Architecture of Enterprise Applications 9
Microservices & Serverless

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*
Shanghai Jiao Tong University
Shanghai, China
http://reins.se.sjtu.edu.cn/~chenhp
e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- Contents
  - Microservice
  - Routing and Filtering
  - Severless
  - Function Service

- Objectives
  - 能够根据业务需求，抽象出系统中的微服务，并能够基于Spring框架开发对应的微服务

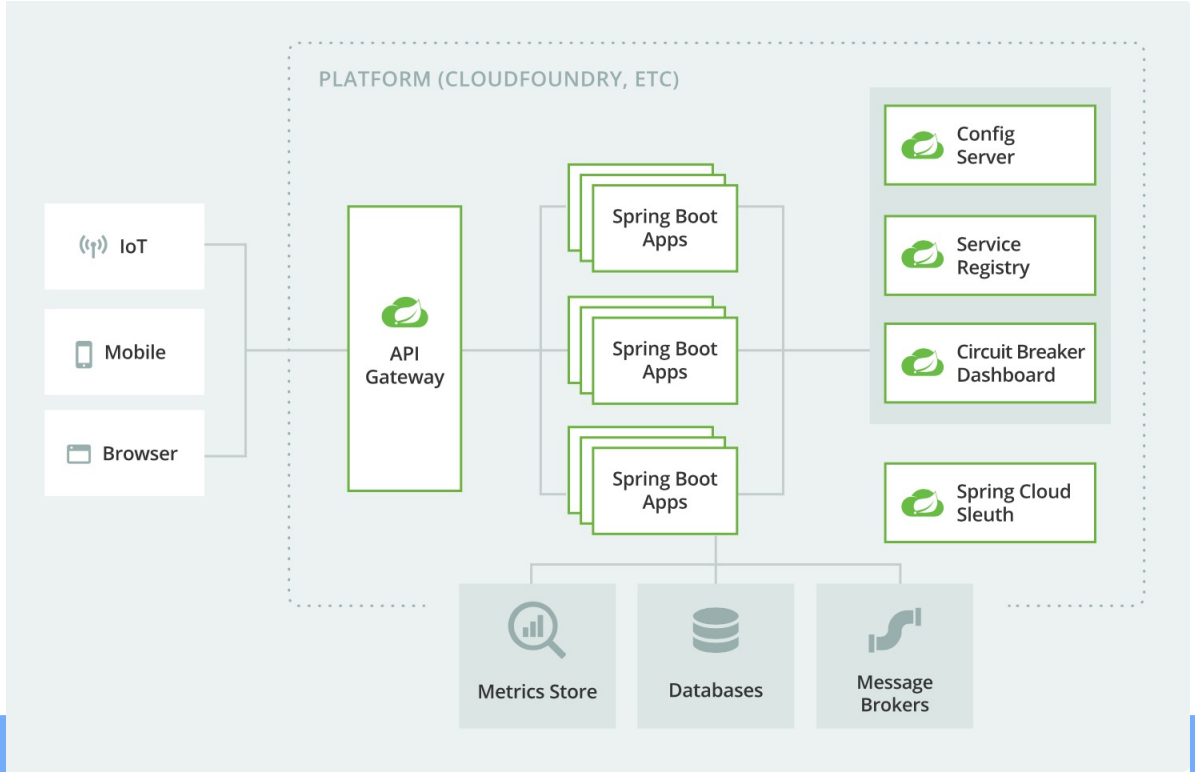
Microservices

- Microservice architectures are the ‘new normal’.
  - Building small, self-contained, ready to run applications can bring great flexibility and added resilience to your code.
  - Spring Boot’s many purpose-built features make it easy to build and run your microservices in production at scale.
  - And don’t forget, no microservice architecture is complete without Spring Cloud – easing administration and boosting your fault-tolerance.

- What are microservices?
  - Microservices are a modern approach to software whereby application code is delivered in small, manageable pieces, independent of others.
- Why build microservices?
  - Their small scale and relative isolation can lead to many additional benefits, such as easier maintenance, improved productivity, greater fault tolerance, better business alignment, and more.

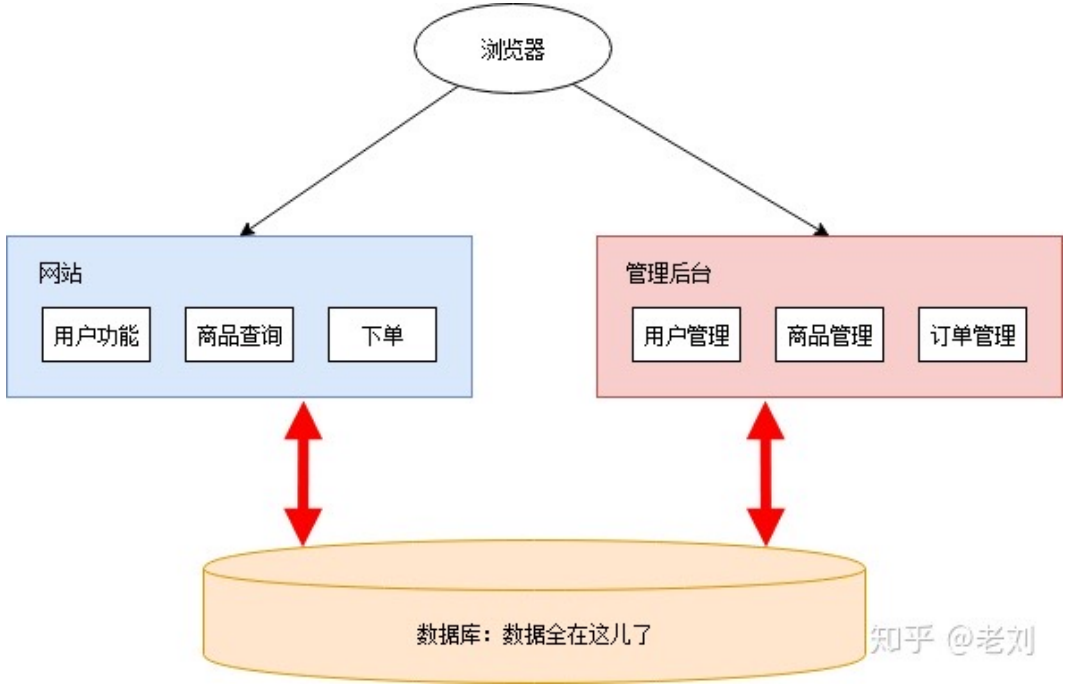
Microservice resilience with Spring Cloud

- The distributed nature of microservices brings challenges.
  - service discovery, load-balancing, circuit-breaking, distributed tracing, and monitoring



