Lesson 9

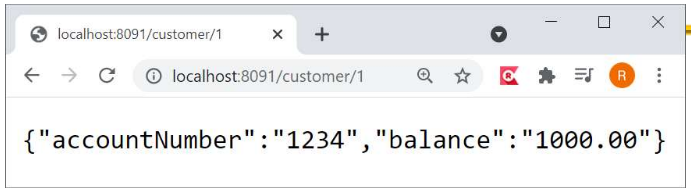
# MICROSERVICES

## Challenges of a microservice architecture

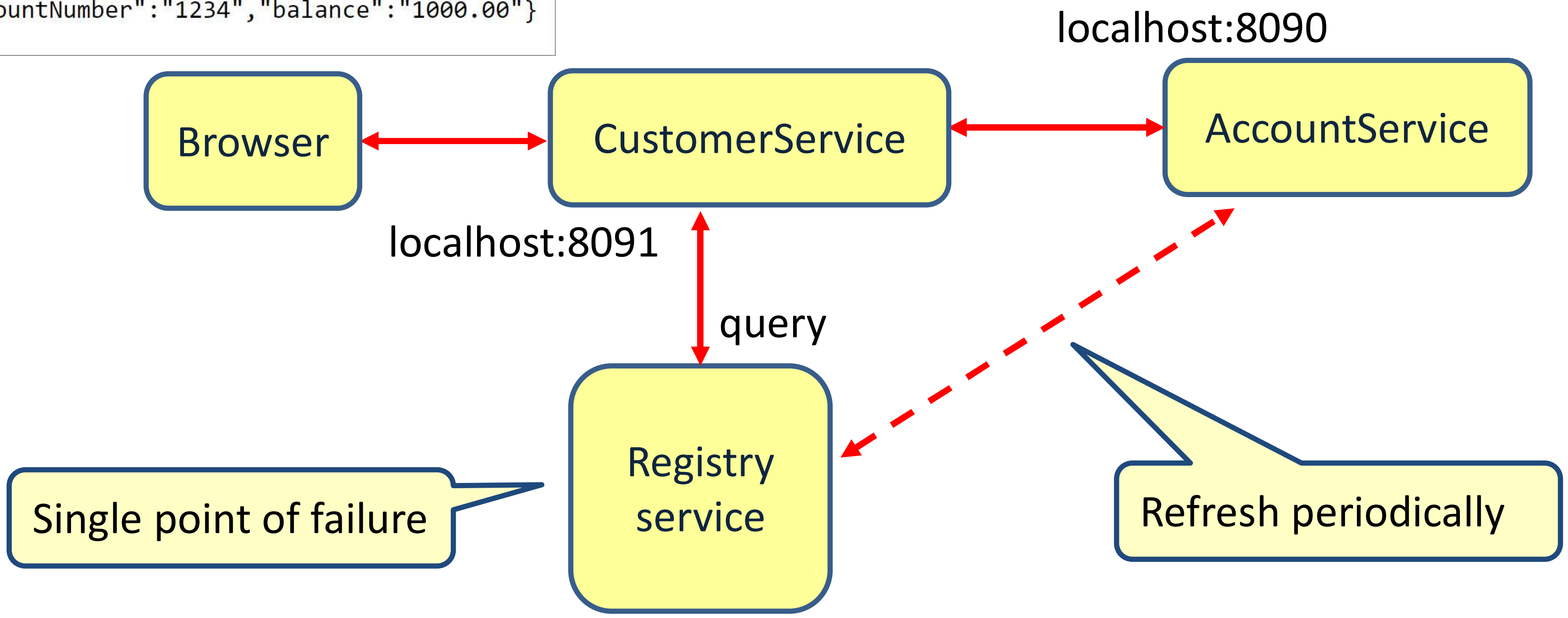
| Challenge | Solution |
|---|---|
| Complex communication | Feign<br>Registry |
| Performance | |
| Resilience | |
| Security | |
| Transactions | |
| Following the process | |
| Keep data in sync | |
| Keep interfaces in sync | |
| Keep configuration in sync | |
| Monitor health of microservices | |
| Follow/monitor business processes | |

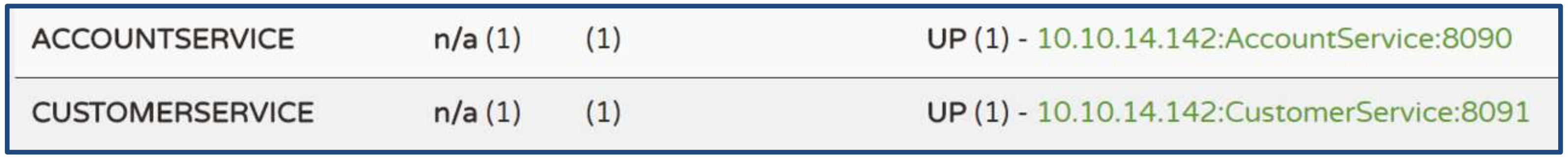
## EUREKA FAILOVER

# Eureka registry



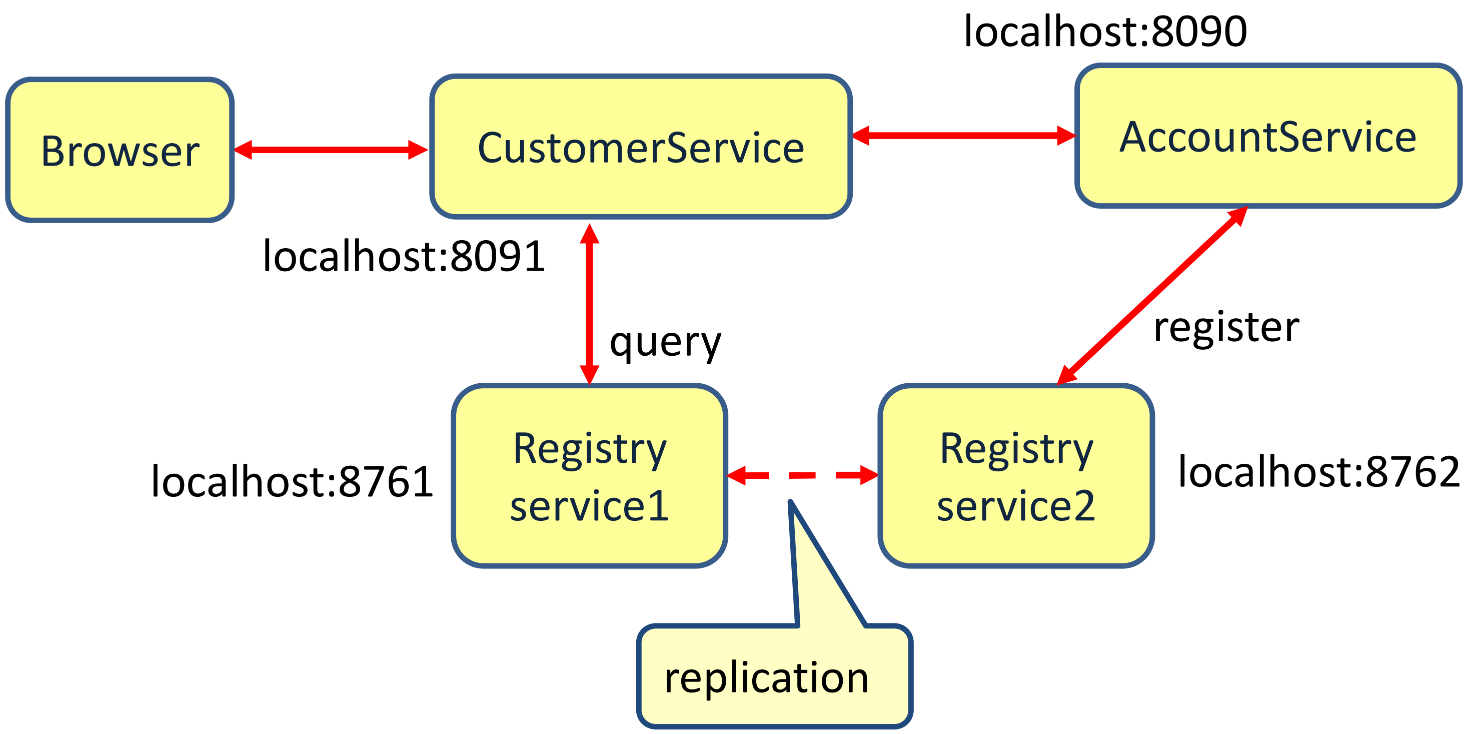
```
{"accountNumber":"1234","balance":"1000.00"}
```

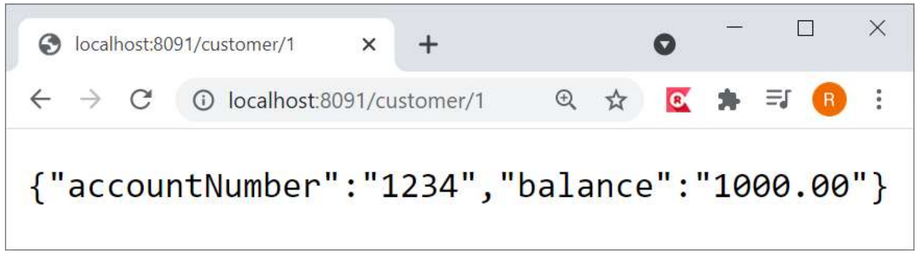




| | | | |
|---|---|---|---|
| ACCOUNTSERVICE | n/a (1) | (1) | UP (1) - 10.10.14.142:AccountService:8090 |
| CUSTOMERSERVICE | n/a (1) | (1) | UP (1) - 10.10.14.142:CustomerService:8091 |

# Registry replication





```
{"accountNumber":"1234","balance":"1000.00"}
```

# Eureka replicas

eurekaserver1

```
server:
  port: 8761

eureka:
  instance:
    hostname: eurekaserver1
    appname: eureka-cluster
  environment: local
  client:
    serviceUrl:
      defaultZone:
              http://eurekaserver2:8762/eureka
    register-with-eureka: true
    fetch-registry: true
```

eurekaserver2

```
server:
  port: 8762

eureka:
  instance:
    hostname: eurekaserver2
    appname: eureka-cluster
  environment: local
  client:
    serviceUrl:
      defaultZone:
              http://eurekaserver1:8761/eureka
    register-with-eureka: true
    fetch-registry: true
```

Eureka replicas must have a different hostname (and port)

