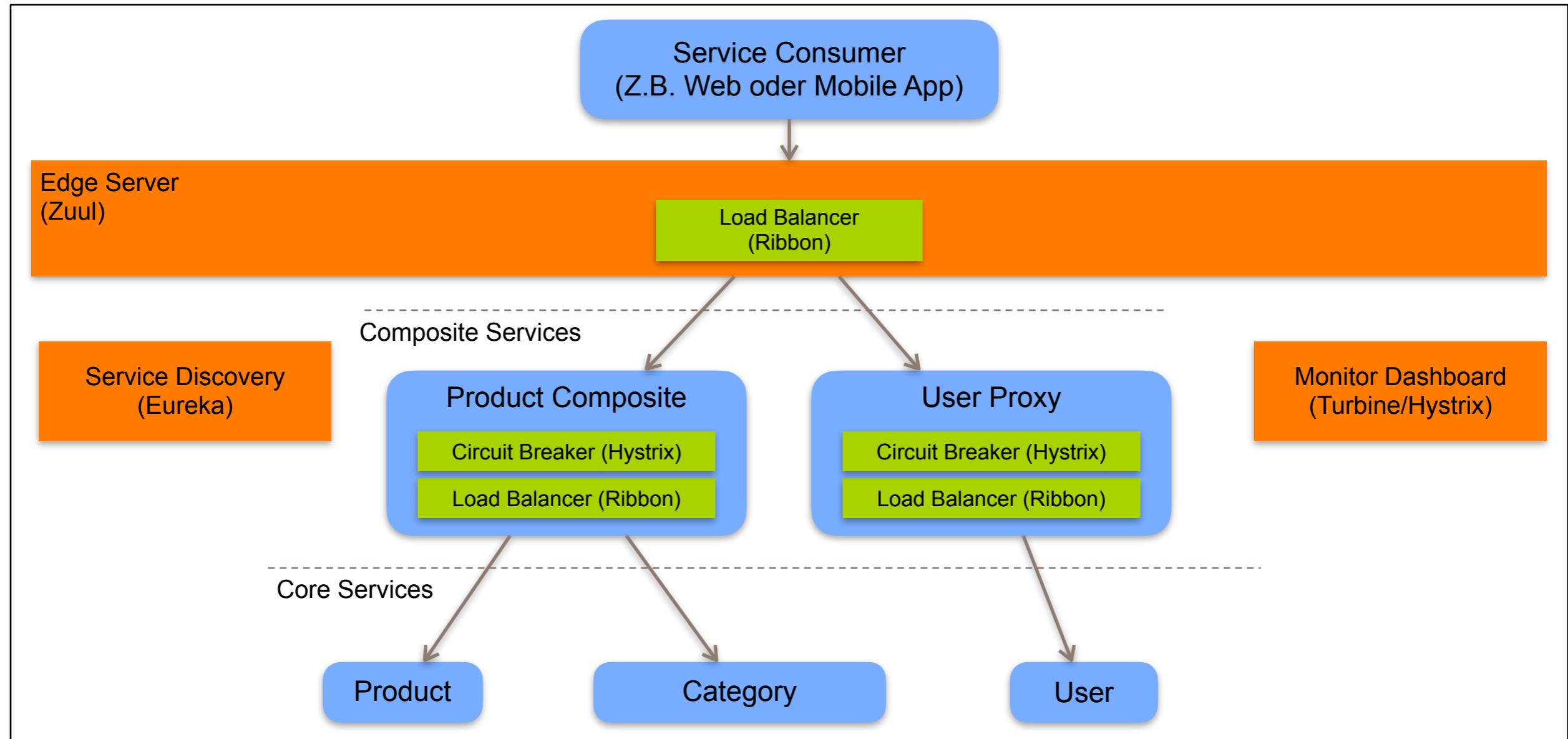


# Spring Cloud Netflix

- Netflix stellt die Laufzeitdienste und Bibliotheken seiner Microservice Tools als freie Software zur Verfügung.
- Spring Cloud integriert viele davon für die Nutzung mit Spring IO.



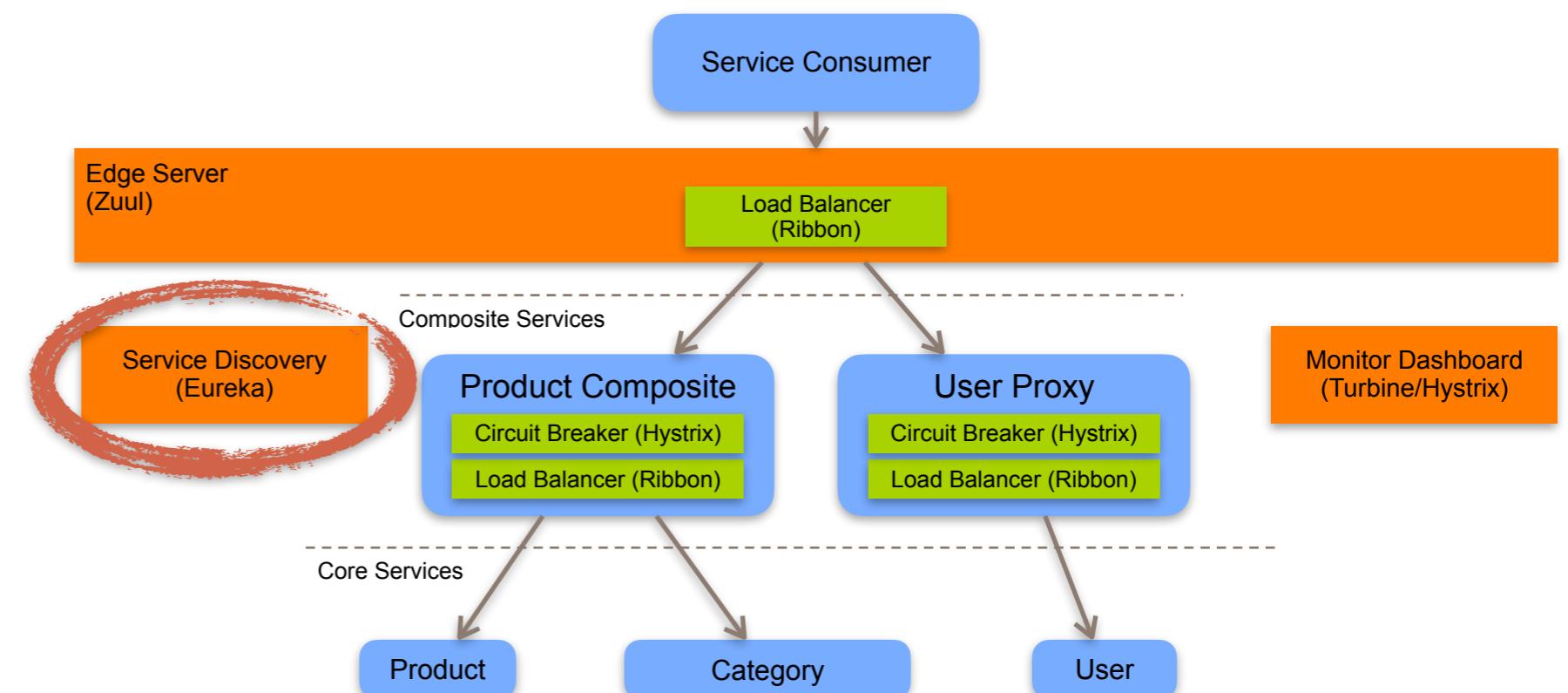
# Microservice Architektur mit Netflix Komponenten