Microservice – a sample

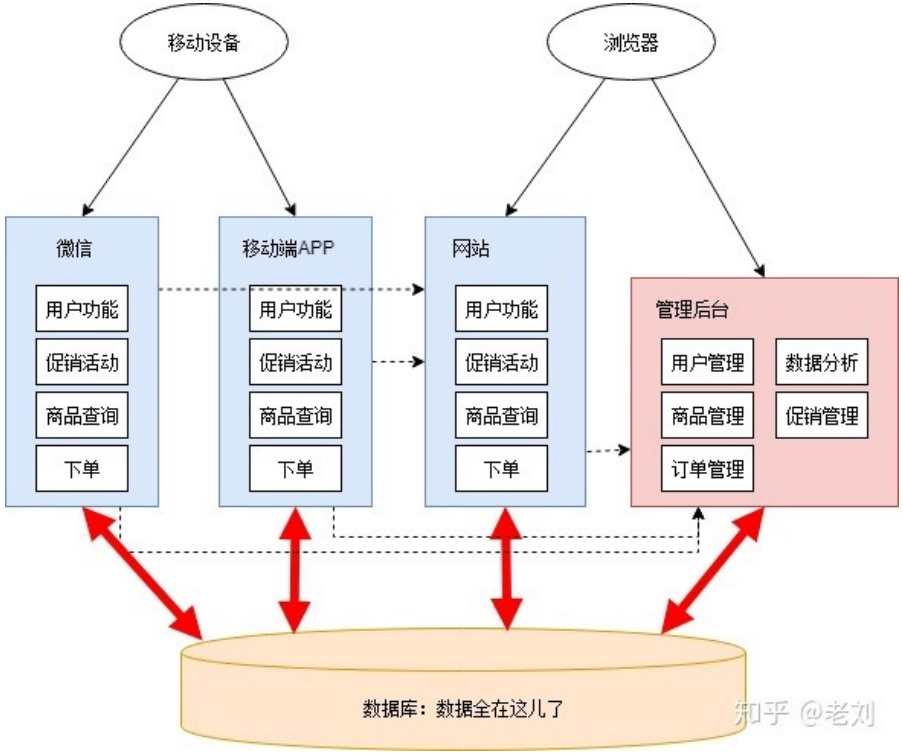
- 在线商店



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

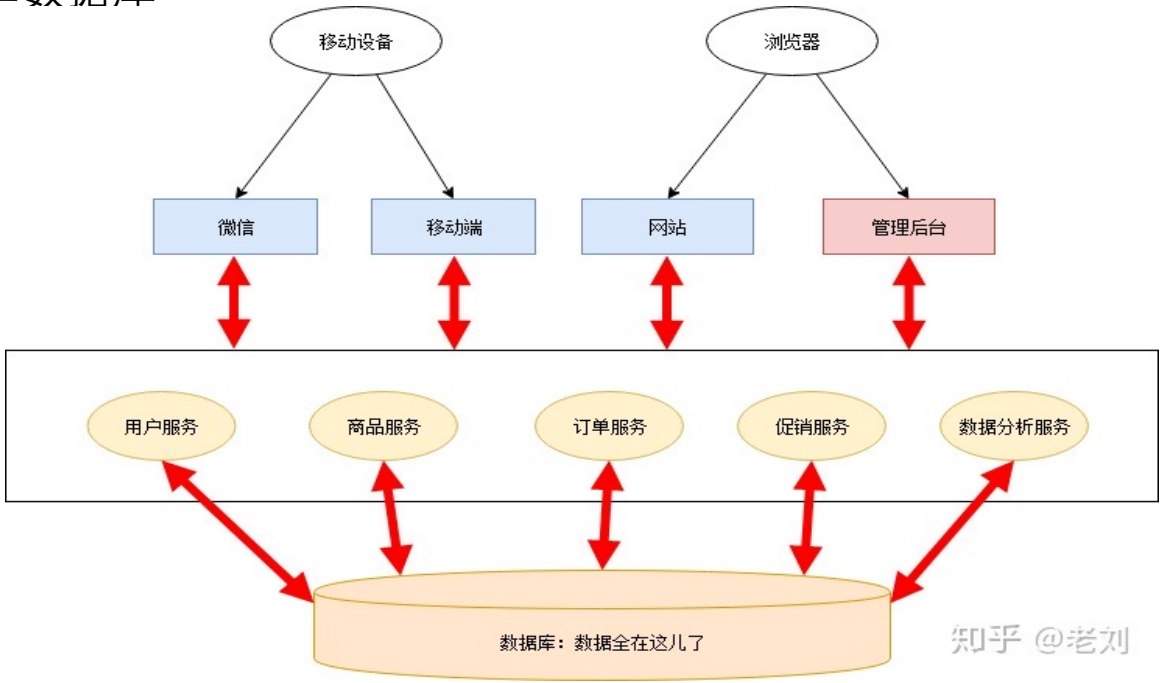
- 移动端



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

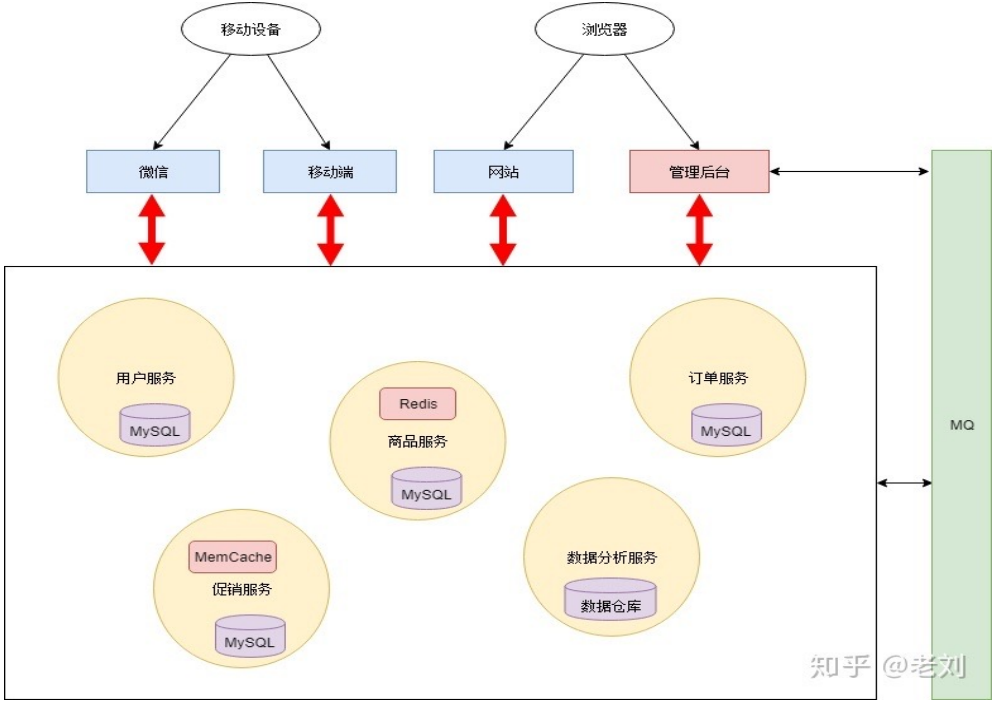
- 服务独立+共享数据库



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

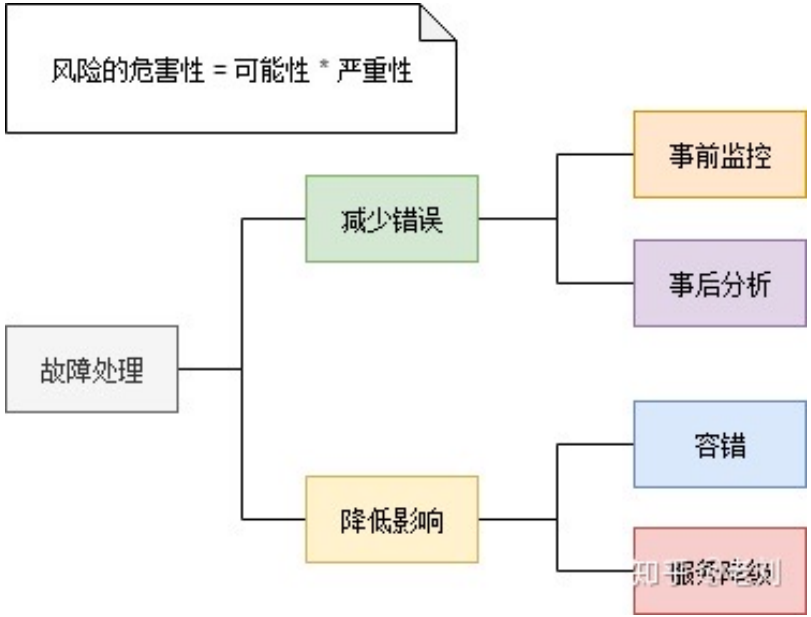
- 独立数据库+消息队列



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

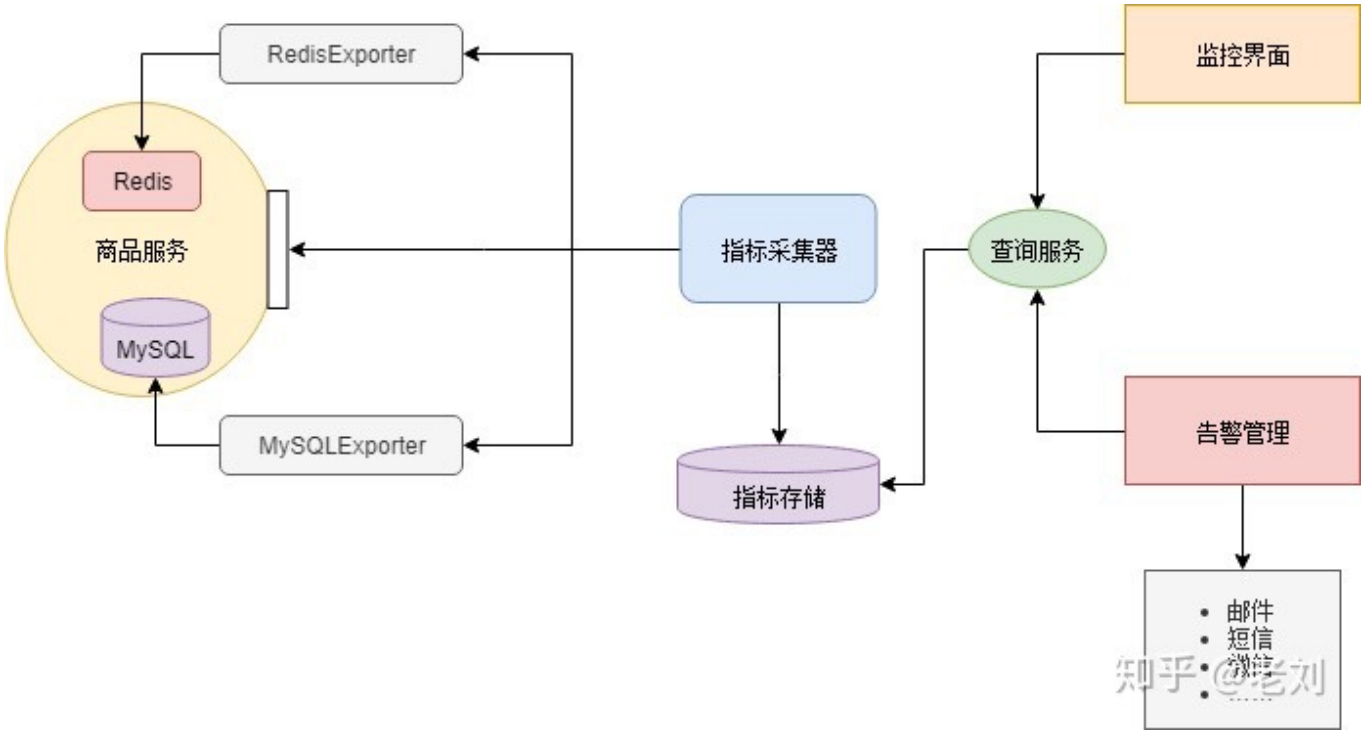
- 故障处理



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

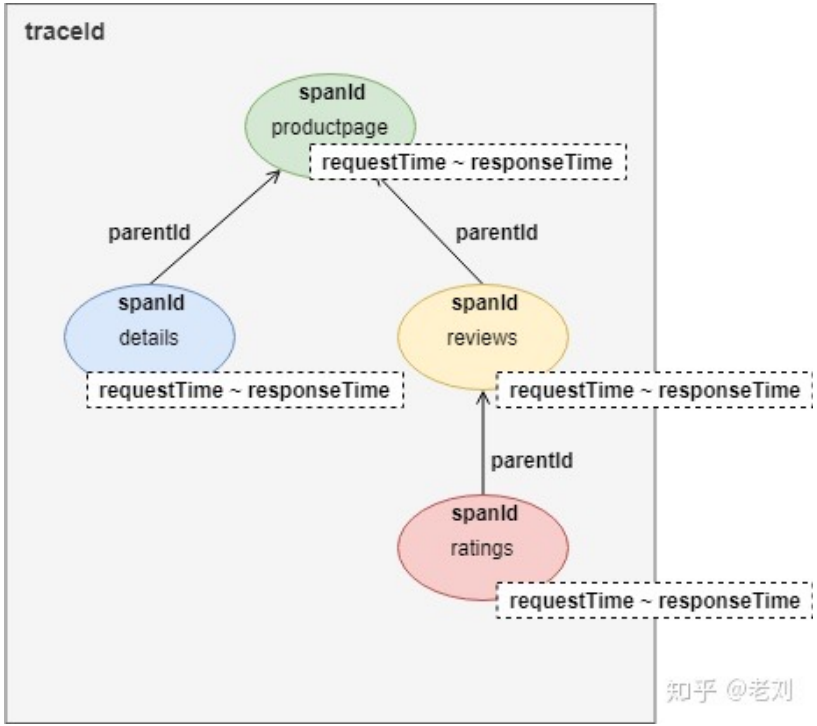
- 监控-发现故障



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

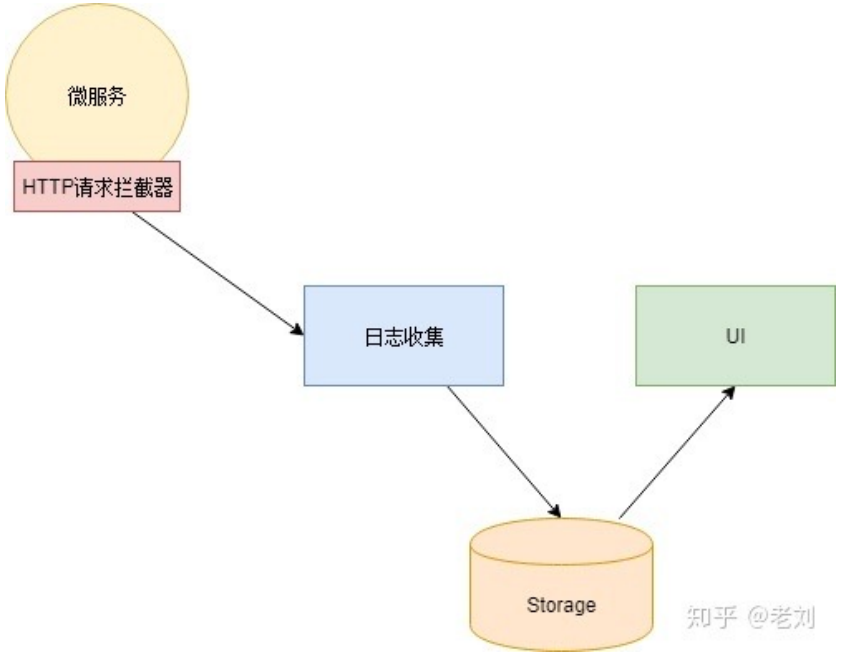
- 定位问题-链路跟踪



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

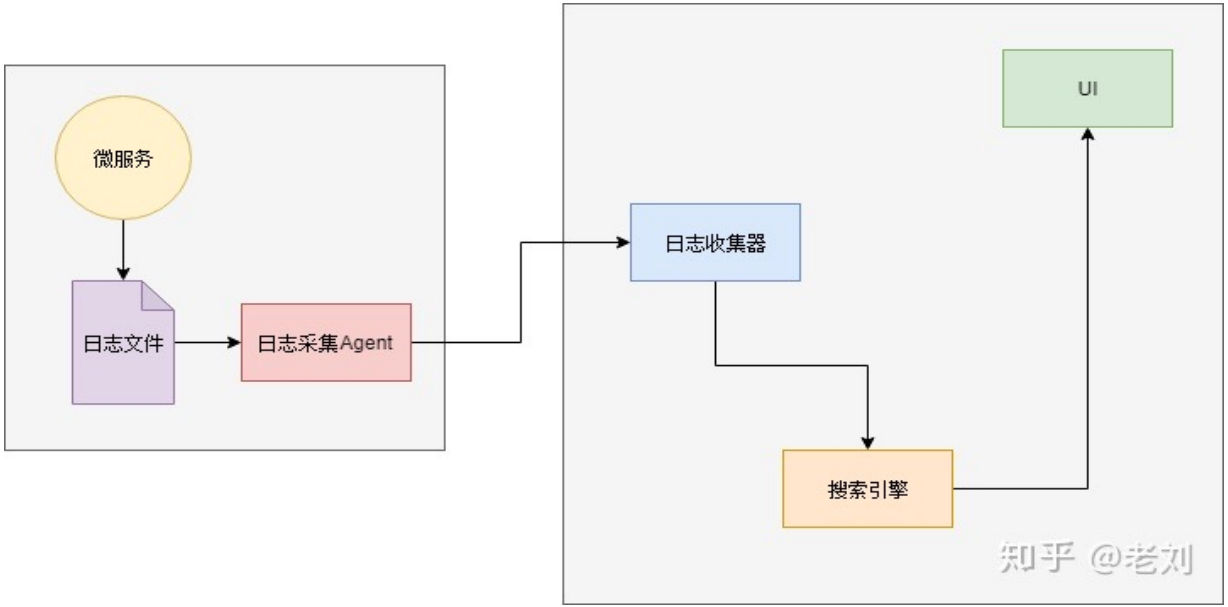
- 日志收集



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

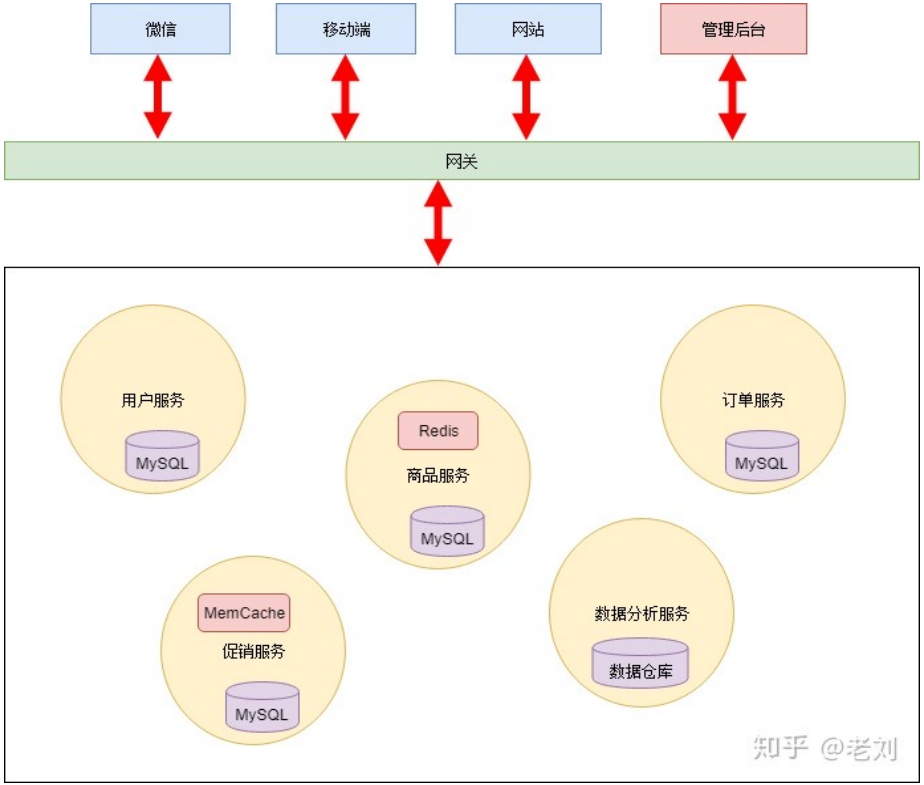
- 分析问题-日志分析



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