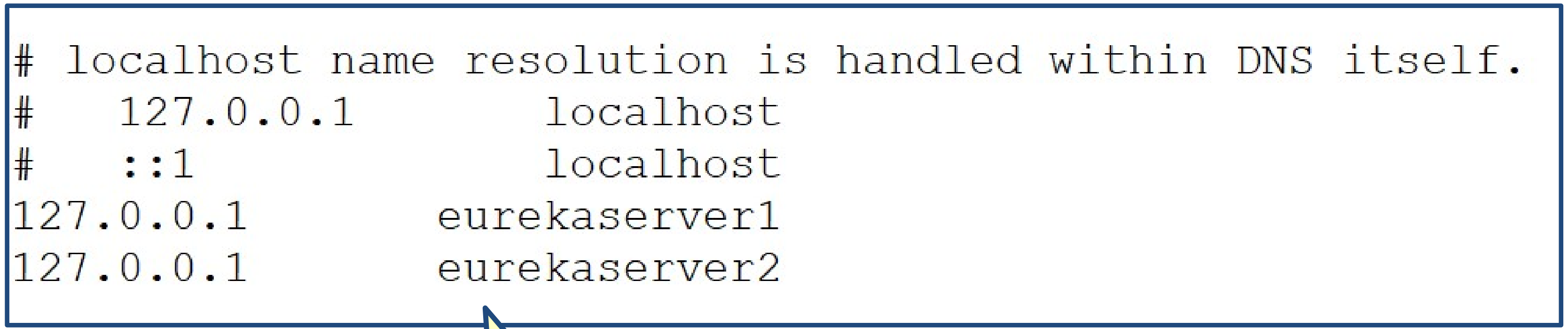
Same app name

Register eurekaserver1 in eurekaserver2

Register eurekaserver2 in eurekaserver1

# Hosts file

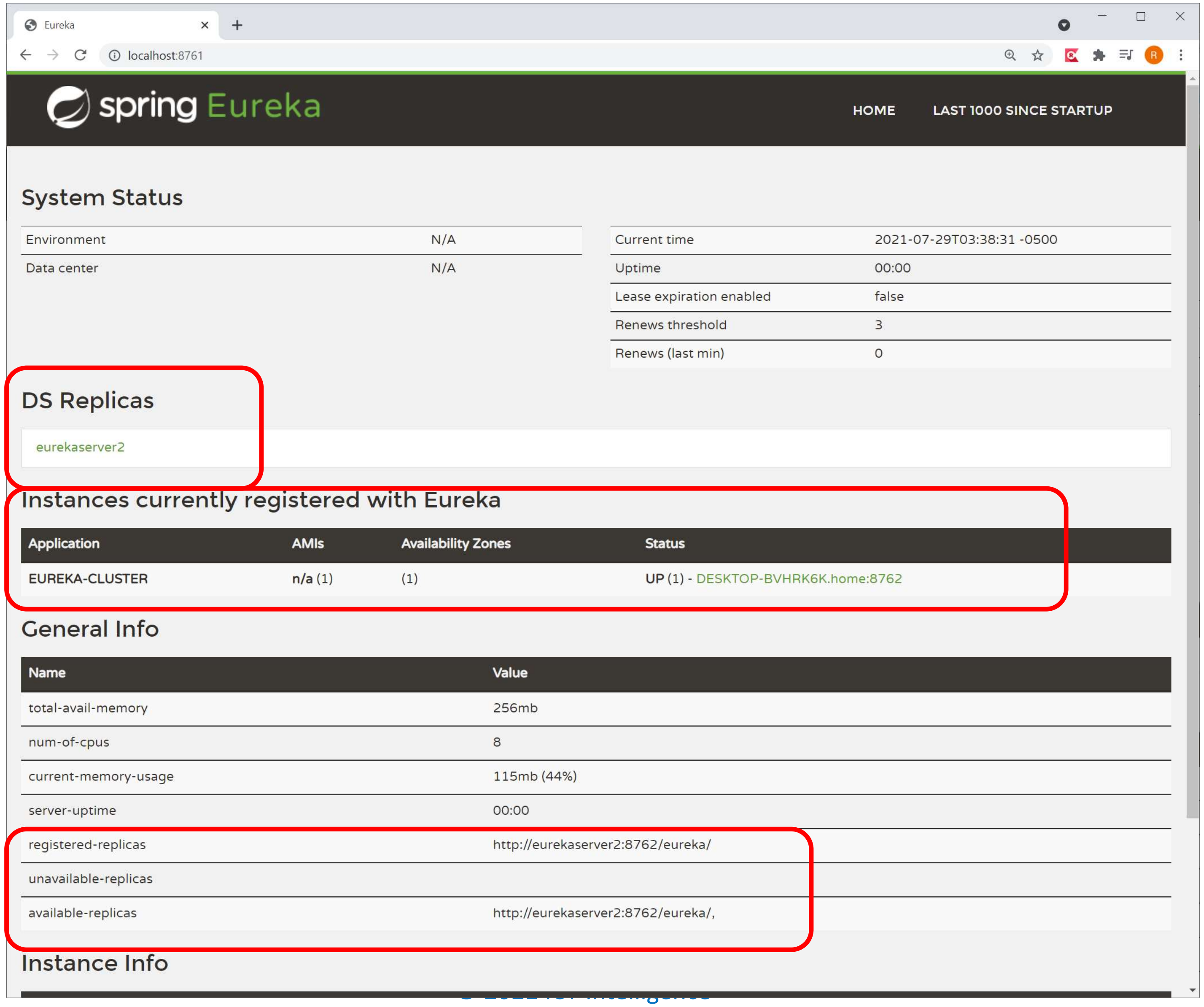
- Window: c:\Windows\System32\Drivers\etc\hosts
- Linux : /etc/hosts



```
# localhost name resolution is handled within DNS itself.
#    127.0.0.1          localhost
#    ::1                localhost
127.0.0.1        eurekaserver1
127.0.0.1        eurekaserver2
```

Map host names to machine addresses

# EurekaServer1



Eureka

localhost:8761

## System Status

| | |
|---|---|
| Environment | N/A |
| Data center | N/A |

| | |
|---|---|
| Current time | 2021-07-29T03:38:31 -0500 |
| Uptime | 00:00 |
| Lease expiration enabled | false |
| Renews threshold | 3 |
| Renews (last min) | 0 |

## DS Replicas

eurekaserver2

## Instances currently registered with Eureka

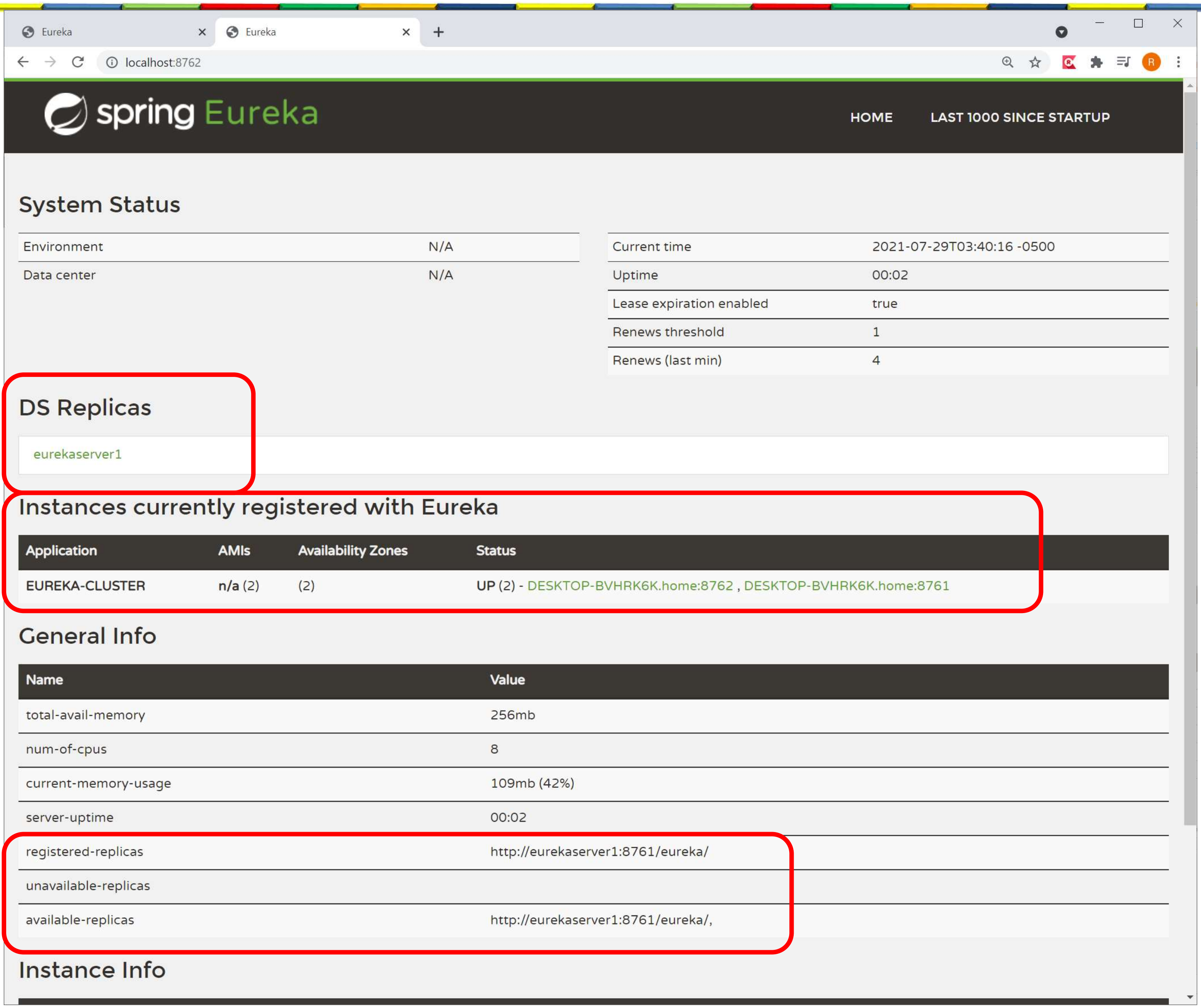
| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| EUREKA-CLUSTER | n/a (1) | (1) | UP (1) - DESKTOP-BVHRK6K.home:8762 |

## General Info

| Name | Value |
|---|---|
| total-avail-memory | 256mb |
| num-of-cpus | 8 |
| current-memory-usage | 115mb (44%) |
| server-uptime | 00:00 |
| registered-replicas | http://eurekaserver2:8762/eureka/ |
| unavailable-replicas | |
| available-replicas | http://eurekaserver2:8762/eureka/, |

## Instance Info

## EurekaServer2



System Status

| | |
|---|---|
| Environment | N/A |
| Data center | N/A |

| | |
|---|---|
| Current time | 2021-07-29T03:40:16 -0500 |
| Uptime | 00:02 |
| Lease expiration enabled | true |
| Renews threshold | 1 |
| Renews (last min) | 4 |

DS Replicas

eurekaserver1

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| EUREKA-CLUSTER | n/a (2) | (2) | UP (2) - DESKTOP-BVHRK6K.home:8762 , DESKTOP-BVHRK6K.home:8761 |

General Info

| Name | Value |
|---|---|
| total-avail-memory | 256mb |
| num-of-cpus | 8 |
| current-memory-usage | 109mb (42%) |
| server-uptime | 00:02 |
| registered-replicas | http://eurekaserver1:8761/eureka/ |
| unavailable-replicas | |
| available-replicas | http://eurekaserver1:8761/eureka/, |

