

Benjamin Swarovsky

Technologische Ansätze zur
Umsetzung einer
Microservice-Architektur




Prototypische Implementierung einer
Anwendung zur Verwaltung der IT-
Kontaktmesse an der Fachhochschule Erfurt

Gliederung

- Einleitung
- Architektur
- API Gateway
- Service Discovery
- Load Balancer
- Zusammenspiel der Technologien
- Fazit
- Demonstration des Prototypen

- **Einleitung**
 - **Problemstellung**
 - **Ziele**
- Architektur
- API Gateway
- Service Discovery
- Load Balancer
- Zusammenspiel der Technologien
- Fazit
- Demonstration des Prototypen

Problemstellung

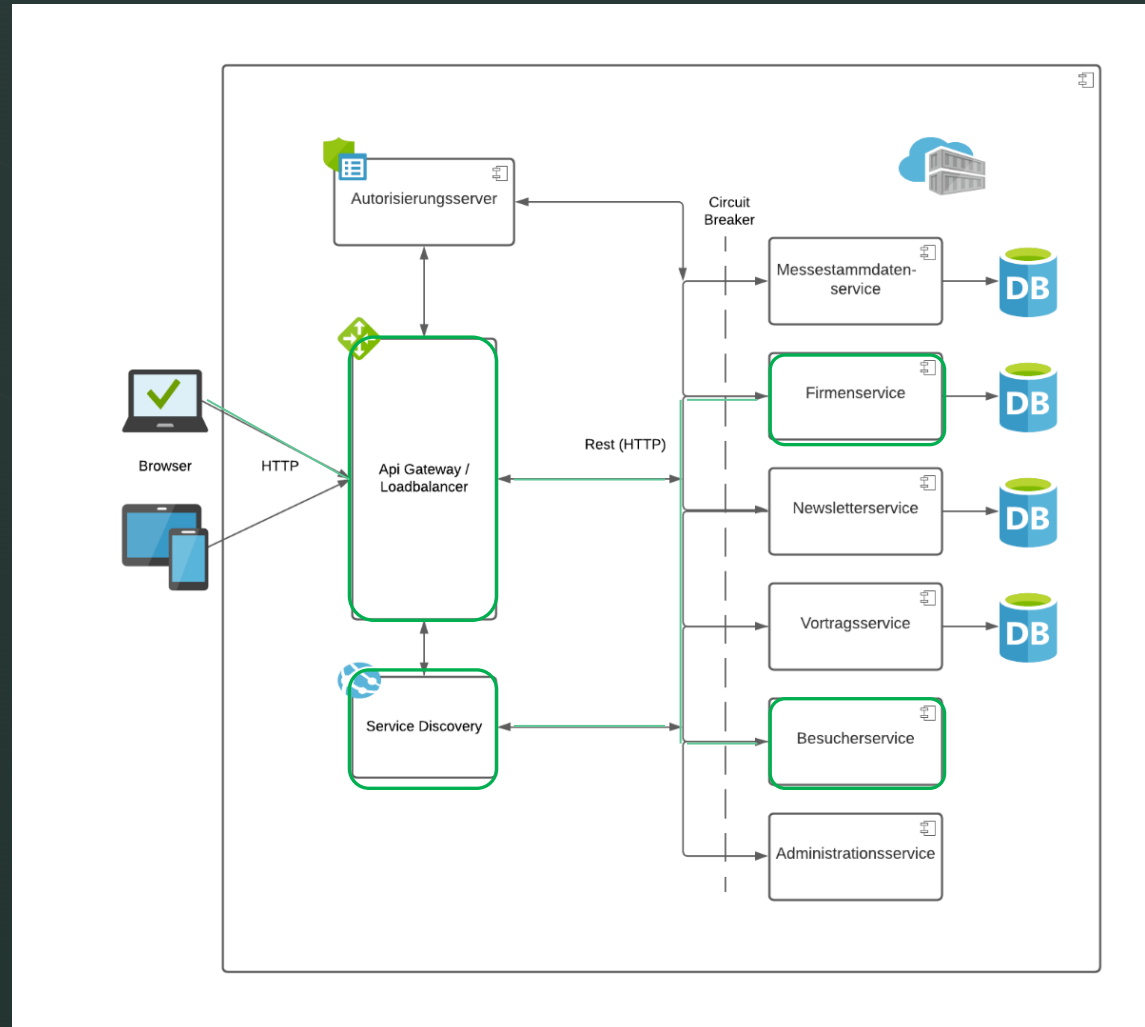
- Monolith:
 - Enge Kopplung 
 - Schlecht skalierbar 
 - Big Ball of Mudd 
- Microservices 
- Anspruchsvolle Architektur 
- Herausforderungen beim Einsatz mehrerer Services:
 - Konsistenz
 - Kommunikation
 - Fehlerbehandlung