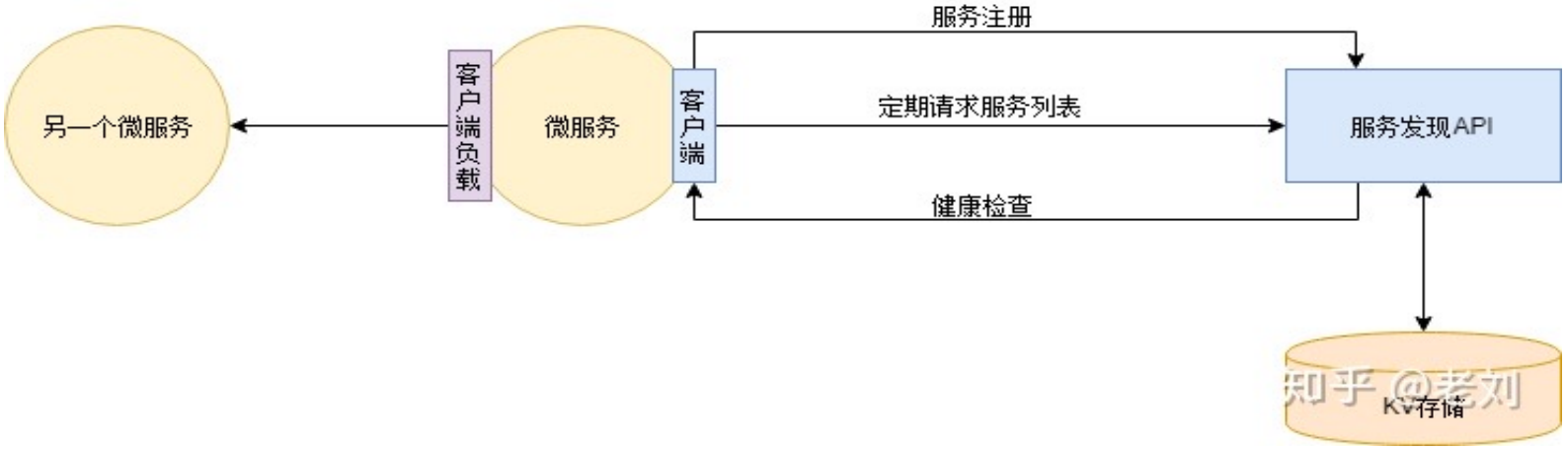
- 网关



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

Microservice – a sample

- 服务注册于发现 - 动态扩容



- From：一文详解微服务架构 https://www.zhihu.com/question/65502802

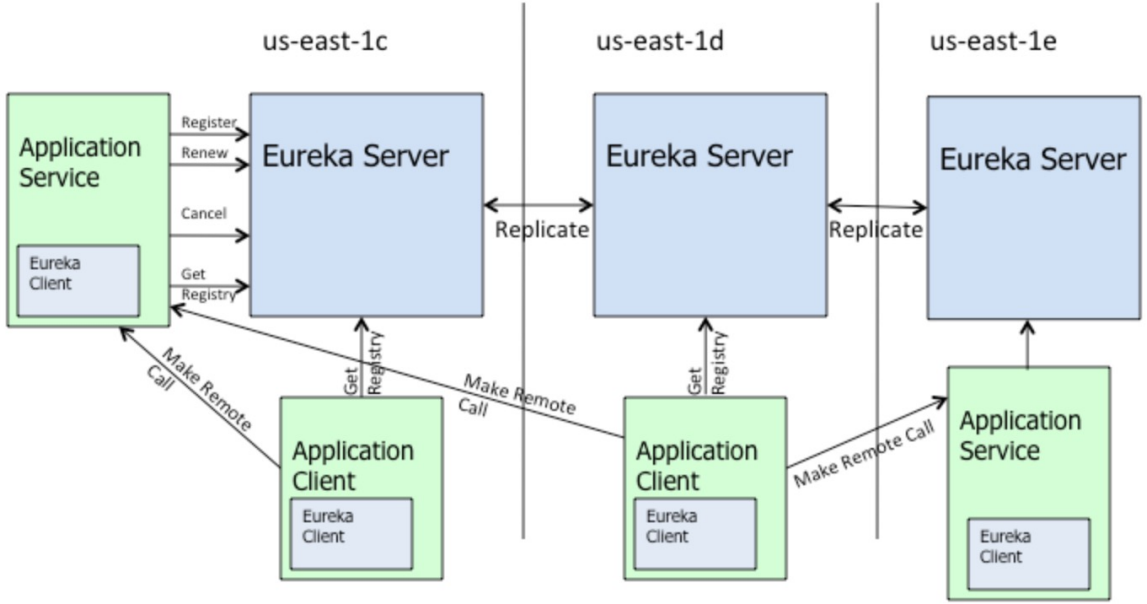
Service Registration & Discovery

- In the cloud,
  - applications can't always know the exact location of other services.
  - A service registry, such as Netflix Eureka, or a sidecar solution, such as HashiCorp Consul, can help.
  - Spring Cloud provides DiscoveryClient implementations for popular registries such as Eureka, Consul, Zookeeper, and even Kubernetes' built-in system.
  - There's also a Spring Cloud Load Balancer to help you distribute the load carefully among your service instances.

Eureka

- Eureka is
  - a REST (Representational State Transfer) based service that is primarily used in the AWS cloud for locating services for the purpose of load balancing and failover of middle-tier servers.