Instance Info

# Accountservice

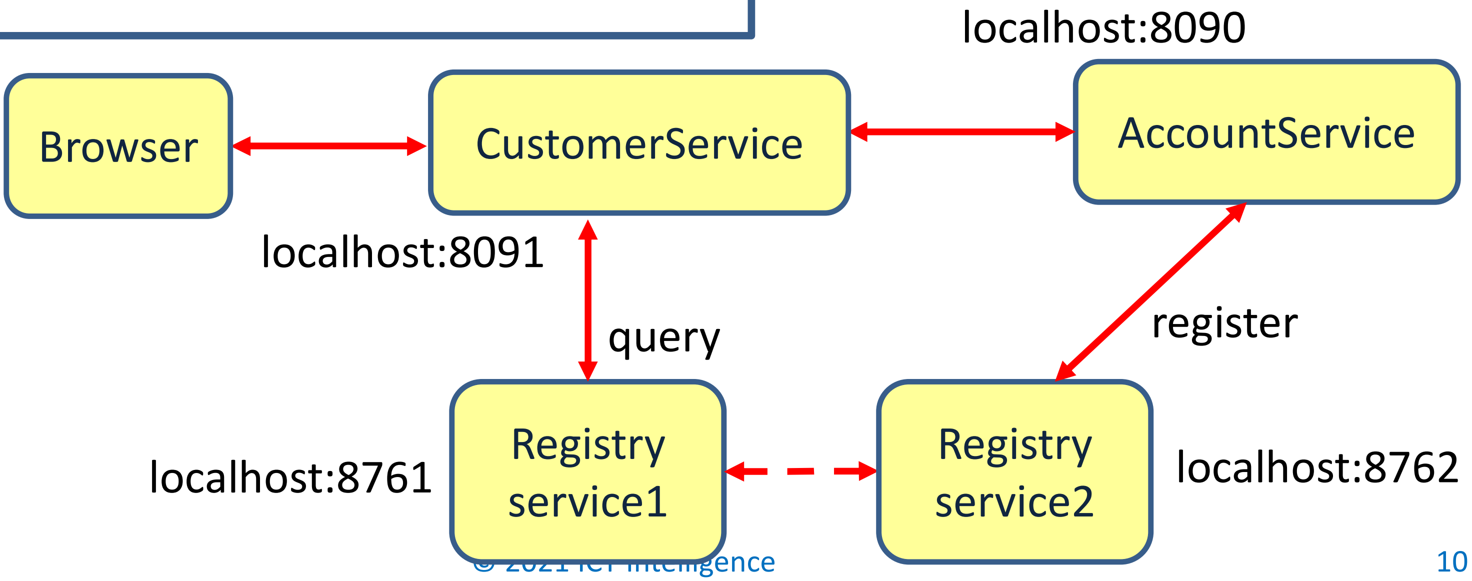
**application.yml**

```yaml
server:
  port: 8090

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8762/eureka/

spring:
  application:
    name: AccountService
```

Register in eurekaserver2



# Customerservice

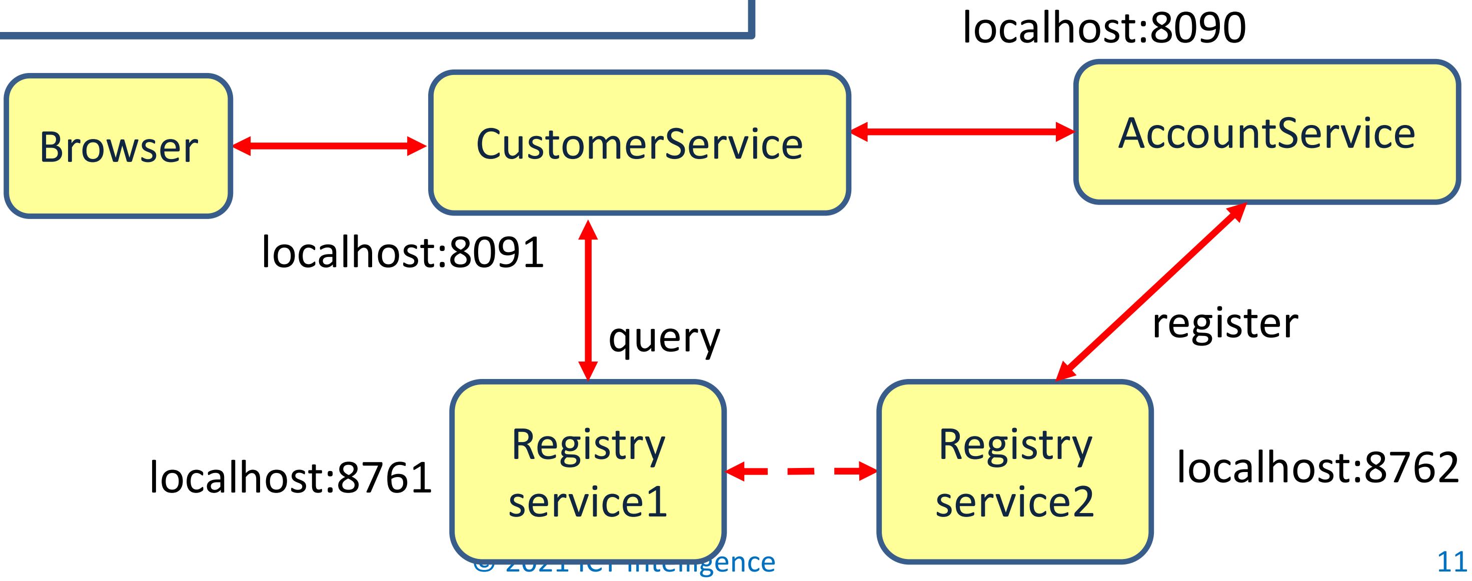
application.yml

```
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/

spring:
  application:
    name: CustomerService
```

Use eurekaserver1



# Eureka high availability

- In the client, multiple Eureka servers can be configured.

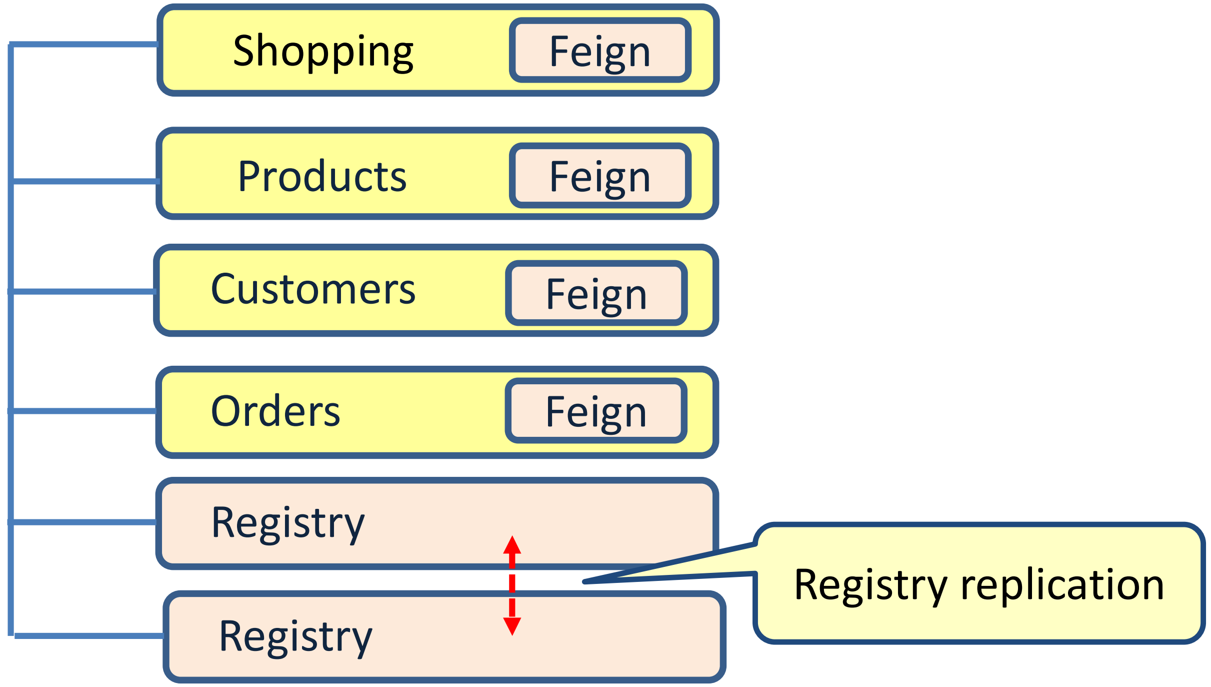
This can be a comma separated list of Eureka instances.
If the first instance does not respond, we try the next instance

**application.yml**

```yaml
server:
  port: 8091

eureka:
  client:
    serviceUrl:
      defaultZone: http://eurekaserver1:8761/eureka/,
                   http://eurekaserver2:8762/eureka/
```

## Implementing microservices

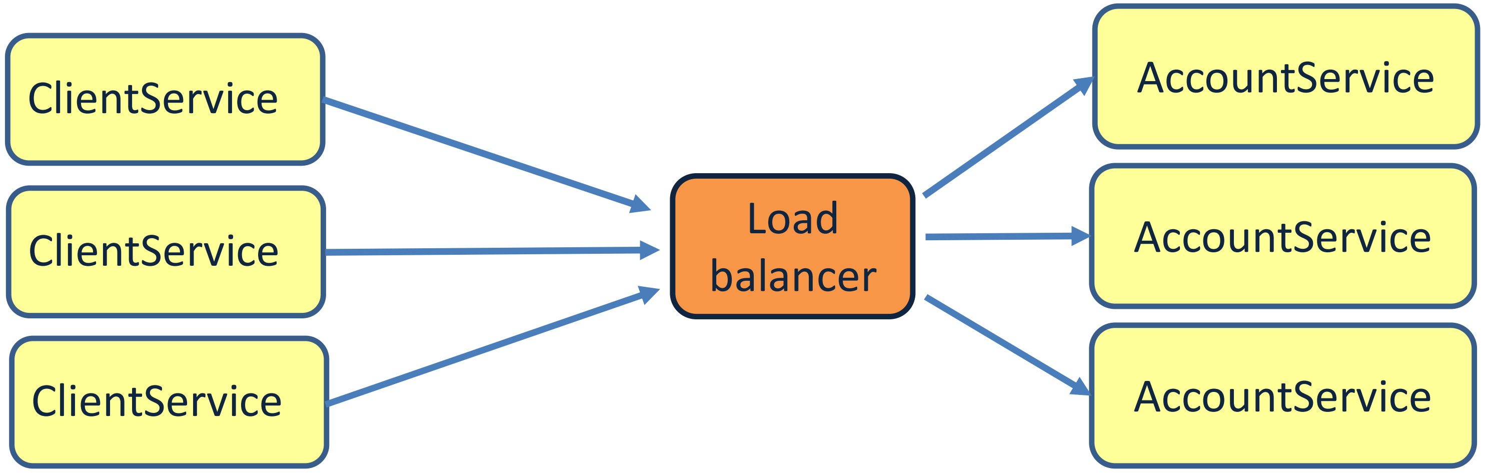


# Challenges of a microservice architecture

| Challenge | Solution |
|---|---|
| Complex communication | Feign<br>Registry |
| Performance | |
| Resilience | Registry replicas |
| Security | |
| Transactions | |
| Following the process | |
| Keep data in sync | |
| Keep interfaces in sync | |
| Keep configuration in sync | |
| Monitor health of microservices | |
| Follow/monitor business processes | |