Ziele

- Technologische Ansätze zur Umsetzung von Microservices (Frameworks, Bibliotheken, Algorithmen, ...)
- Beispielanwendung (Verwaltungsprogramm für die IT-Kontakmesse)
- Implementierung der Technologischen Ansätze (Springboot, Eureka, Jaeger, ...)
- Auswertung
 - Komplexität
 - Welche Probleme traten auf
 - Wie wichtig sind einzelne Technologien für die Umsetzung von Microservices

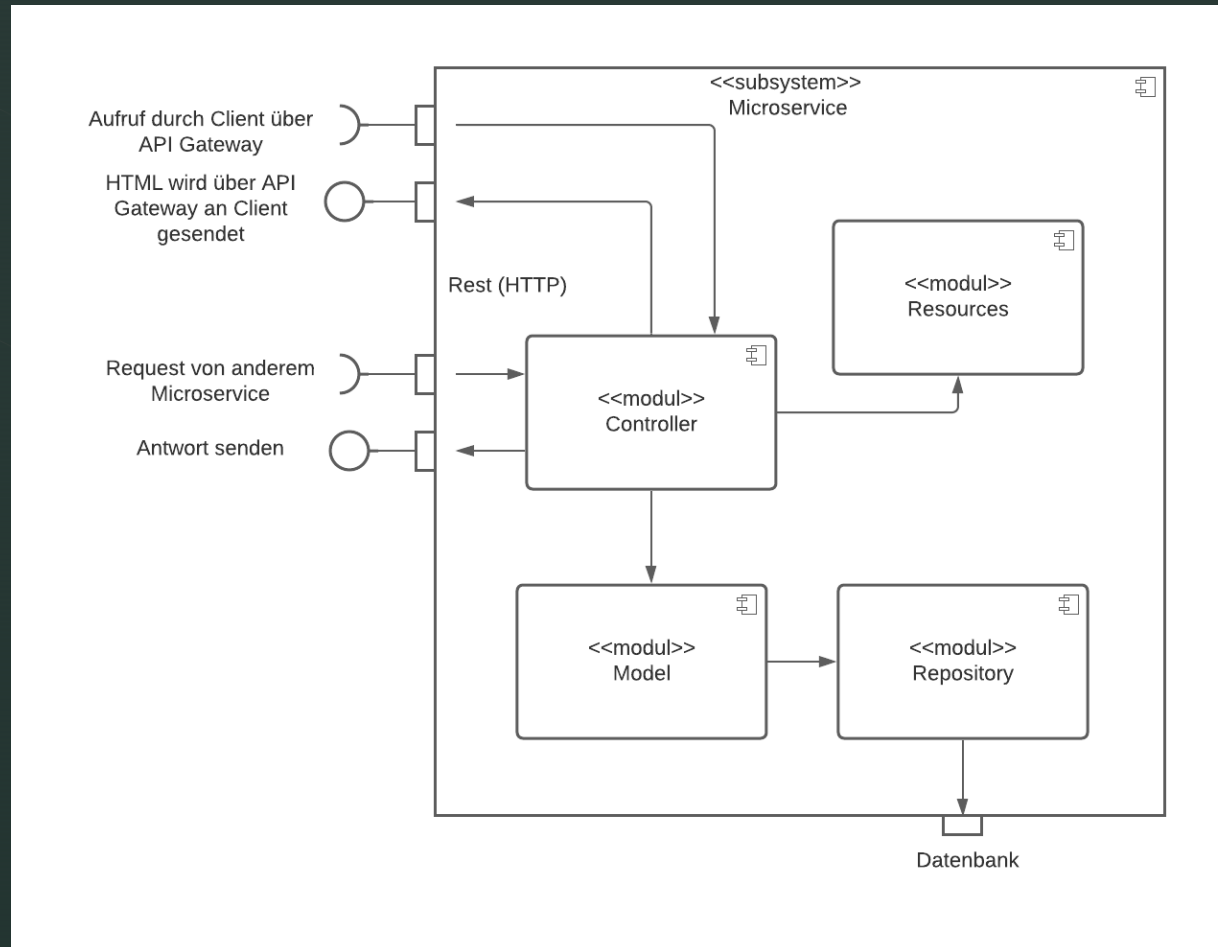
- Einleitung
- **Architektur**
 - **Bausteinsicht Ebene 1**
 - **Bausteinsicht Ebene 2**
- API Gateway
- Service Discovery
- Load Balancer
- Zusammenspiel der Technologien
- Fazit
- Demonstration des Prototypen