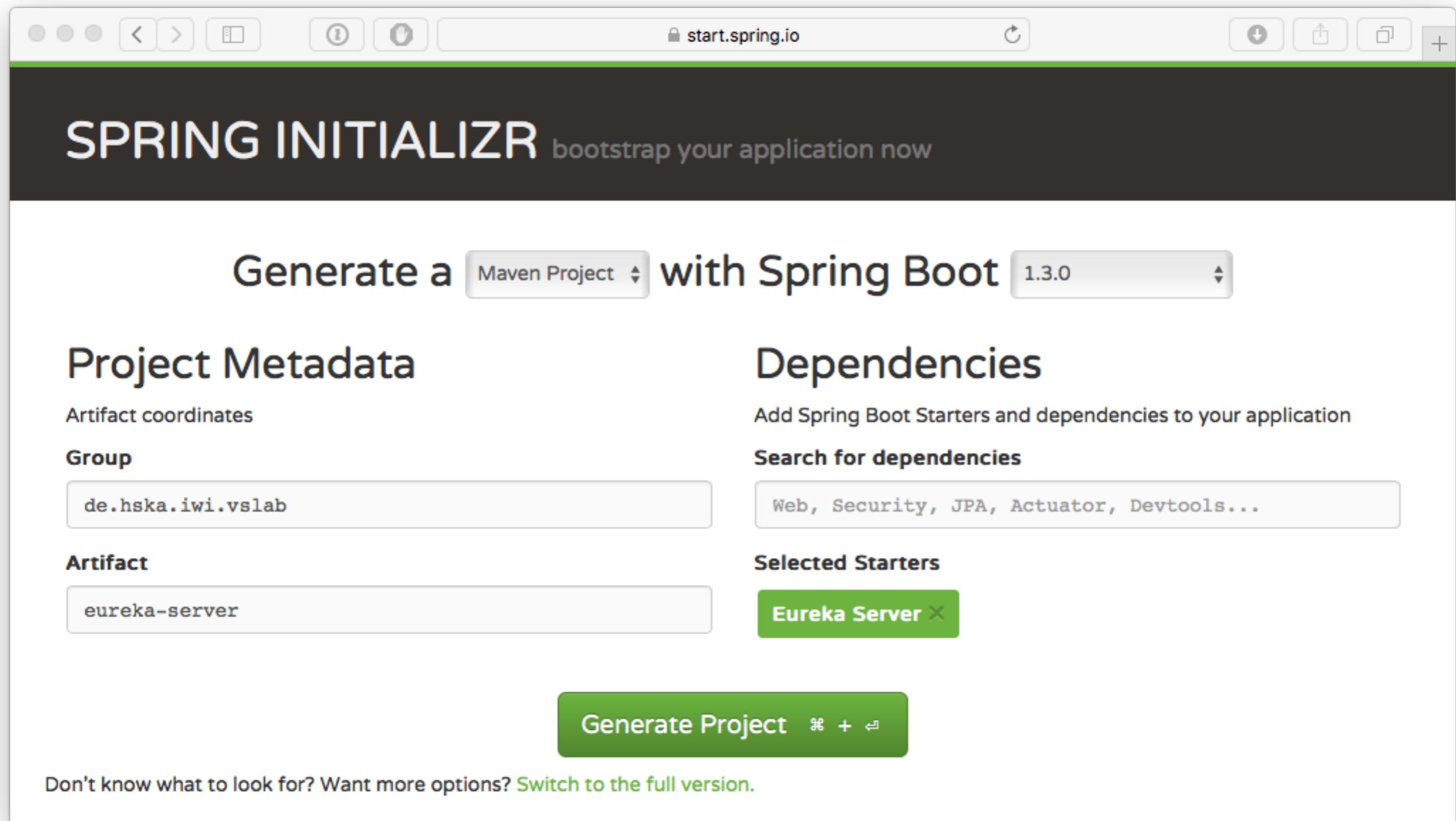
*In Anlehnung an (Larson 2015b)*

# Spring Cloud Netflix Eureka

- Ein *Discovery Server* wird durch *Spring Cloud Eureka* realisiert.
- Der Server wird als *Spring Boot Starter* Projekt erstellt.
- Die "Implementierung" erfolgt rein *deklarativ*.
- Spring Konfiguration erfolgt als *Standalone-Server*.