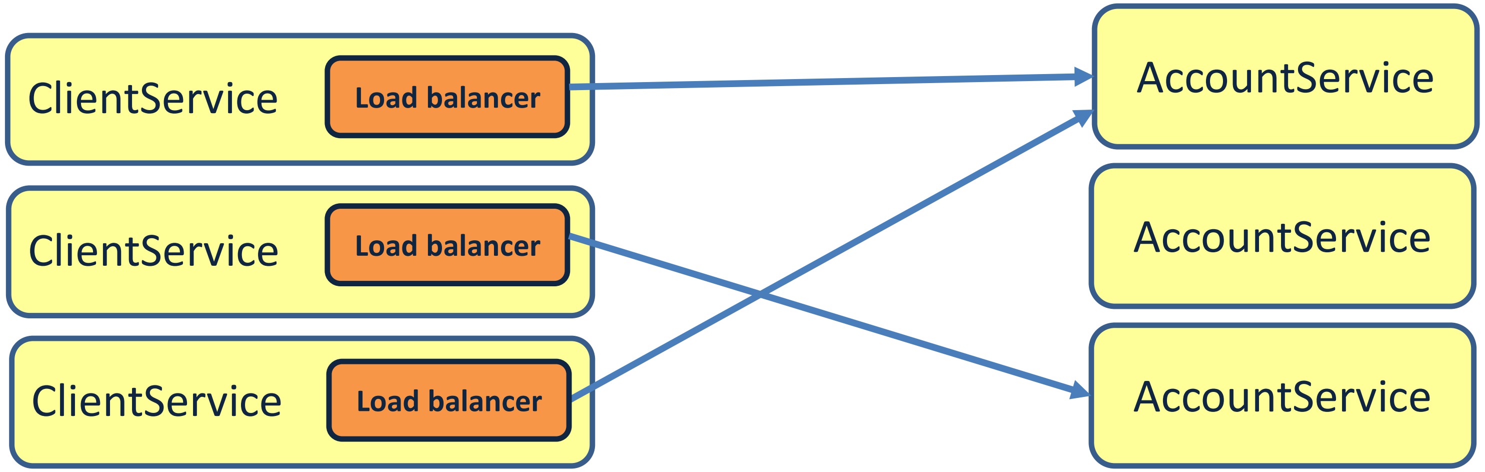
## LOAD BALANCING: RIBBON

# Server side load balancing



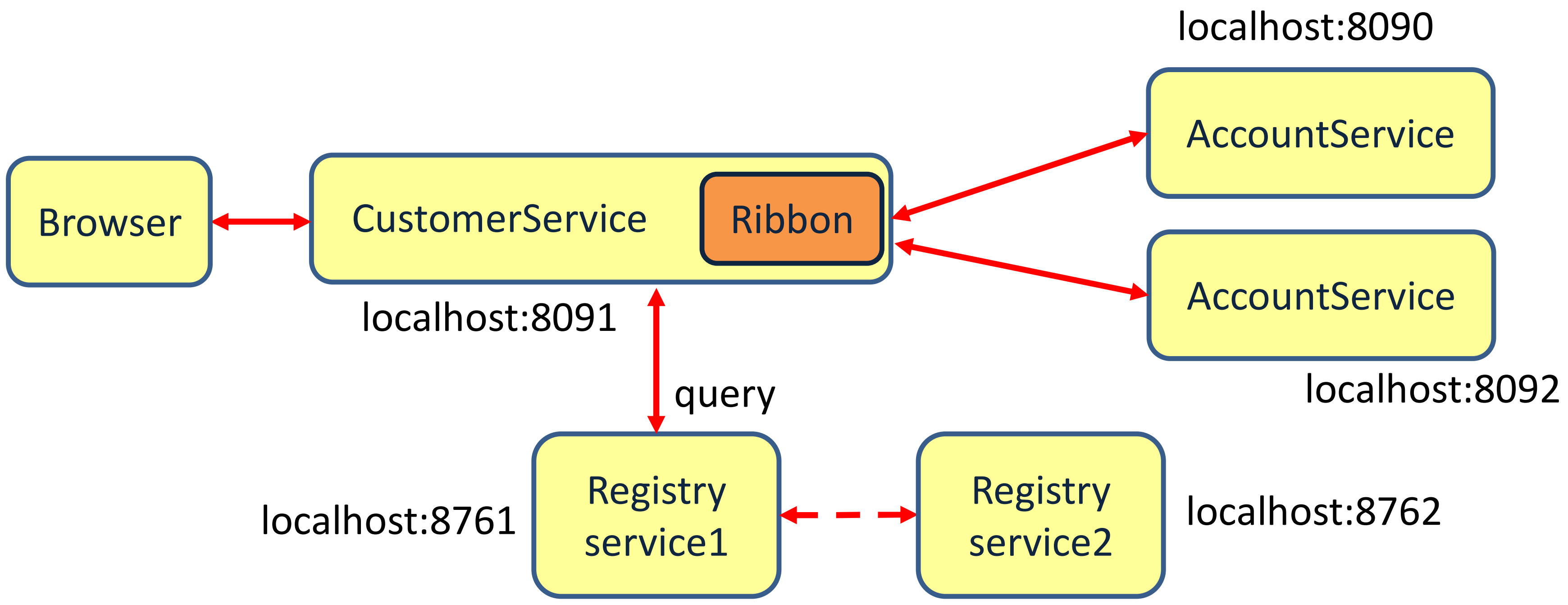
- Single point of failure
- If we add a new instance of AccountService, we need to reconfigure the load balancer
- Extra hop (performance)
- Every microservice needs its own load balancer
- Same load balance algorithm for every client
- Scaling limitation, load balance can handle only a certain number of requests

# Client side load balancing



- No single point of failure
- Simplifies service management
- Only one hop (performance)
- Auto discovery with registry based lookup (flexibility)
- Every client can use its own load balancing algorithm
- Unlimited scalable

# Load balancing using Ribbon



# AccountService

```java
@RestController
public class AccountController {
  @GetMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "1000.00");
  }
}
```

**application.yml**

```yaml
server:
  port: 8090

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/,
                   http://localhost:8762/eureka/

spring:
  application:
    name: AccountService
```

## AccountService2

```java
@RestController
public class AccountController {
  @GetMapping("/account/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
    return new Account("1234", "2000.00");
  }
}
```

**application.yml**

```yaml
server:
  port: 8092

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/,
                    http://localhost:8762/eureka/

spring:
  application:
    name: AccountService
```

# CustomerService: the controller

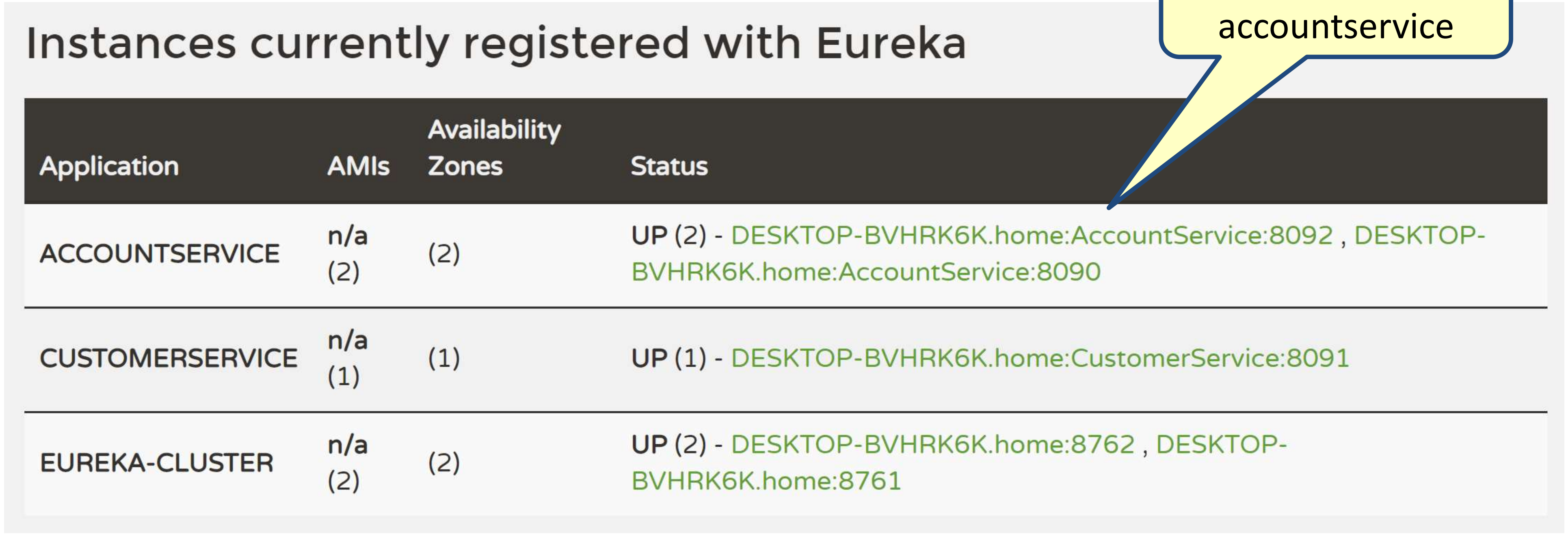
```
@RestController
public class CustomerController {
  @Autowired
  AccountFeignClient accountClient;

  @RequestMapping("/customer/{customerid}")
  public Account getName(@PathVariable("customerid") String customerId) {
        Account account = accountClient.getName(customerId);
        return account;
  }

  @FeignClient(name = "account-service")
  interface AccountFeignClient {
    @RequestMapping("/account/{customerid}")
    public Account getName(@PathVariable("customerid") String customerId);
  }
}
```

Feign automatically uses the Ribbon load balancer

# Eureka

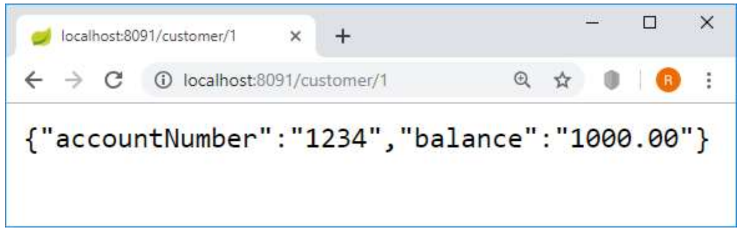


Instances currently registered with Eureka

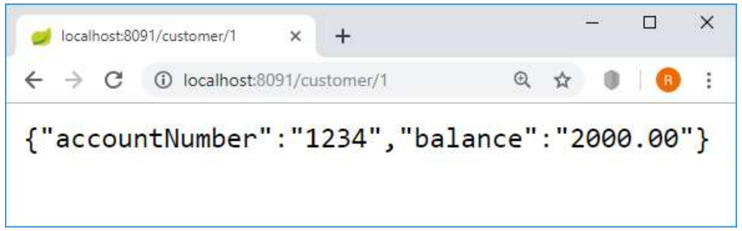
| Application | AMIs | Availability Zones | Status |
|---|---|---|---|
| ACCOUNTSERVICE | n/a (2) | (2) | UP (2) - DESKTOP-BVHRK6K.home:AccountService:8092 , DESKTOP-BVHRK6K.home:AccountService:8090 |
| CUSTOMERSERVICE | n/a (1) | (1) | UP (1) - DESKTOP-BVHRK6K.home:CustomerService:8091 |
| EUREKA-CLUSTER | n/a (2) | (2) | UP (2) - DESKTOP-BVHRK6K.home:8762 , DESKTOP-BVHRK6K.home:8761 |