Bausteinsicht Ebene 1



Quelle: eigene Darstellung

Bausteinsicht Ebene 2

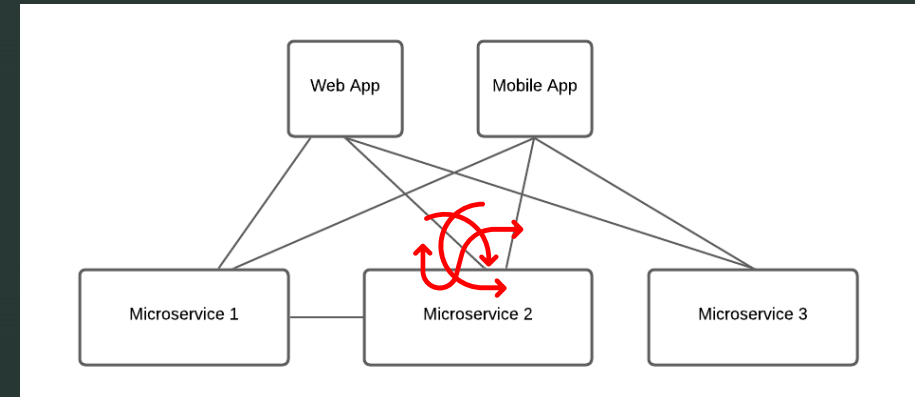


Quelle: eigene Darstellung

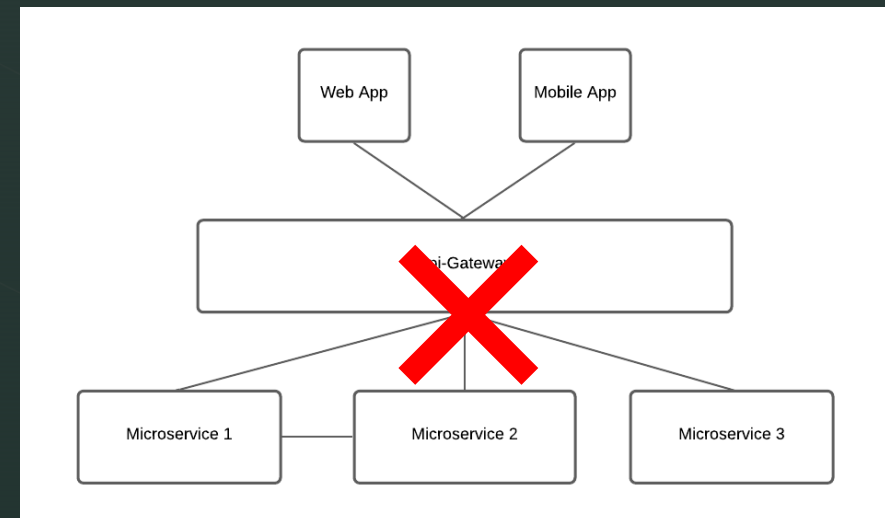
- Einleitung
- Architektur
- **API Gateway / Load Balancer**
 - **API Gateway**
 - **Load Balancer**
 - **Spring Cloud API Gateway**
 - **Implementierung**
- Service Discovery
- Zusammenspiel der Technologien
- Fazit
- Demonstration des Prototypen