# Spring Cloud Netflix Eureka Starter



The screenshot shows the Spring Initializr interface at [start.spring.io](https://start.spring.io). The title "SPRING INITIALIZR bootstrap your application now" is displayed. The main heading "Generate a **Maven Project** with Spring Boot **1.3.0**" is centered. On the left, under "Project Metadata", the "Group" field contains "de.hska.iwi.vslab" and the "Artifact" field contains "eureka-server". On the right, under "Dependencies", the "Selected Starters" list includes "Eureka Server". A large green "Generate Project" button is at the bottom.

SPRING INITIALIZR bootstrap your application now

Generate a **Maven Project** with Spring Boot **1.3.0**

**Project Metadata**

Artifact coordinates

Group

de.hska.iwi.vslab

Artifact

eureka-server

**Dependencies**

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Starters

Eureka Server

Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

# Spring Cloud Netflix Eureka Applikation

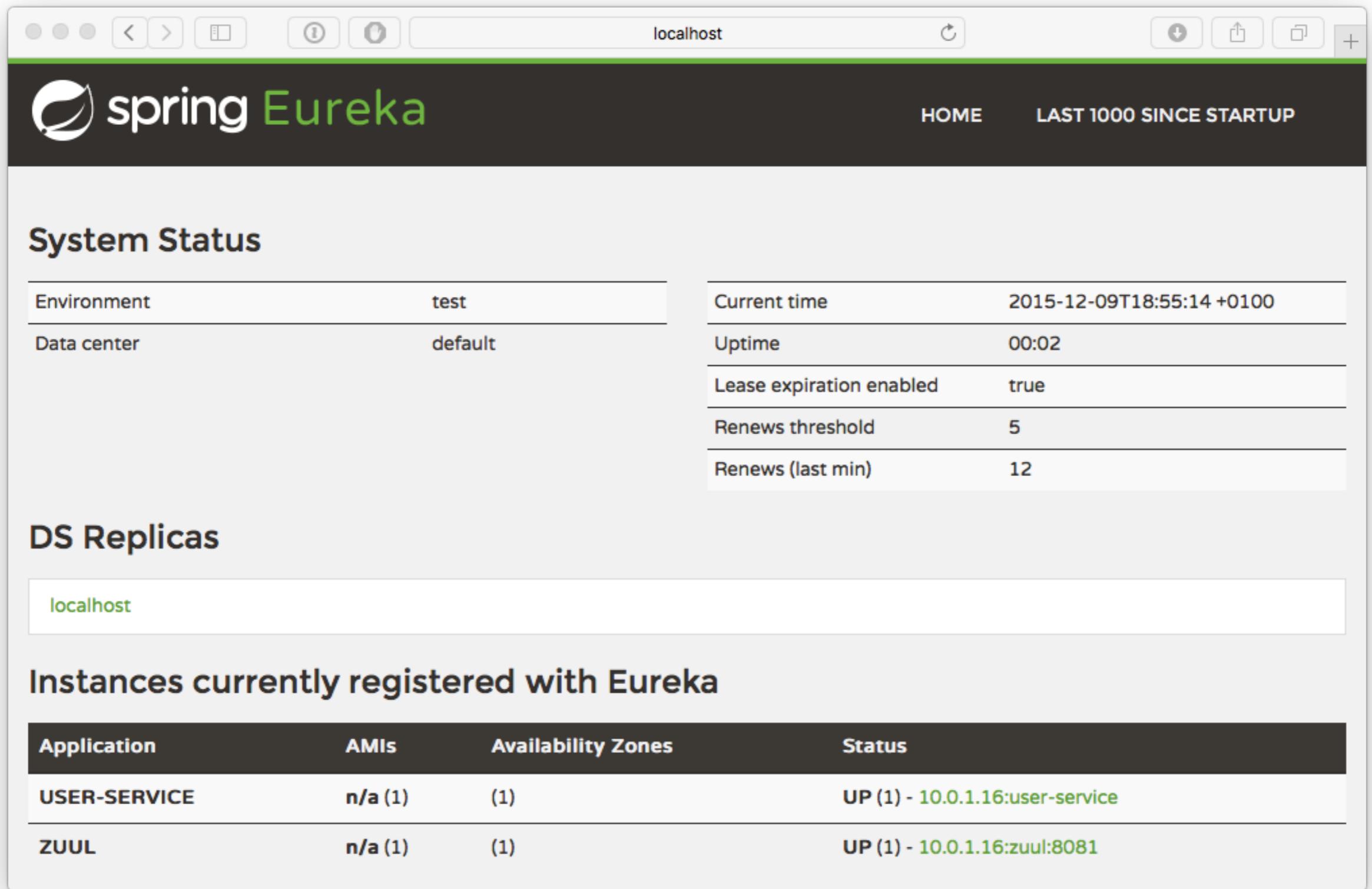
- EurekaApplication ist die **Basisklasse des Eureka Servers**, der als Spring Boot Applikation realisiert ist.
- @EnableEurekaServer aktiviert den Eureka Server.
- Eureka hat auch ein Web Frontend: <http://localhost:8761>

```
package de.hska.iwi.vslab;

import ...

@SpringBootApplication
@EnableEurekaServer
public class EurekaApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaApplication.class, args);
    }
}
```

# Spring Cloud Netflix Eureka Frontend



The screenshot shows the Spring Cloud Netflix Eureka Frontend interface running on localhost. The top navigation bar includes links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into sections: System Status, DS Replicas, and Instances currently registered with Eureka.