  - We call this service, the **Eureka Server**.
  - Eureka also comes with a Java-based client component, the **Eureka Client**, which makes interactions with the service much easier.
  - The client also has a built-in load balancer that does basic round-robin load balancing.
  - At Netflix, a much more sophisticated load balancer wraps Eureka to provide weighted load balancing based on several factors like traffic, resource usage, error conditions etc. to provide superior resiliency.

- High-Level Architecture
  - There is **one** eureka cluster per **region** which knows only about instances in its region. There is at the least **one** eureka server per **zone** to handle zone failures.



Service Application

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
  </dependency>
<dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Service Application

- Main java class

```
@EnableEurekaServer
@SpringBootApplication
public class ServiceRegistrationAndDiscoveryServiceApplication {

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

Service Application

- resources/application.properties

```
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

Client Application

- pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  </dependency>
<dependencies>
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Client Application

- Main java class

```
@SpringBootApplication
public class ServiceRegistrationAndDiscoveryClientApplication {

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

@RestController
class ServiceInstanceRestController {

  @Autowired
  private DiscoveryClient discoveryClient;

  @RequestMapping("/service-instances/{applicationName}")
  public List<ServiceInstance> serviceInstancesByApplicationName(
      @PathVariable String applicationName) {
    return this.discoveryClient.getInstances(applicationName);
  }
}
```

Client Application

- resources/application.properties

```
spring.application.name=a-bootiful-client
```