API Gateway - Grundlagen

- Direkte Kommunikation (Client - Services)
 - Sicherheitsprobleme
 - Enge Kopplung
- Kommunikation über Gateway
 - Kontaktpunkt für Ein/-Ausgehenden Netzverkehr
 - Autorisierung & Authentifizierung
 - Zentrales Logging
 - Nachteil: Ausfall des Gateways zieht den Ausfall des gesamten Systems nach sich



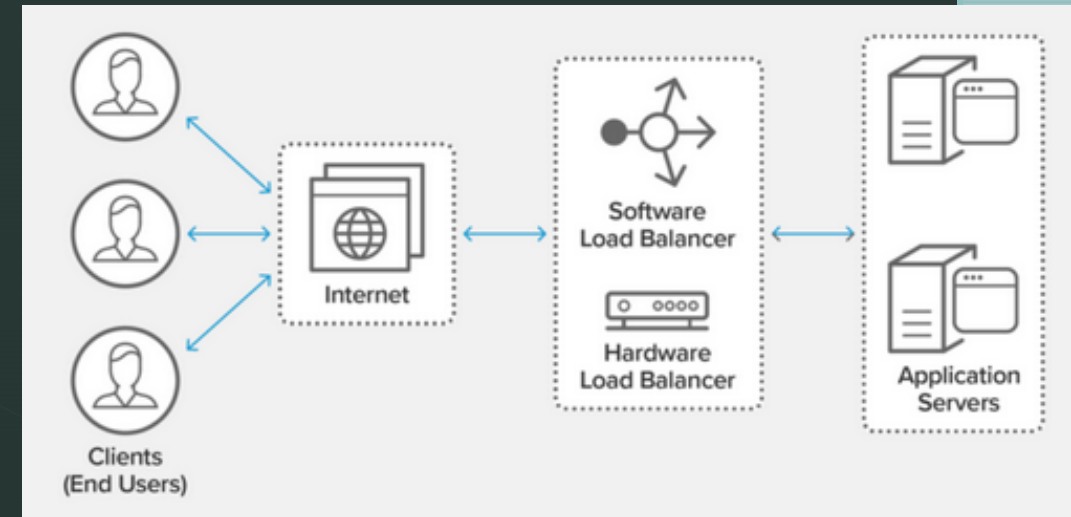
Quelle: eigene Darstellung



Quelle: eigene Darstellung

Load Balancer

- Setzt Lastverteilung in einem Netzwerk um
- Ermittelt welche Rechenressource die Clientanforderung erfüllen kann
- Realisiert als Software- oder Hardware-Load Balancer
- Nutzt Algorithmen wie zum Beispiel: Round Robin



Quelle: NGINX, <https://www.nginx.com/resources/glossary/load-balancing/> (07.09.2021)

Spring Cloud API Gateway

- Basiert auf asynchronen eventgetriebenen Framework Netty
- Features:
 - Ribbon Load Balancer
 - Sicherheitskonfigurationsmöglichkeiten mit Spring Security
 - Einbindung eigener Filter
- Einfache Integration in Spring Anwendung
- Alternativen: Ocelot, KrakenD, Kong