**System Status**

Environment	test	Current time	2015-12-09T18:55:14 +0100
Data center	default	Uptime	00:02
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	12

**DS Replicas**

localhost

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
USER-SERVICE	n/a (1)	(1)	UP (1) - 10.0.1.16:user-service
ZUUL	n/a (1)	(1)	UP (1) - 10.0.1.16:zuul:8081

# Spring Cloud Netflix Eureka Konfiguration

- src/main/resources/application.yml

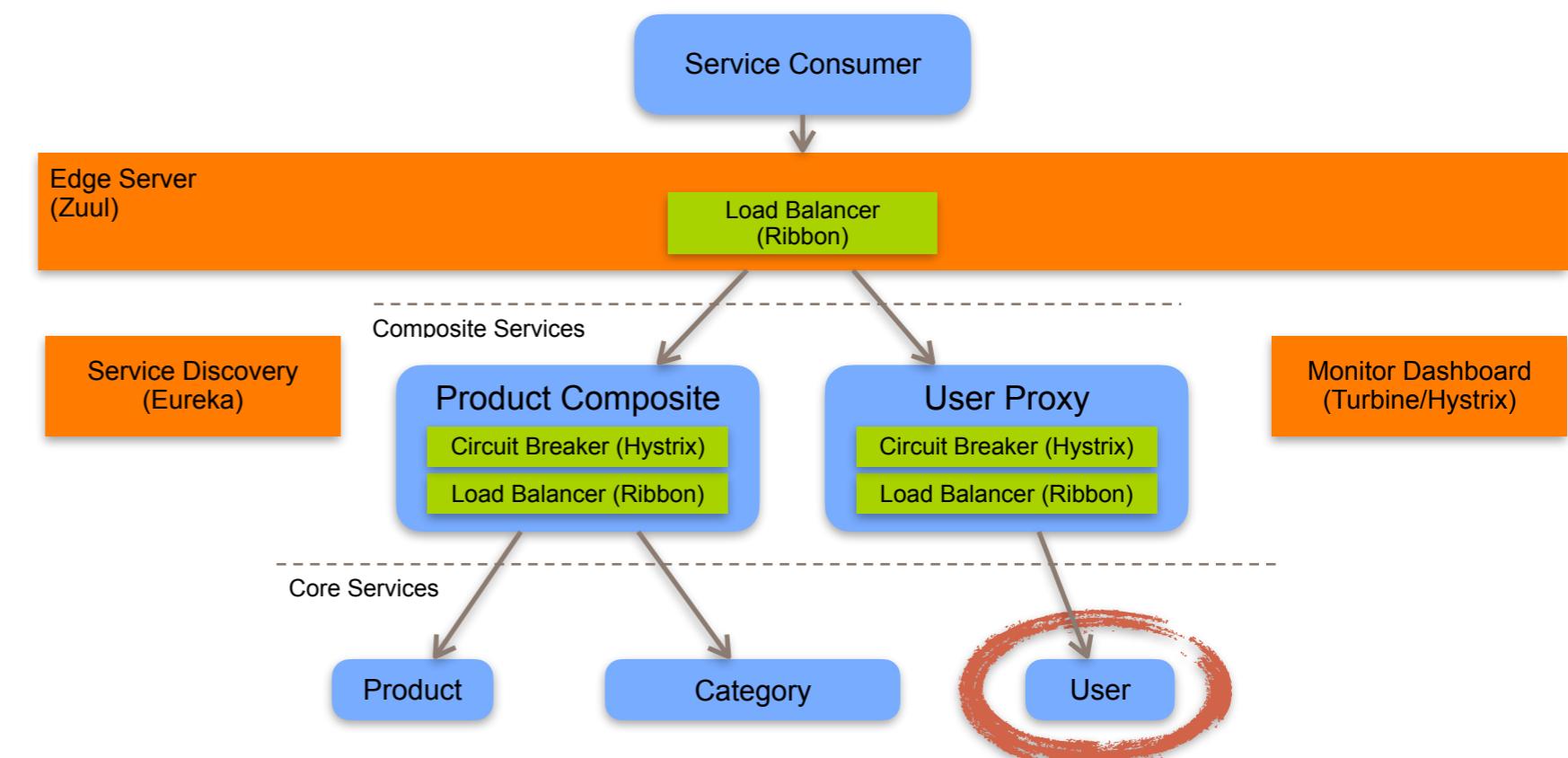
```
server:  
  port: 8761  
  
eureka:  
  client:  
    registerWithEureka: false  
    fetchRegistry: false  
  server:  
    waitTimeInMsWhenSyncEmpty: 0
```

- src/main/resources/bootstrap.yml

```
spring:  
  application:  
    name: eureka  
  cloud:  
    config:  
      enabled: false
```

# Spring Cloud Core Microservice

- Core Microservices bieten ihre Inhalte über *REST Schnittstellen* an.
  - Sie sind keine Clients anderer Microservices.
- Zur Anbindung an die Infrastruktur, registrieren sich Core Microservices beim *Eureka Discovery Server*.
- Zur Implementierung werden *Spring MVC* und *Spring Boot* verwendet.
  - Die Registrierung erfolgt mittels *Spring Cloud Eureka*.



# Spring Cloud Core Microservice Applikation

- UserSrvApplication ist die Basisklasse eines Microservice, der als Spring Boot Applikation realisiert ist.
- @EnableDiscoveryClient aktiviert die Discovery Server Registrierung .
- *Spring Boot Starter* sind **Web**, **JPA** und **Actuator**.
- *Spring Cloud Starter* ist **Eureka Discovery**.

```
package de.hska.iwi.vslab.usrv;  
  
import ...  
  
@SpringBootApplication  
@EnableDiscoveryClient  
public class UserSrvApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(UserSrvApplication.class, args);  
    }  
}
```

# Spring Cloud Core Microservice Konfiguration

- src/main/resources/application.yml

```
server:  
  port: 8080  
spring:  
  application:  
    name: user-service  
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8761/eureka/  
instance:  
  preferIpAddress: true  
  leaseRenewalIntervalInSeconds: 5  
  metadataMap:  
    instanceId: "${spring.application.name}:${random.value}"
```