2 instances of accountservice

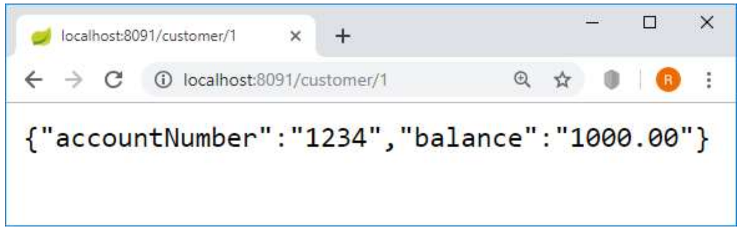
# Round robin



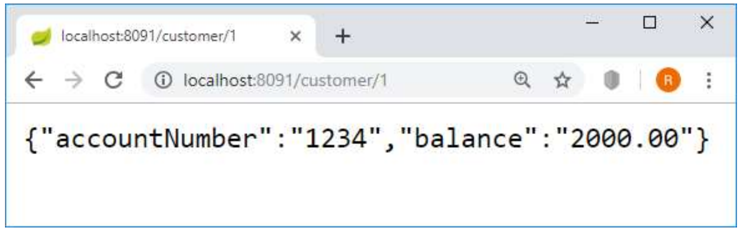
```
{"accountNumber":"1234","balance":"1000.00"}
```



```
{"accountNumber":"1234","balance":"2000.00"}
```

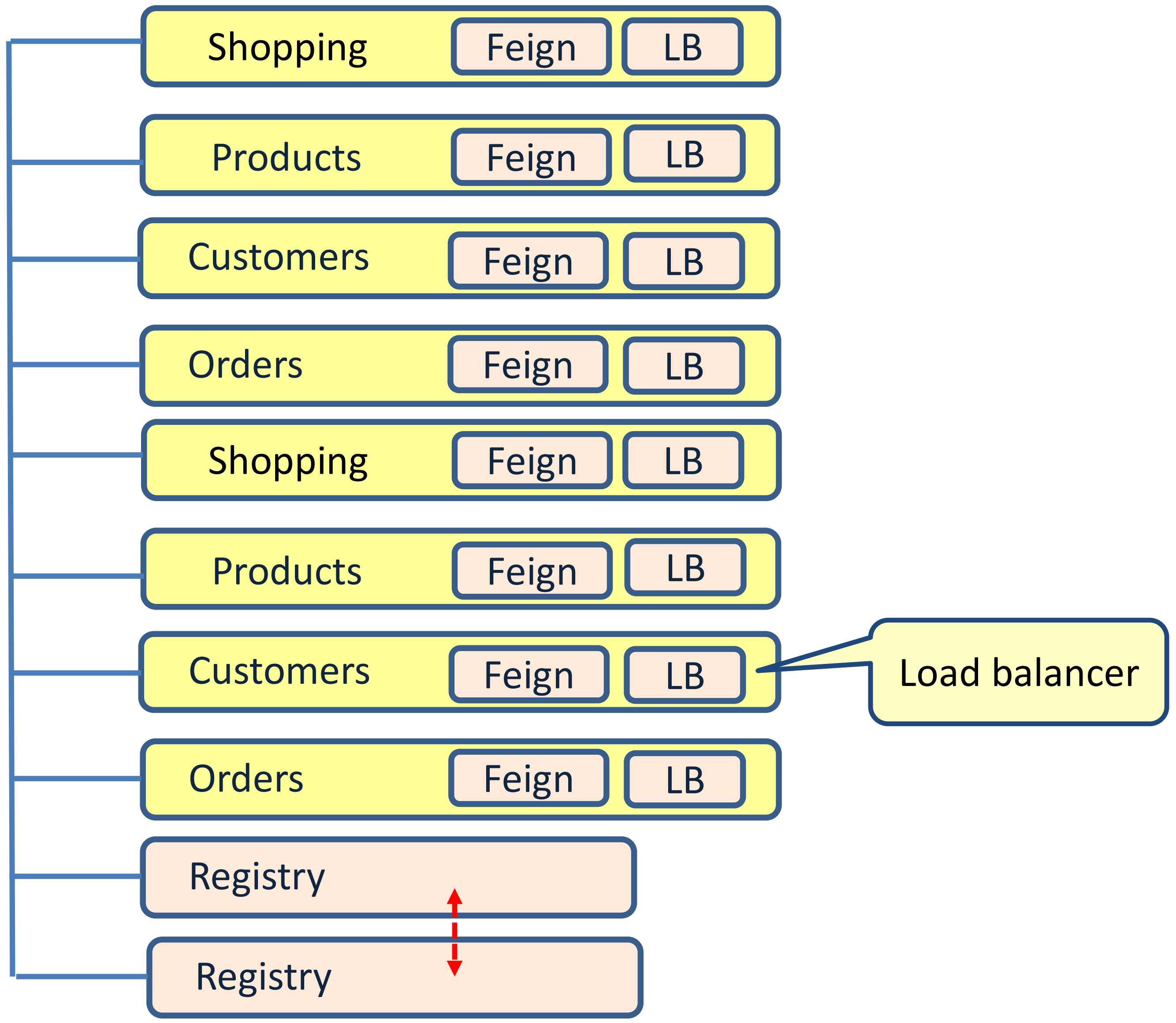


```
{"accountNumber":"1234","balance":"1000.00"}
```



```
{"accountNumber":"1234","balance":"2000.00"}
```

# Implementing microservices

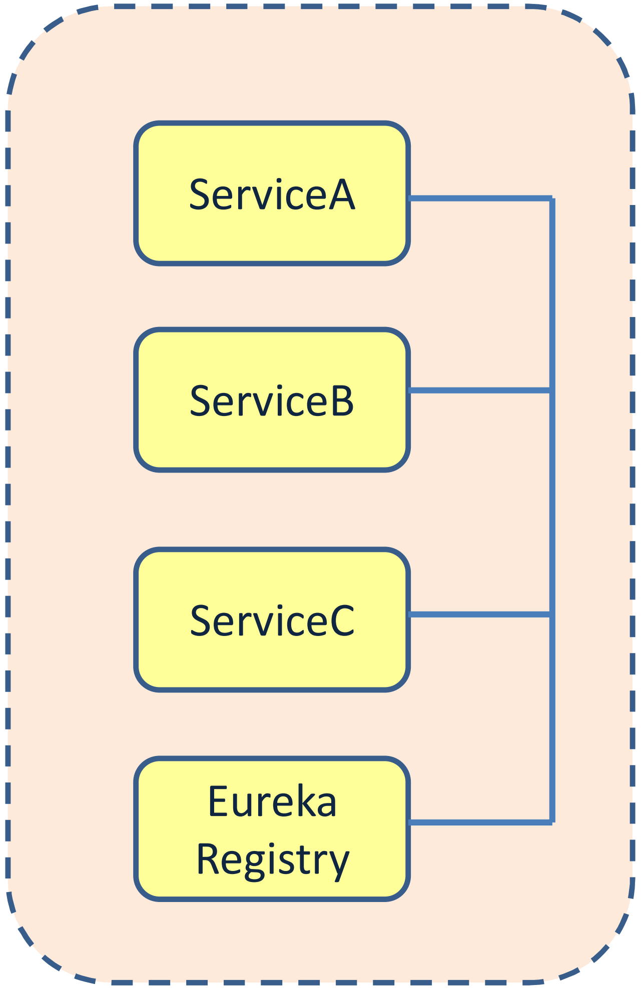


# Challenges of a microservice architecture

| Challenge | Solution |
|---|---|
| Complex communication | Feign<br>Registry |
| Performance | |
| Resilience | Registry replicas<br>Load balancing between multiple service instances |
| Security | |
| Transactions | |
| Following the process | |
| Keep data in sync | |
| Keep interfaces in sync | |
| Keep configuration in sync | |
| Monitor health of microservices | |
| Follow/monitor business processes | |