API Gateway - Implementierung



Quelle: spring.io,
<https://spring.io/projects/spring-boot> (06.11.2021)



Quelle: Maven,
<https://maven.apache.org/> (06.11.2021)

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

Quelle: eigene Darstellung

```
server.port=8081
spring.application.name=ApiGateway

spring.cloud.gateway.routes[2].id=lbFirmenservice      //id der Route

spring.cloud.gateway.routes[2].uri=lb://firmenservice /* Name des aufzurufenden -
                                                    Services
                                                    (für Loadbalancing) */

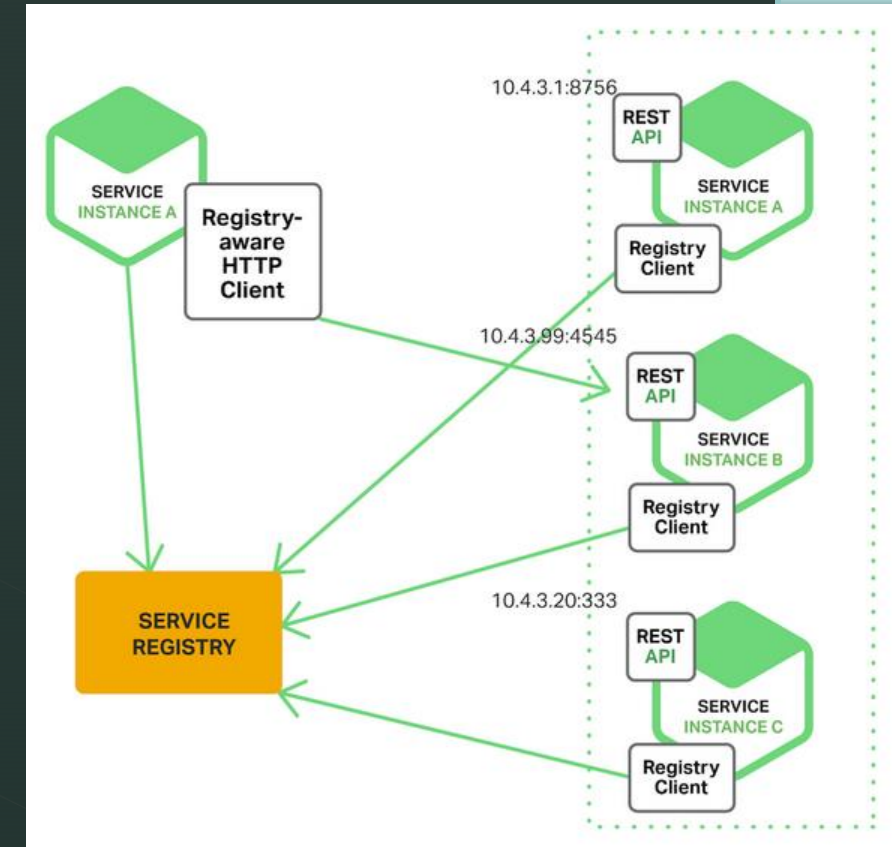
spring.cloud.gateway.routes[2].predicates[0]=Path=/** /* Pfad unter dem
                                                    Loadbalancing
                                                    Erfolgen soll */
```

Quelle: eigene Darstellung

- Einleitung
- Architektur
- API Gateway / Load Balancer
- **Service Discovery**
 - Grundlagen
 - Eureka
 - Implementierung (Eureka-Server)
 - Implementierung (Eureka-Client)
- Zusammenspiel der Technologien
- Fazit
- Demonstration des Prototypen

Service Discovery - Grundlagen

- Services mit dynamisch zugewiesenen Netzwerkstandorten können sich gegenseitig finden
- Services registrieren sich an einer Registry
- Adressauflösung kann über Namen des Services erfolgen
- Unter anderem Clientseitige und Serverseitige Discovery