Test the Application

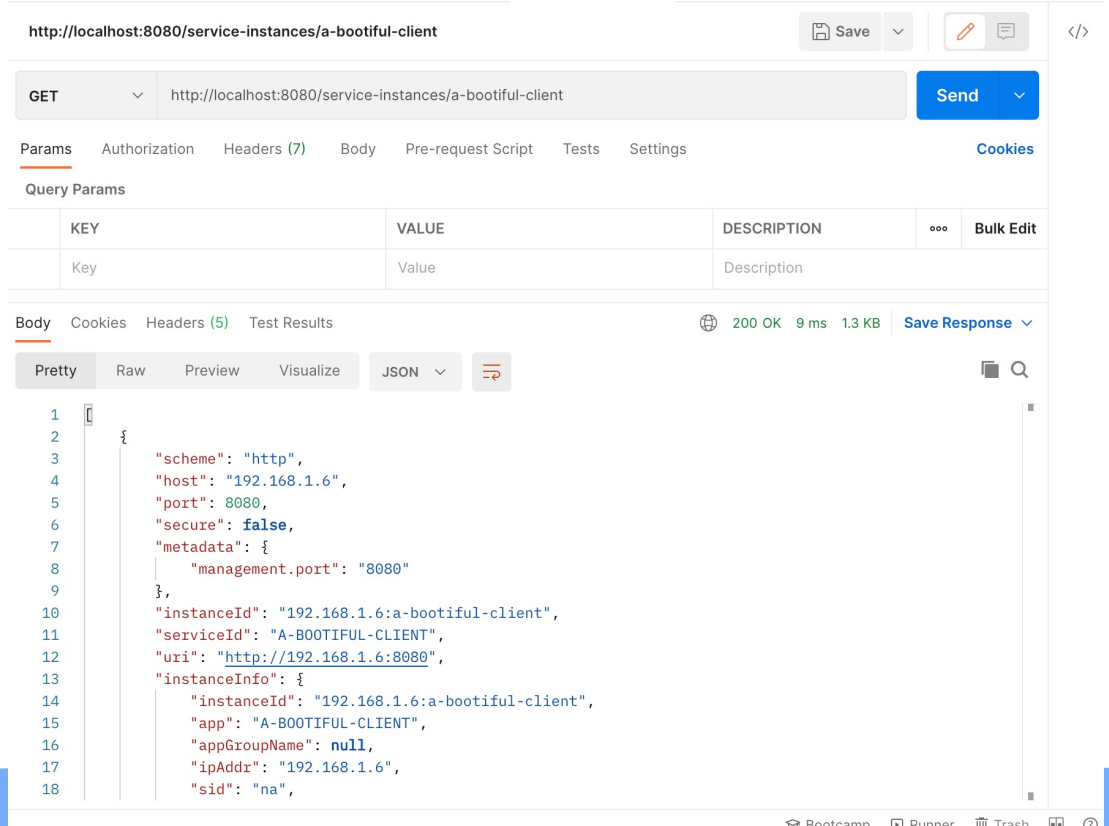
- http://localhost:8080/service-instances/a-bootiful-client


```
[{"instanceId":"172.20.10.6:a-bootiful-client","serviceId":"A-BOOTIFUL-
CLIENT","uri":"http://172.20.10.6:8080","instanceInfo":
{"instanceId":"172.20.10.6:a-bootiful-client","app":"A-BOOTIFUL-
CLIENT","appGroupName":null,"ipAddr":"172.20.10.6","sid":"na","homePageUrl":"htt
p://172.20.10.6:8080/","statusPageUrl":"http://172.20.10.6:8080/actuator/info","
healthCheckUrl":"http://172.20.10.6:8080/actuator/health","secureHealthCheckUrl"
:null,"vipAddress":"a-bootiful-client","secureVipAddress":"a-bootiful-
client","countryId":1,"dataCenterInfo":
{"@class":"com.netflix.appinfo.InstanceInfo$DefaultDataCenterInfo","name":"MyOwn
"},"hostName":"172.20.10.6","status":"UP","overriddenStatus":"UNKNOWN","leaseInf
o":
{"renewalIntervalInSecs":30,"durationInSecs":90,"registrationTimestamp":16210586
04835,"lastRenewalTimestamp":1621058604835,"evictionTimestamp":0,"serviceUpTimes
tamp":1621058604235},"isCoordinatingDiscoveryServer":false,"metadata":
{"management.port":"8080"},"lastUpdatedTimestamp":1621058604835,"lastDirtyTimest
amp":1621058604180,"actionType":"ADDED","asgName":null},"scheme":"http","host":"
172.20.10.6","port":8080,"metadata":{"management.port":"8080"},"secure":false}]
```

Test the Application

REliable, INtelligent & Scalable Systems

- http://localhost:8080/service-instances/a-bootiful-client



Spring Cloud Gateway

- pom.xml

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

Spring Cloud Gateway

- Main Java Class

```
@SpringBootApplication
public class GatewayApplication {
  @Bean
  public RouteLocator myRoutes(RouteLocatorBuilder builder) {
    return builder.routes()
        // Add a simple re-route from: /get to: http://httpbin.org:80
        // Add a simple "Hello:World" HTTP Header
        .route(p -> p
            .path("/get") // intercept calls to the /get path
            .filters(f -> f.addRequestHeader("Hello", "World")) // add header
            .uri("http://httpbin.org:80")) // forward to httpbin
        .build();
  }

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

Test the Gateway

REliable, INtelligent & Scalable Systems

- http://localhost:8080/get


```
{
  "args": {},
  "headers": {
    "Accept":
"text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
,application/signed-exchange;v=b3;q=0.9",
    "Accept-Encoding": "gzip, deflate, br",
    "Accept-Language": "zh-CN,zh;q=0.9,en;q=0.8",
    "Cache-Control": "max-age=0",
    "Content-Length": "0",
    "Forwarded": "proto=http;host=\"localhost:8080\";for=\"0:0:0:0:0:0:0:1:57682\"",
    "Hello": "World",
    "Host": "httpbin.org",
    "Sec-Ch-Ua": "\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"90\", \"Google Chrome\";v=\"90\"",
    "Sec-Ch-Ua-Mobile": "?0",
    "Sec-Fetch-Dest": "document",
    "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "none",
    "Sec-Fetch-User": "?1",
    "Upgrade-Insecure-Requests": "1",
    "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/90.0.4430.212 Safari/537.36",
    "X-Amzn-Trace-Id": "Root=1-609f744e-42b239bf33e7115d666cf18a",
    "X-Forwarded-Host": "localhost:8080"
  },
  "origin": "0:0:0:0:0:0:0:1, 223.104.213.94",
  "url": "http://localhost:8080/get"
}
```

Netflix Zuul Gateway

- This guide walks you through the process of routing and filtering requests to a microservice application by using the Netflix Zuul edge service library.

- You will write a simple microservice application and then build a reverse proxy application that uses Netflix Zuul to forward requests to the service application.
  - You will also see how to use Zuul to filter requests that are made through the proxy service.

Routing and Filtering – Book App

- Main Java Class

```
@RestController
@SpringBootApplication
public class RoutingAndFilteringBookApplication {

  @RequestMapping(value = "/available")
  public String available() {
    return "Spring in Action";
  }

  @RequestMapping(value = "/checked-out")
  public String checkedOut() {
    return "Spring Boot in Action";
  }

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

Routing and Filtering – Book App

- application.properties

```
spring.application.name=book

server.port=8090
```

Routing and Filtering – Edge Service

- Main Java Class

```
@EnableZuulProxy
@SpringBootApplication
public class RoutingAndFilteringGatewayApplication {

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

  @Bean
  public SimpleFilter simpleFilter() {
    return new SimpleFilter();
  }

}
```

Routing and Filtering – Edge Service

- SimpleFilter

```
public class SimpleFilter extends ZuulFilter {

  private static Logger log = LoggerFactory.getLogger(SimpleFilter.class);
  @Override
  public String filterType() {
    return "pre";
  }

  @Override
  public int filterOrder() {
    return 1;
  }

  @Override
  public boolean shouldFilter() {
    return true;
  }

  @Override
  public Object run() {
    RequestContext ctx = RequestContext.getCurrentContext();
    HttpServletRequest request = ctx.getRequest();

    log.info(String.format("%s request to %s", request.getMethod(), request.getRequestURL().toString()));

    return null;
  }

}
```

Routing and Filtering – Edge Service

- application.properties

```
zuul.routes.books.url=http://localhost:8090

ribbon.eureka.enabled=false

server.port=8080
```