# Spring Cloud Core Microservice Controller

- Die `@RestController` Annotation weist darauf hin, dass UserController REST Endpunkte hat.
- Die Datenbank UserRepo wird per *Dependency Injection* eingebunden
- Haben wir schon früher im Tutorium behandelt

```
package de.hska.iwi.vslab.usrv;
import ...

@RestController
public class UserController {

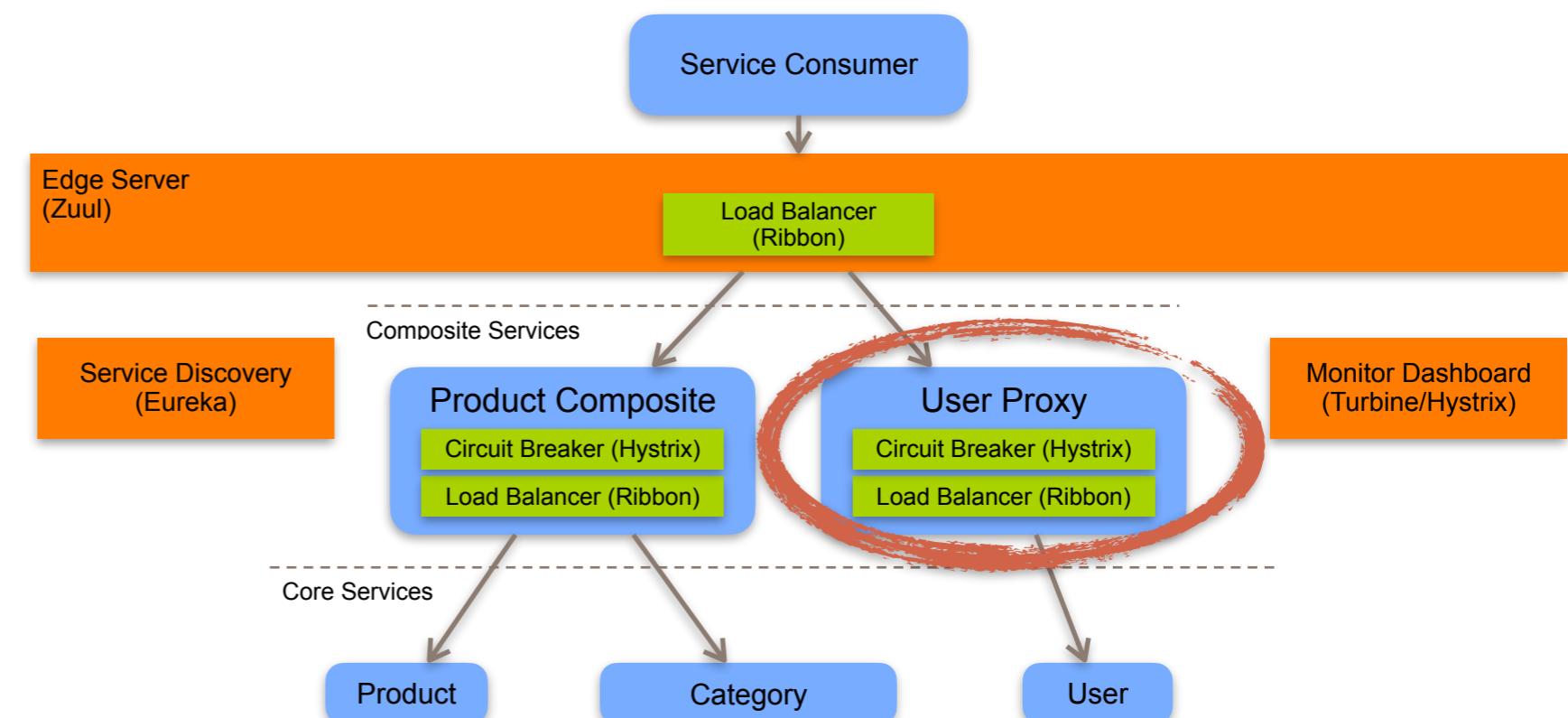
    @Autowired
    private UserRepo repo;

    @RequestMapping(value = "/users/{userId}", method = RequestMethod.GET)
    public ResponseEntity<User> getUser(@PathVariable Long userId) {
        User user = repo.findOne(userId);
        return new ResponseEntity<>(user, HttpStatus.OK);
    }

    ...
}
```

# Spring Cloud Composite Microservice

- Composite Microservices bieten *REST Schnittstellen* auf ihre Inhalte an.
- Sie sind auch *Clients* anderer Microservices.
  - Die Implementierung erfolgt mit **Spring REST Template**.
- Die Implementierung berücksichtigt *dynamische Bindung* und *Resilienz*.
  - Eine Absicherung der Client Funktionen erfolgt als **Hystrix Operationen**.
  - Die Auflösung genutzter Service Endpunkte erfolgt dynamisch per **Ribbon**.
- Der Service bietet ein *Monitoring Frontend* als **Hystrix Dashboard**.



# Spring Cloud Composite Microservice Applikation

- UserProxyApplication ist die Basisklasse eines komplexen Microservice.
  - @EnableDiscoveryClient aktiviert die *Discovery Server Registrierung*.
  - @EnableCircuitBreaker aktiviert *Hystrix als Circuit Breaker*.
  - @EnableHystrixDashboard aktiviert *Hystrix als lokales Monitoring Dashboard*.
  - @RibbonClient aktiviert *Ribbon als Load Balancer*. (oder @LoadBalanced)
- *Spring Boot Starter* sind **Web** und **Actuator**.
- *Spring Cloud Starter* sind **Eureka Discovery**, **Ribbon**, **Hystrix** und **Hystrix-Dashboard**.

```
package de.hska.iwi.vsys.microlab;
import ...

@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker
@EnableHystrixDashboard
@RibbonClient("user-proxy")
public class UserProxyApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserProxyApplication.class, args);
    }
}
```

# Spring Cloud Composite Microservice Konfiguration

- src/main/resources/application.yml

```
server:  
  port: 8088  
spring:  
  application:  
    name: user-proxy  
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8761/eureka/  
instance:  
  preferIpAddress: true  
  leaseRenewalIntervalInSeconds: 5  
  metadataMap:  
    instanceId: "${spring.application.name}:${random.value}"
```

# Spring Cloud Microservice Client

- Per `@HystrixCommand` werden *zu sichernde Operationen* deklariert.
  - Es können *Schwellenwerte* und *alternative Operationen* definiert werden.
- Im URI des REST-Templates wird *der erste Teil des hierarchischen Namens* von Ribbon per Eureka aufgelöst.