Quelle: NGINX, <https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/> (07.09.2021)

Eureka Discovery Service

- Clientseitige Service Discovery
- Leicht über Spring zu integrieren
- Alternativen sind zum Beispiel NGINX (serverseitig) oder Zookeeper (clientseitig)

System Status

Environment	N/A	Current time	2021-10-31
Data center	N/A	Uptime	00:13
		Lease expiration enabled	true
		Renews threshold	10
		Renews (last min)	12

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
APIGATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-R5UR9R2:ApiGateway:8081
BESUCHERSERVICE	n/a (1)	(1)	UP (1) - Besucherservice:c984498f-d278-4c49-ac60-aeda8517de80
FIRMENSERVICE	n/a (2)	(2)	UP (2) - Firmenservice:7476cd7c-0024-4804-a4a2-1355742e5c80 , Firmenservice:d8f5b0a9-872
VORTRAGSSERVICE	n/a (1)	(1)	UP (1) - Vortragsservice:a0383f69e095e4e7ff7fa0bc616c92f4

Quelle: eigene Darstellung

Service Discovery - Implementierung (Eureka-Server)

- Annotation `@EnableEurekaServer` in Main-Klasse
- Abhängigkeit `spring-cloud-starter-netflix-eureka-server` in `pom.xml`



Quelle: [spring.io](https://spring.io/projects/spring-boot),
<https://spring.io/projects/spring-boot> (06.11.2021)

```
server.port=8010

spring.application.name=discovery-service

eureka.client.register-with-eureka=false // verhindert die Registrierung des -
eureka.client.fetch-registry=false      // Servers mit sich selbst

eureka.client.serviceUrl.defaultZone = http://localhost:8010/eureka /*URL Eureka-Server */
```

Quelle: eigene Darstellung

Service Discovery Implementierung (Eureka-Client)

- Annotation `@EnableEurekaClient` in Main-Klasse
- Abhängigkeit `spring-cloud-starter-netflix-eureka-client` in `pom.xml`



Quelle: [spring.io](https://spring.io/projects/spring-boot),
<https://spring.io/projects/spring-boot> (06.11.2021)

```
Server.port = ${PORT:0} //Port wird automatisch festgelegt  
  
spring.application.name = firmenservice  
  
eureka.instance.instance-id=${spring.application.name}:${random.uuid} //Instanz-Id  
  
eureka.client.serviceUrl.defaultZone = http://localhost:8010/eureka
```

Quelle: eigene Darstellung

- Einleitung
- Architektur
- API Gateway / Load Balancer
- Service Discovery
- **Zusammenspiel der Technologien**
 - **Anwendungsfall**
 - **Ablauf**
- Fazit
- Demonstration des Prototypen