Routing and Filtering – Edge Service

- pom.xml

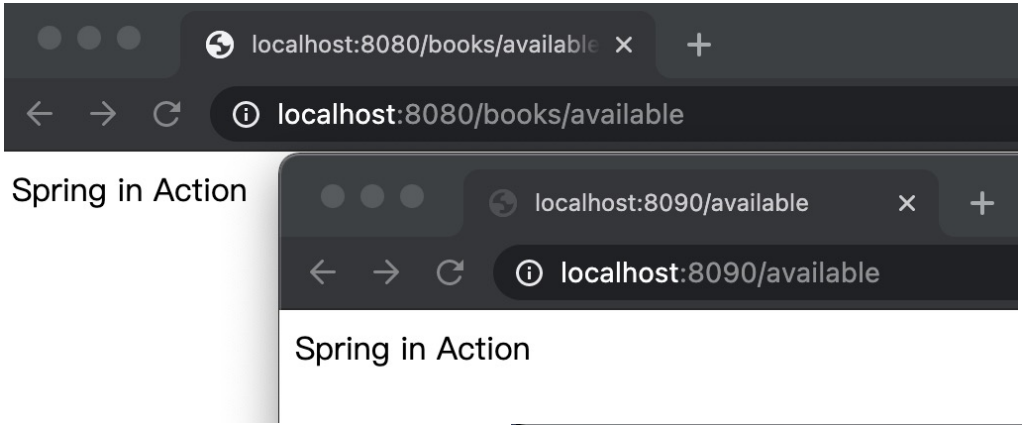
```
<properties>
  <java.version>8</java.version>
  <spring-cloud.version>Hoxton.SR9</spring-cloud.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
  </dependency>
```

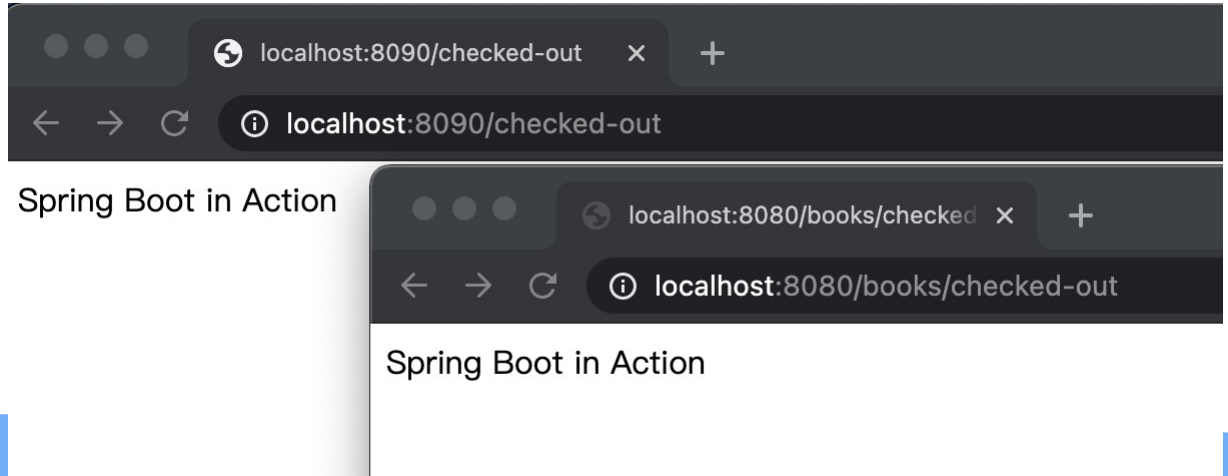
Test Netflix Zuul Gateway

- You can access the book application
  - directly at localhost:8090/available and
  - through the Gateway service at localhost:8080/books/available.
- Visit one of the Book service endpoints
  - (localhost:8080/books/available or localhost:8080/books/checked-out)
  - and you should see your request's method logged by the Gateway application before it is handed on to the Book application, as the following sample logging output shows:

```
2019-10-02 10:58:34.694 INFO 11608 --- [nio-8080-exec-4] c.e.r.filters.pre.SimpleFilter : GET request to http://localhost:8080/books/available
```

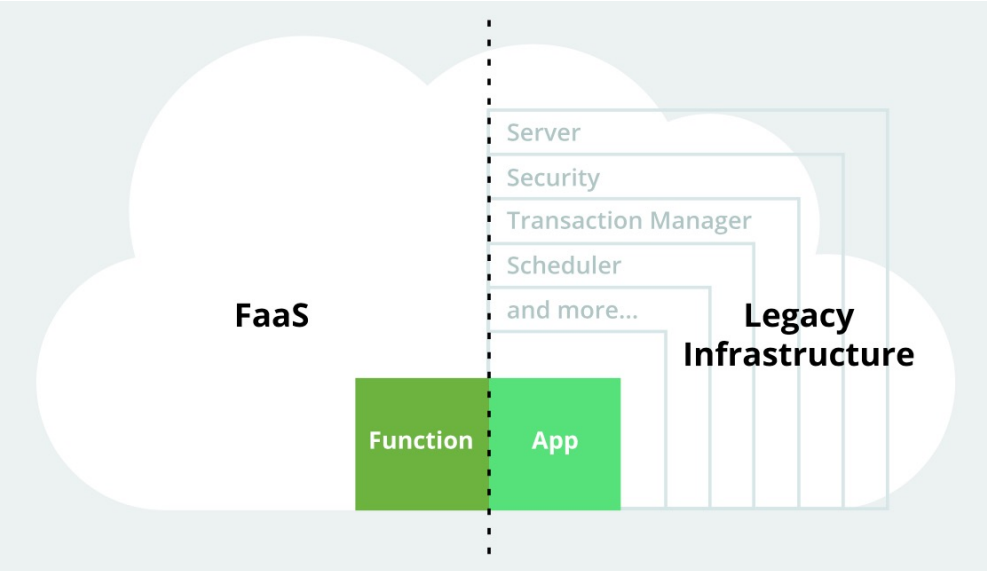
Test Netflix Zuul Gateway





Serverless

- Serverless applications
  - take advantage of modern cloud computing capabilities and abstractions to let you focus on logic rather than on infrastructure.
  - In a serverless environment, you can concentrate on writing application code while the underlying platform takes care of scaling, runtimes, resource allocation, security, and other “server” specifics.



Serverless

- What is serverless?
  - Serverless workloads are "event-driven workloads that aren't concerned with aspects normally handled by server infrastructure."
  - Concerns like "how many instances to run" and "what operating system to use" are all managed by a Function as a Service platform (or FaaS), leaving developers free to focus on business logic.

- Serverless characteristics?
  - Serverless applications have a number of specific characteristics, including:
    - Event-driven code execution with triggers
    - Platform handles all the starting, stopping, and scaling chores
    - Scales to zero, with low to no cost when idle
    - Stateless

Spring Cloud Function

- Spring Cloud Function
  - provides capabilities that lets Spring developers take advantage of serverless or FaaS platforms.
- Spring Cloud Function
  - provides adaptors so that you can run your functions on the most common FaaS services including :
  - Amazon Lambda, Apache OpenWhisk, Microsoft Azure, and Project Riff.

- Spring Cloud Function is a project with the following high-level goals:
  - Promote the implementation of business logic via functions.
  - Decouple the development lifecycle of business logic from any specific runtime target so that the same code can run as a web endpoint, a stream processor, or a task.
  - Support a uniform programming model across serverless providers, as well as the ability to run standalone (locally or in a PaaS).
  - Enable Spring Boot features (auto-configuration, dependency injection, metrics) on serverless providers.