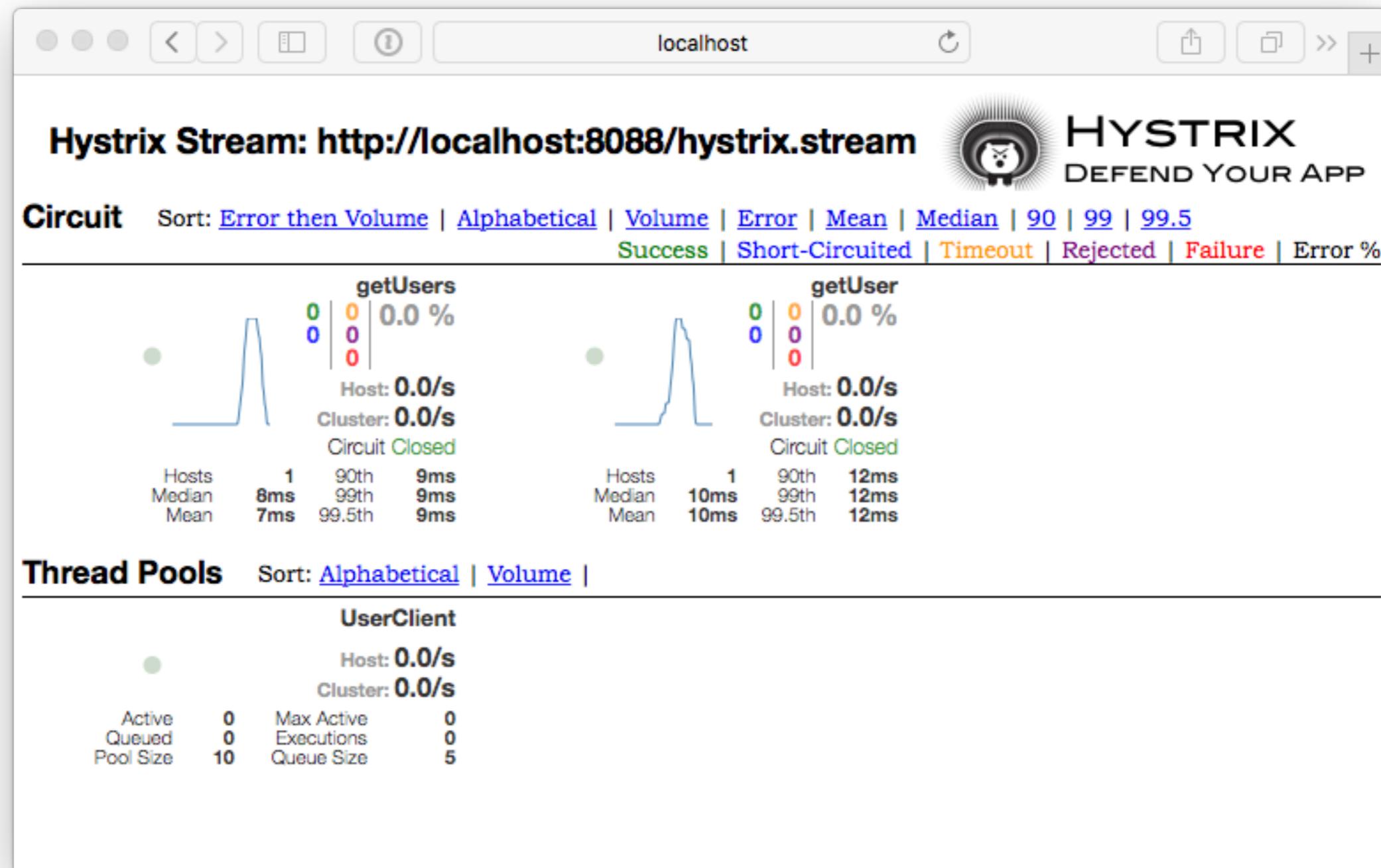
```
package de.hska.iwi.vsys.microlab;
import ...
@Component
public class UserClient {
    private final Map<Long, User> userCache = new LinkedHashMap<Long, User>();
    @Autowired
    private RestTemplate restTemplate;

    @HystrixCommand(fallbackMethod = "getUserCache", commandProperties = {
        @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "2") })
    public User getUser(Long userId) {
        User tmpuser =
            restTemplate.getForObject("http://user-service/users/" + userId, User.class);
        userCache.putIfAbsent(userId, tmpuser);
        return tmpuser;
    }

    public User getUserCache(Long userId) {
        return userCache.getOrDefault(userId, new User());
    }
}
```