Anwendungsfall

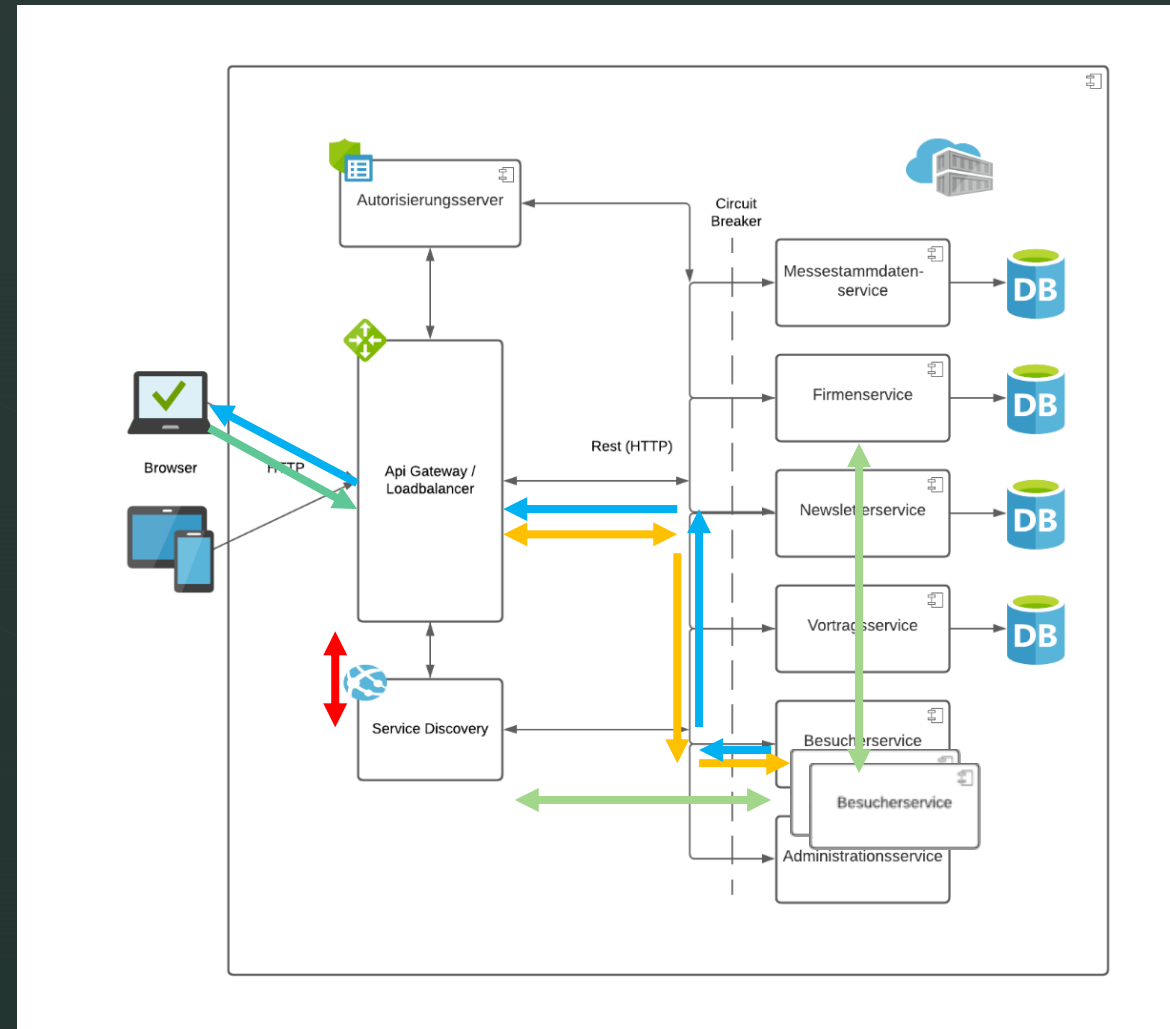
- Ausgabe der teilnehmenden Firmen über den Besucherservice
- Der Besucherservice wurde mehrfach instanziiert

Teilnehmende Firmen für die Messe	
Firmenname:	Firma 1
Webseite:	www.firma_1.de
Firmenname:	Firma 2
Webseite:	www.firma_2.de
Firmenname:	Firma 3
Webseite:	www.firma_3.de

Quelle: eigene Darstellung

Ablauf

1. Aufruf der URL :
localhost:8081/besucherservice/firmen
2. Adressauflösung „besucherservice“ über Eureka
3. Weiterleitung des Aufrufes über Gateway und Loadbalancer (Loadbalancer wählt verfügbaren Besucherservice)
4. Besucherservice ruft den Endpunkt „/firmen“ auf wodurch die Firmendaten vom Firmenservice angefordert werden (Adressauflösung über Eureka)
5. Besucherservice Sendet Die Daten (HTML) über das Gateway zum Client