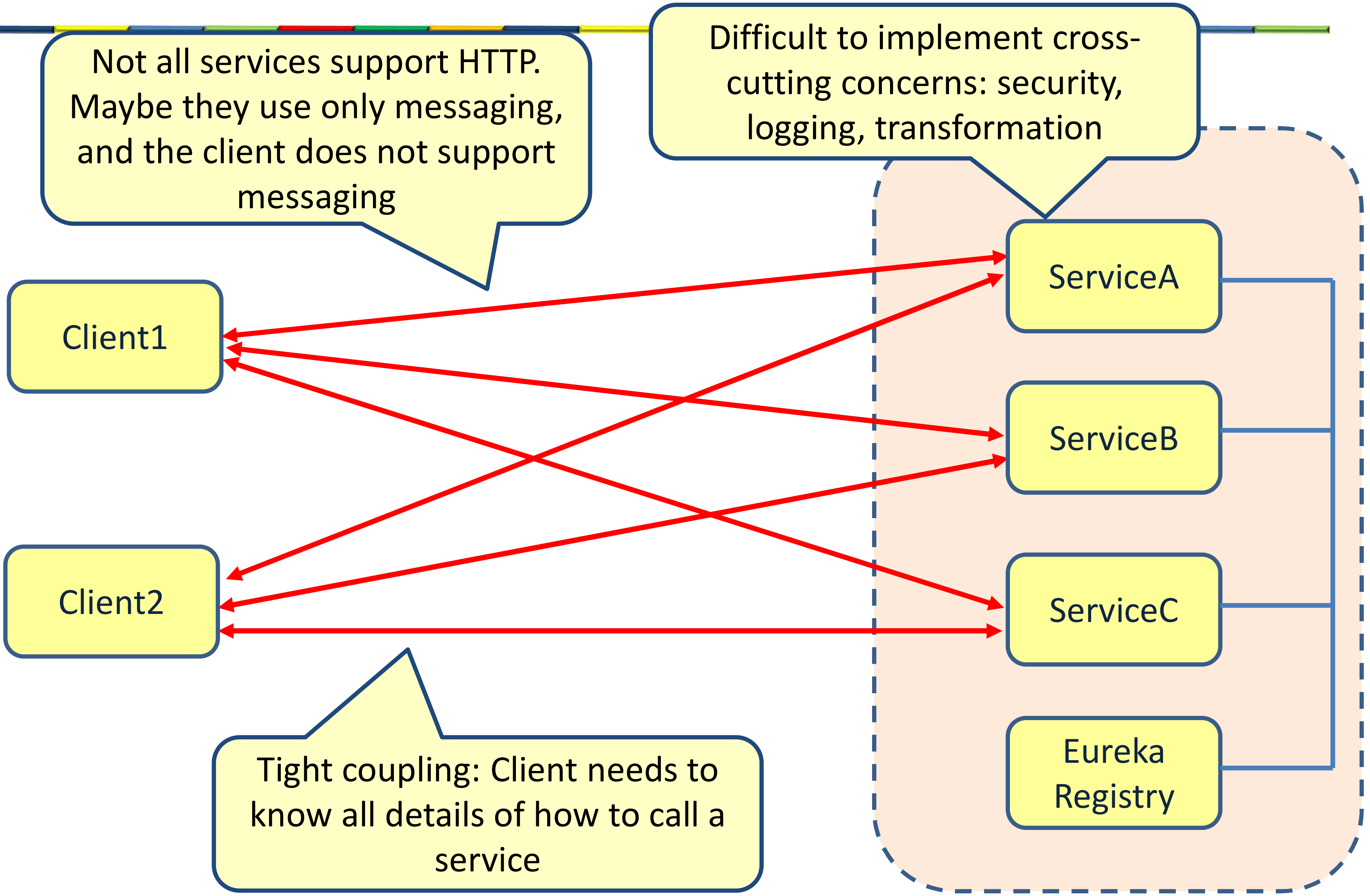
## API GATEWAY

# Microservice architecture

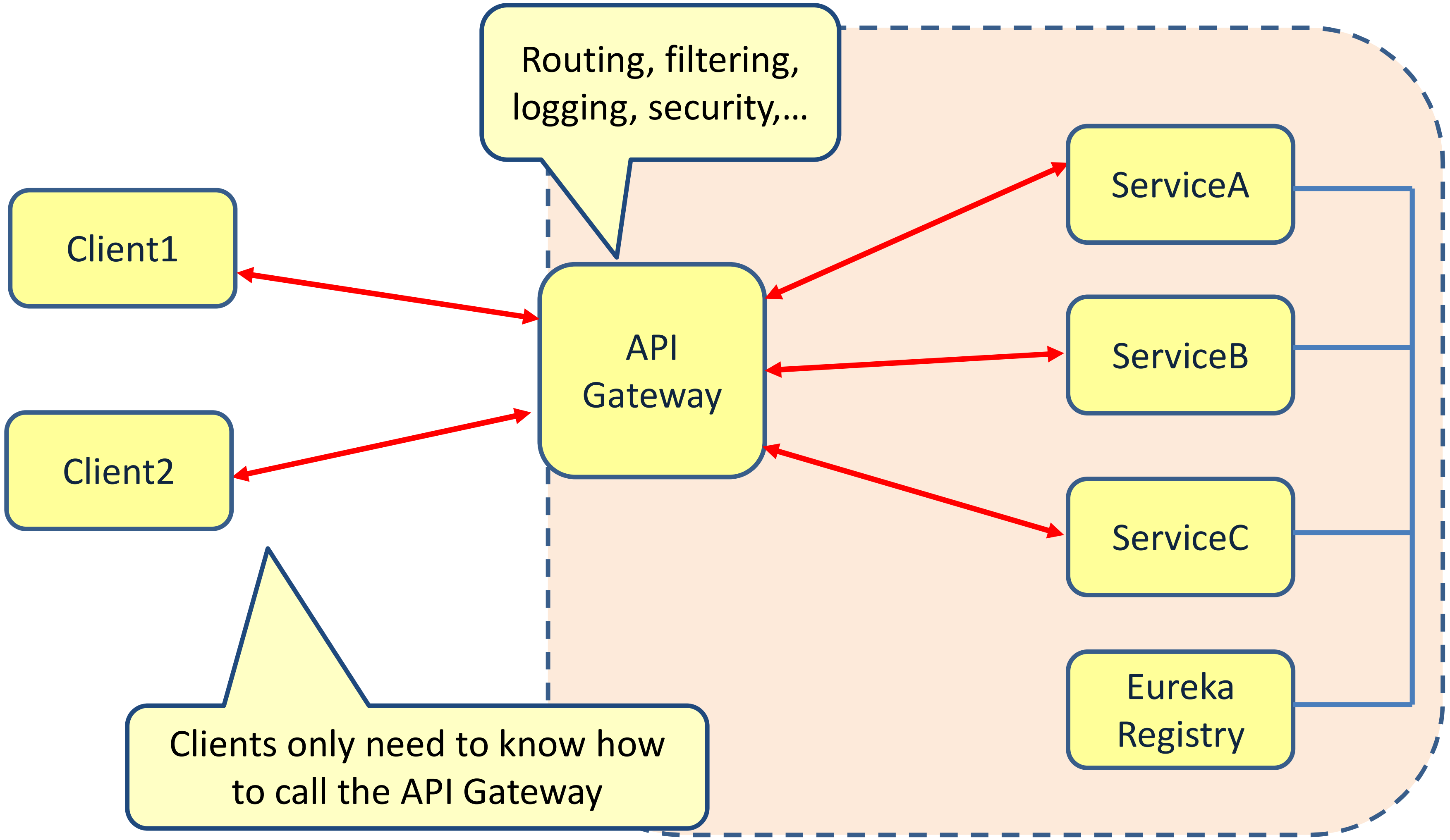


Services talk to each other using the registry

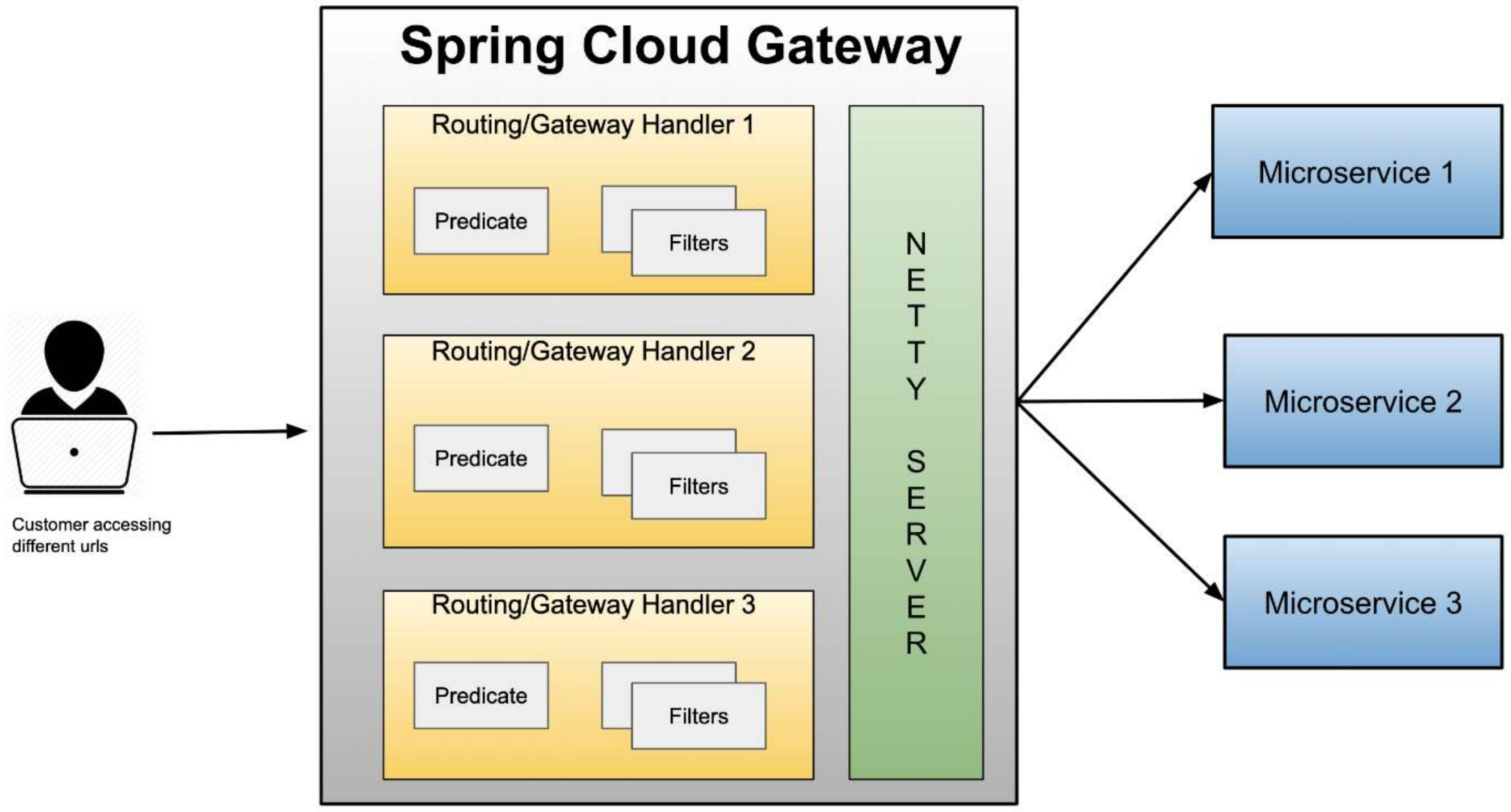
# Adding clients


Not all services support HTTP. Maybe they use only messaging, and the client does not support messaging
Difficult to implement cross-cutting concerns: security, logging, transformation
Client1
Client2
ServiceA
ServiceB
ServiceC
Eureka Registry
Tight coupling: Client needs to know all details of how to call a service

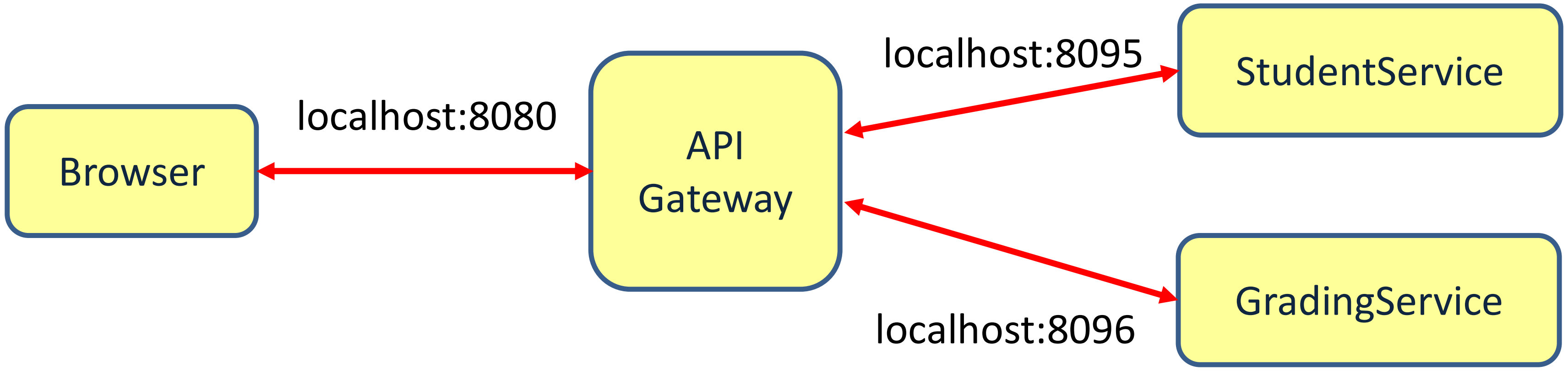
# Api Gateway


Routing, filtering, logging, security,…
Client1
Client2
API Gateway
ServiceA
ServiceB
ServiceC
Eureka Registry
Clients only need to know how to call the API Gateway