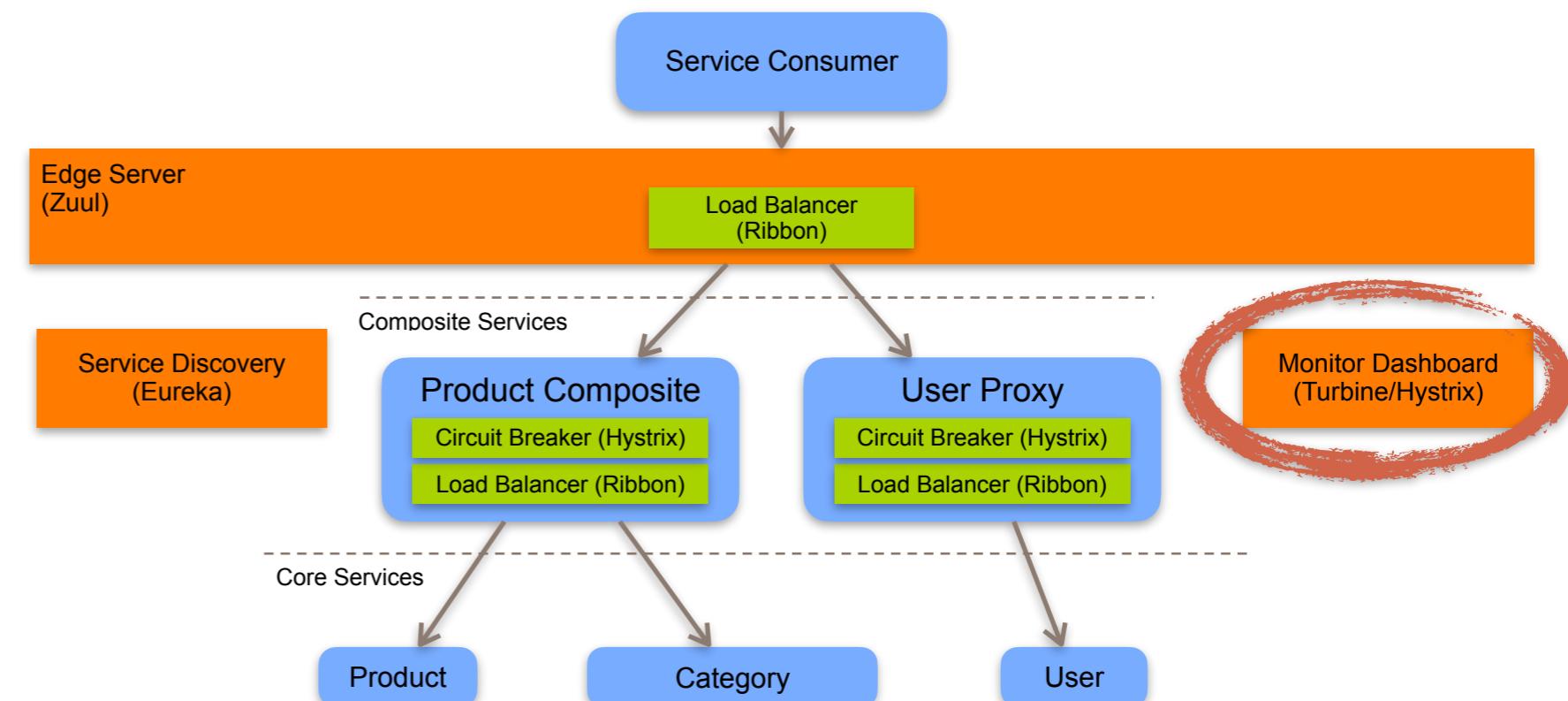
# Spring Cloud Hystrix Dashboard

- Hystrix *Dashboard*: <http://localhost:8088/hystrix>
- Hystrix *Event Stream*: <http://localhost:8088/hystrix.stream>