Quelle: eigene Darstellung

- Einleitung
- Architektur
- API Gateway / Load Balancer
- Service Discovery
- Zusammenspiel der Technologien
- **Fazit**
- Demonstration des Prototypen

Fazit

- Service Discovery
 - Einfache Implementierung mit Eureka
 - Verbessert Wartbarkeit (ermöglicht dynamische Adresszuweisung)
 - Ist Voraussetzung für den Einsatz des Load Balancers
 - nicht zwingend erforderlich (bei wenigen Microservices)
- API Gateway
 - nicht zwingend erforderlich (bei sehr einfachen Anwendungen)
 - Verbessert Wartbarkeit (Zentralisiert API-Management)
 - Einfache Implementierung mit Spring Cloud API Gateway
- Load Balancer
 - Nur notwendig wenn Microservices mehrfach instanziiert werden
 - Trägt zur Erhöhung der Zuverlässigkeit bei (Lastenausgleich)
 - Einfache Implementierung über Spring Cloud API Gateway mit Ribbon

Ende des Vortrages

Quellen

- **Gnatyk , Romana . 2018.** N-iX. [Online] 03. Oktober 2018. [Zitat vom: 07. August 2021.] <https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/>.
- **ComputerWeekly, Redaktion. 2020.** ComputerWeekly. [Online] Juli 2020. [Zitat vom: 16. August 2021.] <https://www.computerweekly.com/de/definition/Load-Balancing>.
- **Donner, Andreas Dipl. -Ing und Kunkel, Richard. 2019.** Ip Insider. [Online] 06. März 2019. [Zitat vom: 21. August 2021.] <https://www.ip-insider.de/die-vorteile-des-software-load-balancings-a-801344/>.
- **Anil, Nish. 2021.** docs.microsoft. [Online] 28. September 2021. [Zitat vom: 24. August 2021.] <https://docs.microsoft.com/de-de/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>.
- **baeldung. 2021.** Baeldung. [Online] 07. April 2021. [Zitat vom: 10. Oktober 2021.] <https://www.baeldung.com/maven>.
- **Bayer, Thomas. 2019.** predic8. [Online] 18. November 2019. [Zitat vom: 04. September 2021.] <https://www.predic8.de/microservices-spring-boot-spring-cloud.htm>.
- **Gillis, Alexander S. 2021.** TechTarget. [Online] April 2021. [Zitat vom: 14. August 2021.] <https://whatis.techtarget.com/de/definition/Service-Discovery-Diensterkennung>.

Quellen

- **Hruschka, Peter und Starke, Gernot. 2017.** *arc42 template*. Januar 2017.
- **laverma. 2020.** La Verma. [Online] 16. September 2020. [Zitat vom: 29. August 2021.] <https://lalverma.medium.com/spring-boot-microservices-api-gateway-e9dbcd4bb754>.
- **NGINX.** NGINX. [Online] [Zitat vom: 07. September 2021.] <https://www.nginx.com/resources/glossary/load-balancing/>.
- **RedHat.** Red Hat. [Online] [Zitat vom: 20. August 2021.] <https://www.redhat.com/de/topics/api/what-is-a-rest-api>.
- **—. 2015.** NGINX. [Online] 12. Oktober 2015. [Zitat vom: 20. August 2021.] <https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/>.
- **sidion. 2019.** sidion. [Online] 28. Juni 2019. [Zitat vom: 25. August 2021.] <https://www.sidion.de/lernen/sidion-labor/blog/spring-cloud-gateway.html>.
- **Wellner, Michael. 2019.** sidion. [Online] 28. Juni 2019. [Zitat vom: 29. August 2021.] <https://www.sidion.de/lernen/sidion-labor/blog/spring-cloud-gateway.html>.
- **—. 2020.** Dev Insider. [Online] 03. Juli 2020. [Zitat vom: 19. August 2021.] <https://www.dev-insider.de/was-ist-ein-framework-a-938758/>.

Demonstration des Prototypen