# Spring cloud gateway



# Api Gateway example



# StudentService

```java
@SpringBootApplication
@EnableDiscoveryClient
public class StudentServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(StudentServiceApplication.class, args);
  }
}
```

**application.yml**

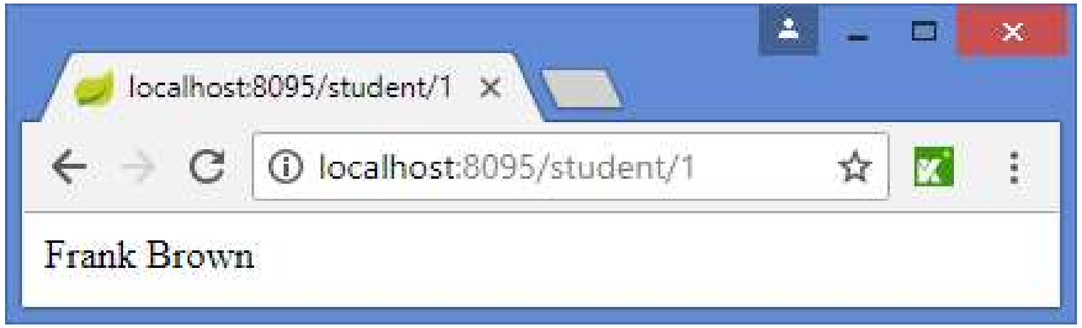
```yaml
server:
  port: 8095

spring:
  application:
    name: StudentService

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

## StudentService: the controller

```java
@RestController
public class StudentController {
  @GetMapping("/student/{studentid}")
  public String getName(@PathVariable("studentid") String studentid) {
    return "Frank Brown";
  }
}
```



```
Frank Brown
```

## GradingService

```java
@SpringBootApplication
@EnableDiscoveryClient
public class GradingServiceApplication {

  public static void main(String[] args) {
    SpringApplication.run(StudentServiceApplication.class, args);
  }
}
```

**application.yml**

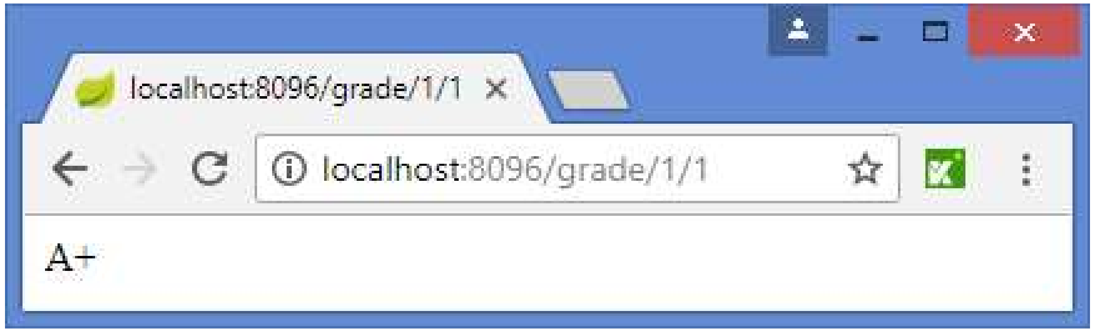
```yaml
server:
  port: 8096

spring:
  application:
    name: GradingService

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

## GradingService: the controller

```
@RestController
public class GradingController {
  @GetMapping("/grade/{studentid}/{courseid}")
  public String getGrade(@PathVariable("studentid") String studentid,
                         @PathVariable("courseid") String courseid) {
    return "A+";
  }
}
```


localhost:8096/grade/1/1

```
A+
```

## Spring Cloud Gateway

```java
@SpringBootApplication
public class CloudgatewayApplication {

  public static void main(String[] args) {
    SpringApplication.run(CloudgatewayApplication.class, args);
  }
}
```

**POM.xml**

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

# Spring Cloud Gateway

**application.yml**

```
spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      routes:
      - id: studentModule
        uri: http://localhost:8095/
        predicates:
        - Path=/student/**
      - id: gradingModule
        uri: http://localhost:8096/
        predicates:
        - Path=/grade/**

server:
  port: 8080
```

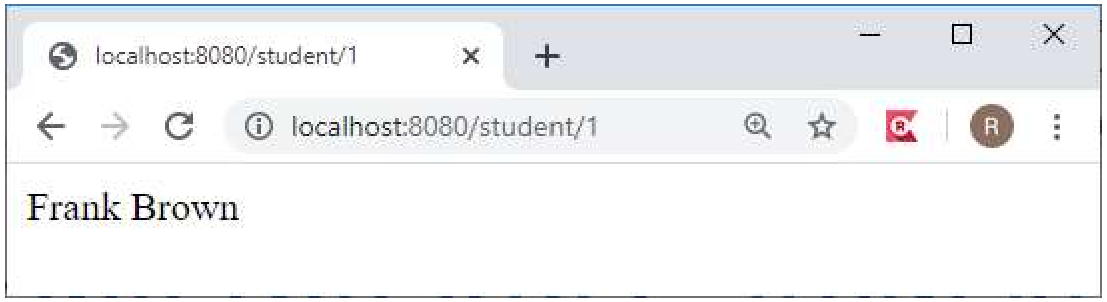
Id should be unique for every route

Route /student to localhost:8095

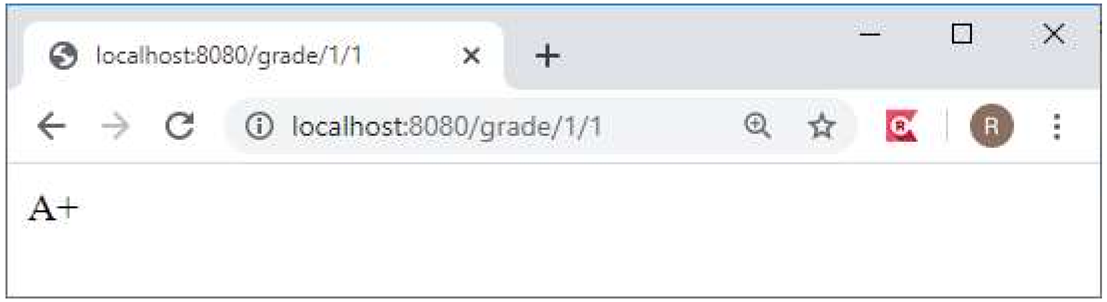
Route /grades to localhost:8096

A route is matched if predicate is true

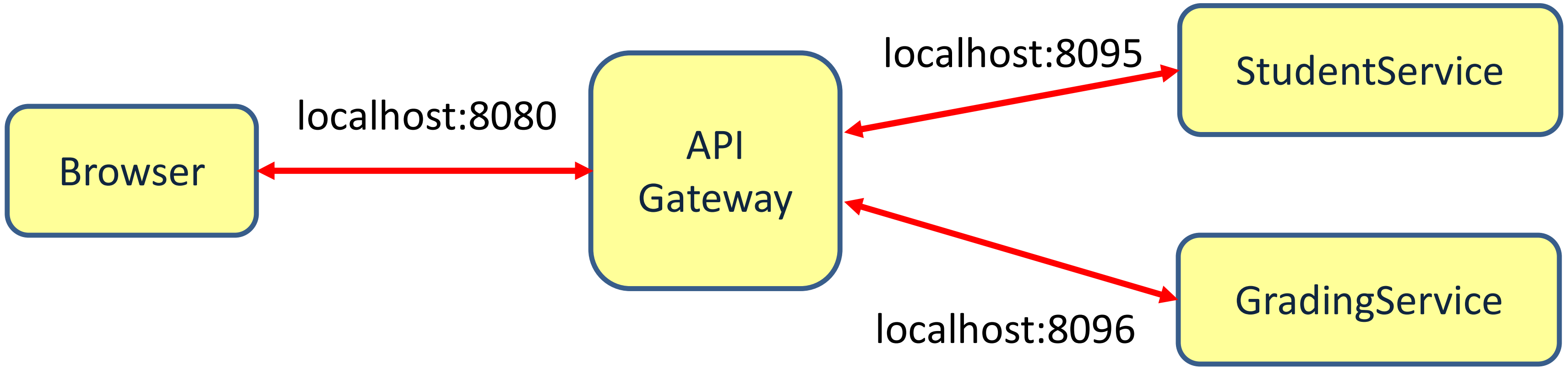
# Using the API Gateway


localhost:8080/student/1

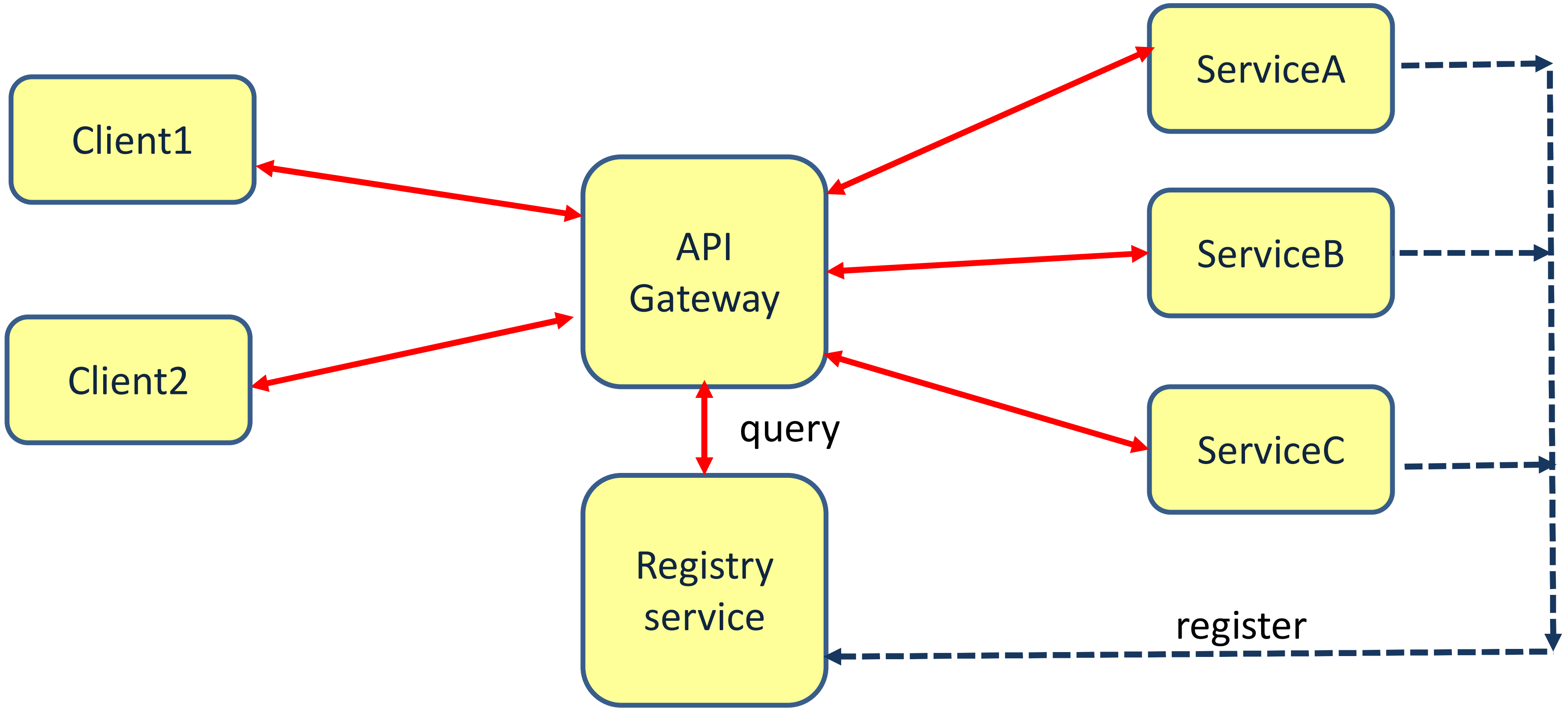
Frank Brown


localhost:8080/grade/1/1

A+



# Api Gateway and registry service



# Spring cloud gateway with the registry

**application.yml**

```yaml
spring:
  application:
    name: api-gateway
  cloud:
    gateway:
      routes:
      - id: studentModule
        uri: lb://StudentService
        predicates:
        - Path=/student/**
      - id: gradingModule
        uri: lb://GradingService
        predicates:
        - Path=/grade/**

server:
  port: 8080

eureka:
  client:
    serviceUrl:
      defaultZone: http://localhost:8761/eureka/
```