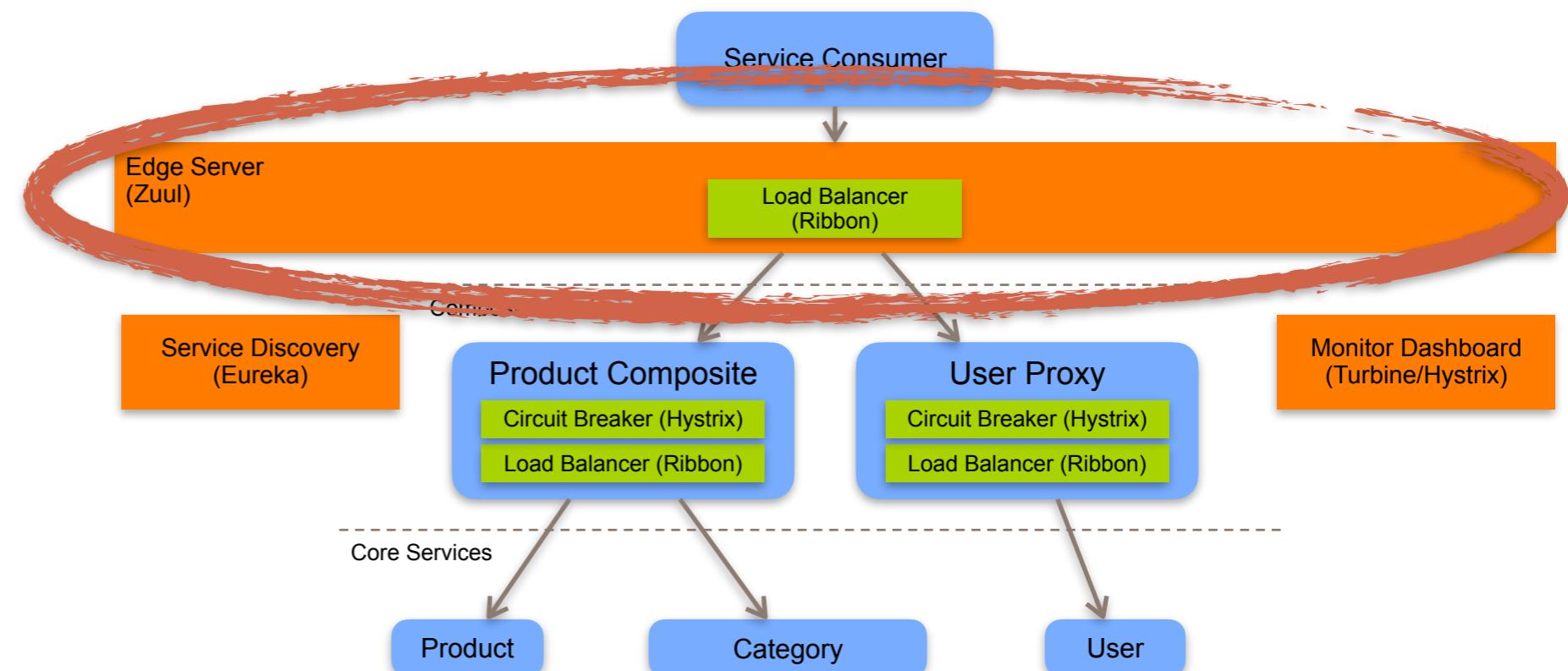
# Spring Cloud Netflix Turbine

- Ein *Turbine Server* kann Event Streams z.B. von *Hystrix* abfragen.
  - Turbine kann dabei auch Eureka für das Discovery von Streams nutzen.
- Die abgefragten Streams können *aggregiert* werden und stehen als *komplexer Event Stream* wieder zur Verfügung.
- Der Turbine Server wird als einfache *Spring Boot Anwendung* realisiert und dort mit der Annotation `@EnableTurbine` aktiviert.
- Der Turbine Server kann auch ein *Hystrix Dashboard* einschließen.



# Spring Cloud Netflix Zuul

- Der *Zuul Edgeserver* ist der *technische Zugangspunkt* der Microservice-basierten Anwendung für externe Clients.
- Zuul arbeitet dabei als *Router* und *serverseitiger Load Balancer*.
- Zuul kann auch die *Authentifizierung* von Clients übernehmen.
- In Spring Cloud ist Zuul als Reverse Proxy vorkonfiguriert.
  - Zuul nutzt dann *Ribbon als Load Balancer* und
  - in URLs wird der erste Name im Pfad zum *Lookup in Eureka* verwendet.



# Spring Cloud Netflix Zuul Applikation

- ZuulApplication ist die **Basisklasse des Zuul Servers**, der als Spring Boot Applikation realisiert ist.
  - @EnableZuulProxy aktiviert Zuul als Proxy.
- *Spring Boot Starter* ist **Actuator**.
- *Spring Cloud Starter* sind **Eureka Discovery** und **Zuul**.

```
package de.hska.iwi.vsmlab;

import ...

@SpringBootApplication
@EnableZuulProxy
public class ZuulApplication {
    public static void main(String[] args) {
        SpringApplication.run(ZuulApplication.class, args);
    }
}
```

# Spring Cloud Netflix Zuul Konfiguration

## ■ src/main/resources/application.yml

```
server:  
  port: 8081  
  
eureka:  
  instance:  
    leaseRenewalIntervalInSeconds: 10  
    statusPageUrlPath: /info  
    healthCheckUrlPath: /health  
  
zuul:  
  ignoredServices: '*'  
  routes:  
    user-proxy: /user-api/**
```

Routing Regeln verstecken  
die internen Microservices.

## ■ src/main/resources/bootstrap.yml

```
spring:  
  application:  
    name: zuul  
  cloud:  
    config:  
      enabled: false  
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8761/eureka/
```

# Struktur — Microservice Middleware

## Microservice-basierte Systeme

- Charakteristik von Microservice-basierten Systemen
- Herausforderungen im Betrieb
- Microservice Middleware

## Spring Cloud Framework

- Spring (Cloud (Netflix))
- Netflix-basierte Systemarchitektur

## Laboraufgabe

# Aufgabe 3.2/3 Middleware Integration

- **3.2.a** Als Basis der Infrastruktur soll ein *Eureka Server als Spring Boot Anwendung* realisiert werden. Alle implementierten Microservices sollen so erweitert werden, dass sie sich als *Discovery Clients* registrieren.
- **3.2.b** Composite Microservices sollen zu *Ribbon-Clients* erweitert werden, die die verwendeten Core Microservices dynamisch Binden. Der Zugriff auf die Core Microservices soll *per Hystrix gesichert* werden.
- **3.2.c** Es soll ein *Zuul Edge Server als Spring Boot Anwendung* realisiert werden. Der Edge Server soll eine *öffentliche API* bereitstellen, die die Client Aufrufe auf die implementierten API Gateways leitet.
- **3.2.d** Es soll ein *Hystrix Dashboard als Spring Boot Anwendung* (oder als Teil einer geeigneten anderen Komponente) realisiert werden. Auf dem Dashboard sollen die Aktivitäten des Edge Servers und der Hystrix-gesicherten Komponenten beobachtet werden.

**Aufgabe 3.3** Alle Spring Boot Prozesse sollen in dedizierten **Docker Containern** bereitgestellt werden (sowohl Microservices als auch Middleware Services). Die Container sollen mittels **Docker Compose** in Beziehung gesetzt und gemeinsam gesteuert werden.

# Literatur und Web Ressourcen

## Literatur

[Wolff 2015] Eberhard Wolff, "Microservices, Grundlagen flexibler Softwarearchitekturen", dpunkt, November 2015

## Aus dem Web (abgerufen am 14.5.2019)

[Larson 2015a] Magnus Larson, "An operations model for Microservices", <http://callistaenterprise.se/blogg/teknik/2015/03/25/an-operations-model-for-microservices/>

[Larson 2015b] Magnus Larson, „Building microservices with Spring Cloud and Netflix OSS, part 1“, <http://callistaenterprise.se/blogg/teknik/2015/04/10/building-microservices-with-spring-cloud-and-netflix-oss-part-1/>

[Spring 2017a] Spring Cloud, "Spring Cloud", <https://spring.io/projects/spring-cloud>

[Netflix 2017] Netflix Open Source Software Center, <https://netflix.github.io>

## Demo Source

<https://github.com/zirpins/vsmlab.git>