Route /student to the service with the name StudentService using the load balancer

Route /grade to the service with the name GradingService using the load balancer

URL to Eureka

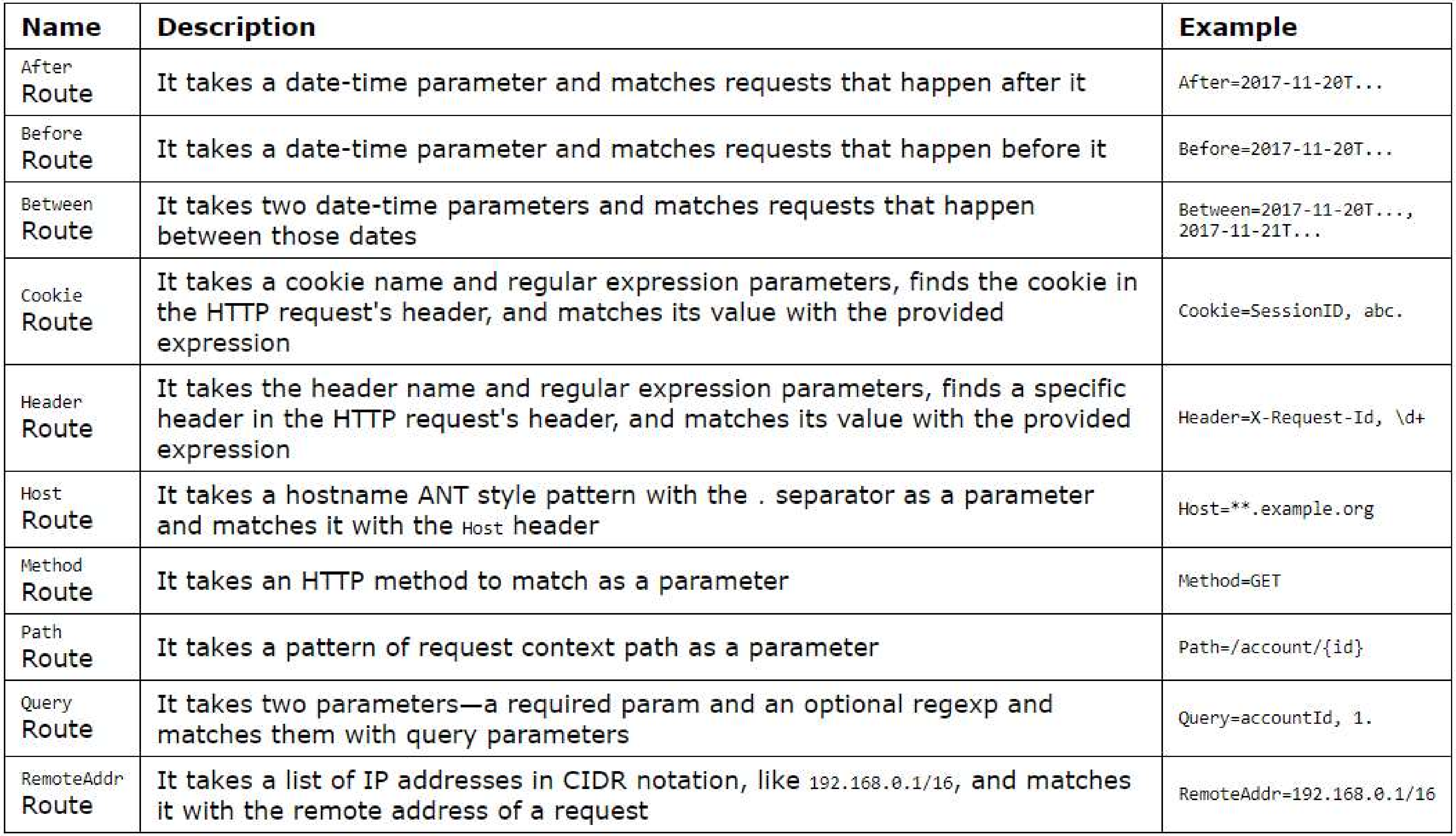
# Java based config

```
@Configuration
public class BeanConfig {

  @Bean
  public RouteLocator gatewayRoutes(RouteLocatorBuilder builder) {
    return builder.routes()
                .route(r -> r.path("/student/**")
                        .uri("lb://StudentService")
                        .id("studentModule"))

                .route(r -> r.path("/grade/**")
                        .uri("lb://GradingService")
                        .id("gradesModule"))
                .build();
  }
}
```

# Build-in predicates



| Name | Description | Example |
|---|---|---|
| After Route | It takes a date-time parameter and matches requests that happen after it | After=2017-11-20T... |
| Before Route | It takes a date-time parameter and matches requests that happen before it | Before=2017-11-20T... |
| Between Route | It takes two date-time parameters and matches requests that happen between those dates | Between=2017-11-20T..., 2017-11-21T... |
| Cookie Route | It takes a cookie name and regular expression parameters, finds the cookie in the HTTP request's header, and matches its value with the provided expression | Cookie=SessionID, abc. |
| Header Route | It takes the header name and regular expression parameters, finds a specific header in the HTTP request's header, and matches its value with the provided expression | Header=X-Request-Id, \d+ |
| Host Route | It takes a hostname ANT style pattern with the . separator as a parameter and matches it with the Host header | Host=**.example.org |
| Method Route | It takes an HTTP method to match as a parameter | Method=GET |
| Path Route | It takes a pattern of request context path as a parameter | Path=/account/{id} |
| Query Route | It takes two parameters—a required param and an optional regexp and matches them with query parameters | Query=accountId, 1. |
| RemoteAddr Route | It takes a list of IP addresses in CIDR notation, like 192.168.0.1/16, and matches it with the remote address of a request | RemoteAddr=192.168.0.1/16 |

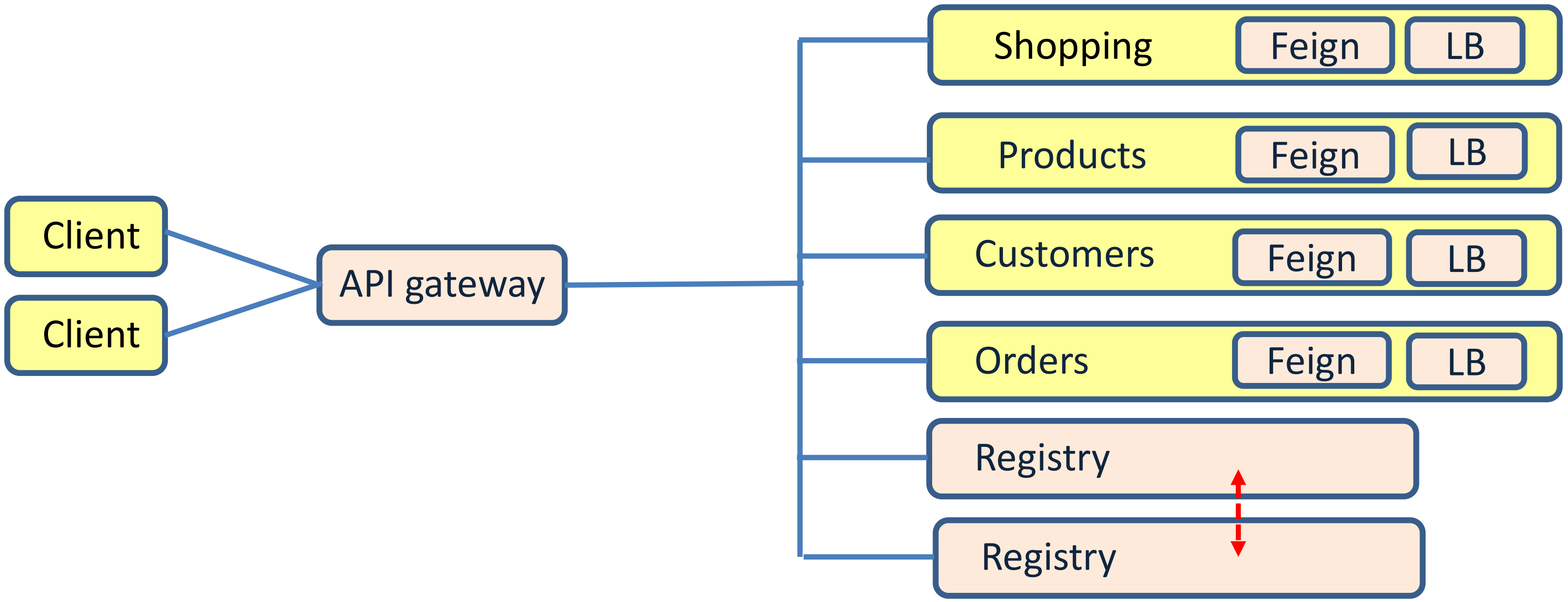
# Filters

- Pre and post filters
  - Build-in filters
  - Custom filters
  - Global filters

```
cloud:
    gateway:
      routes:
      - id: studentModule
        uri: lb://StudentService
        filters:
        - AddRequestHeader=X-myHeader, Hello
        - AddRequestParameter=name, John
        - AddResponseHeader=X-someHeader, Hello World
        predicates:
        - Path=/student/**
```

Build-in filters

# Implementing microservices



# Challenges of a microservice architecture

| Challenge | Solution |
|---|---|
| Complex communication | Feign<br>Registry<br>API gateway |
| Performance | |
| Resilience | Registry replicas<br>Load balancing between multiple service instances |
| Security | |
| Transactions | |
| Following the process | |
| Keep data in sync | |
| Keep interfaces in sync | |
| Keep configuration in sync | |
| Monitor health of microservices | |
| Follow/monitor business processes | |