Spring Cloud Function

- The java.util.function package from core Java serves as the foundation of the programming model used by Spring Cloud Function.
- In a nutshell, Spring Cloud Function provides:
  - Choice of programming styles: reactive, imperative, or hybrid.
  - Function composition and adaptation (such as composing imperative functions with reactive).
  - Support for reactive function with multiple inputs and outputs to let functions handle merging, joining, and other complex streaming operations.
  - Transparent type conversion of inputs and outputs.
  - Packaging functions for deployments, specific to the target platform (such as Project Riff, AWS Lambda, and more; see below).
  - Functions with flexible signatures (POJO functions) - “if it looks like a function, it’s a function”
  - All other benefits of Spring's idioms and programming model.

Function Sample

- Main Java Class

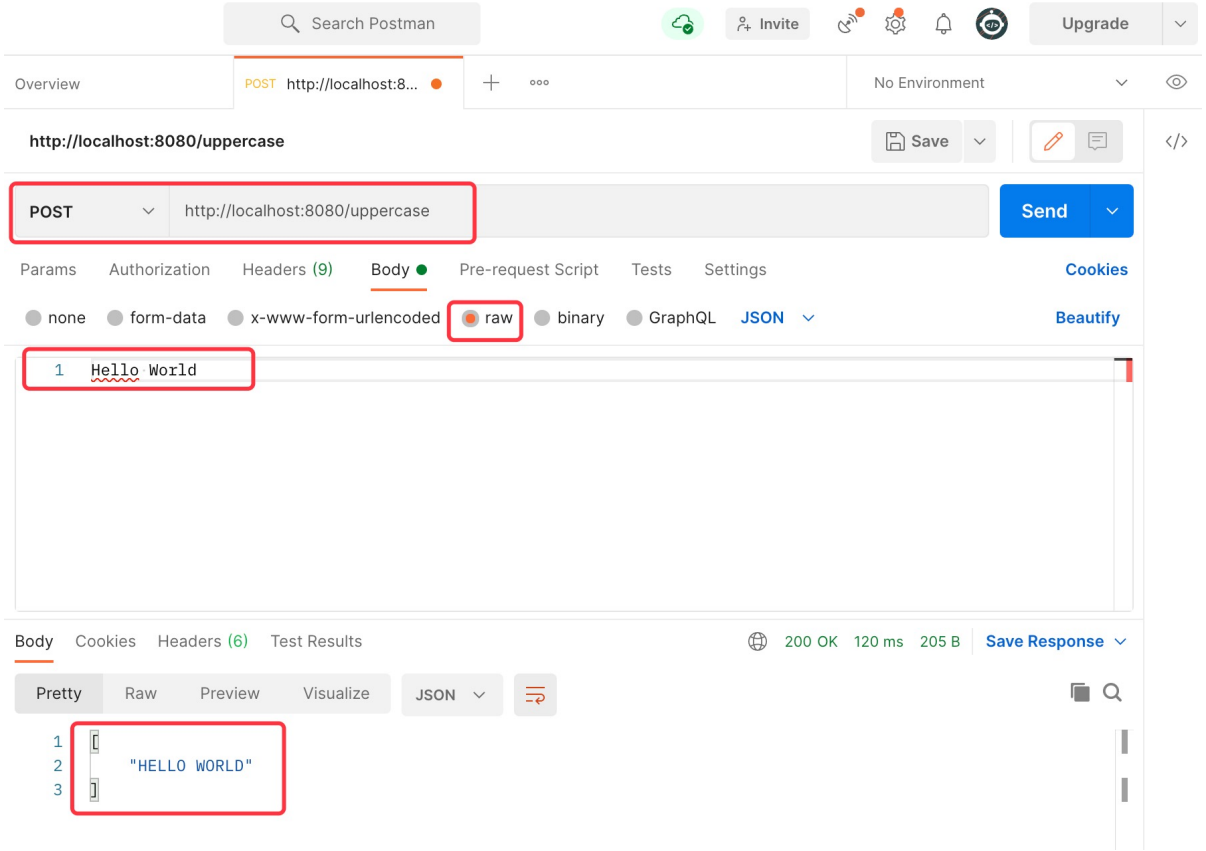
```
@SpringBootApplication
public class FunctionsampleApplication {

   @Bean
   public Function<Flux<String>, Flux<String>> uppercase() {
      return flux -> flux.map(value -> value.toUpperCase());
   }

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

}
```

Function Sample

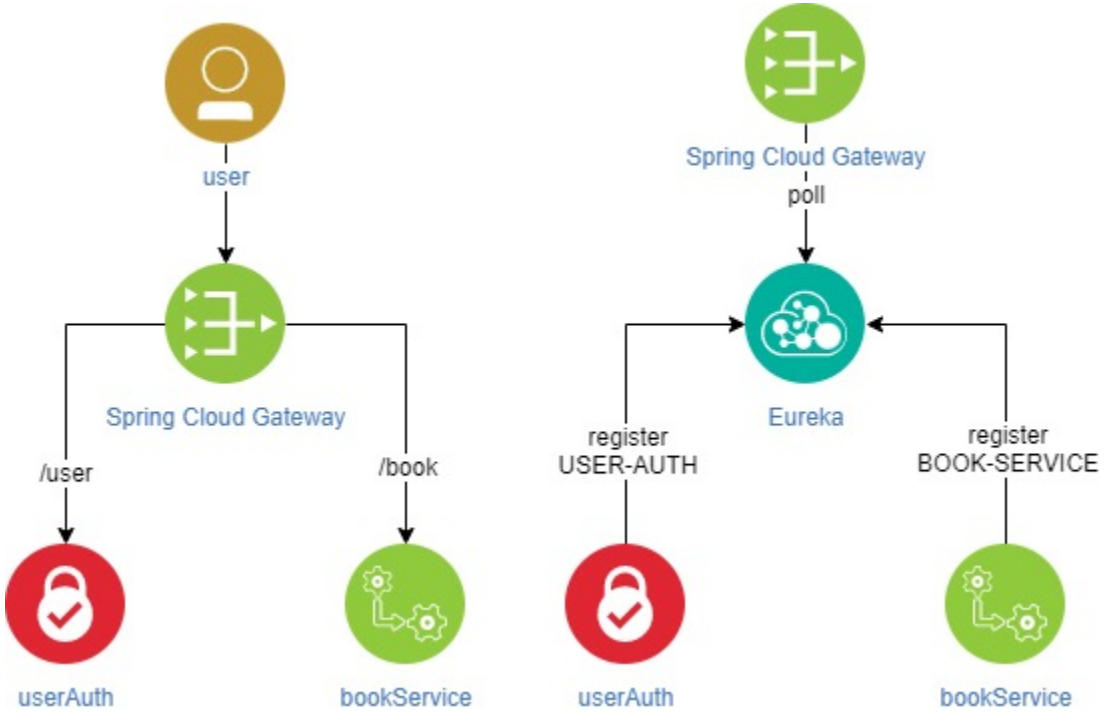
- pom.xml

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-function-web</artifactId>
</dependency>


<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Function Sample



Microservice Demo



Microservice Demo Eureka

- application.yaml

```yaml
server:
  port:
    8040
spring:
  application:
    name: Eureka
eureka:
  instance:
    prefer-ip-address: true
  client:
    fetch-registry: false
    register-with-eureka: false
    serviceUrl:
      defaultZone: http://localhost:8040/eureka
```

Microservice Demo Eureka

- EurekaApplication

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

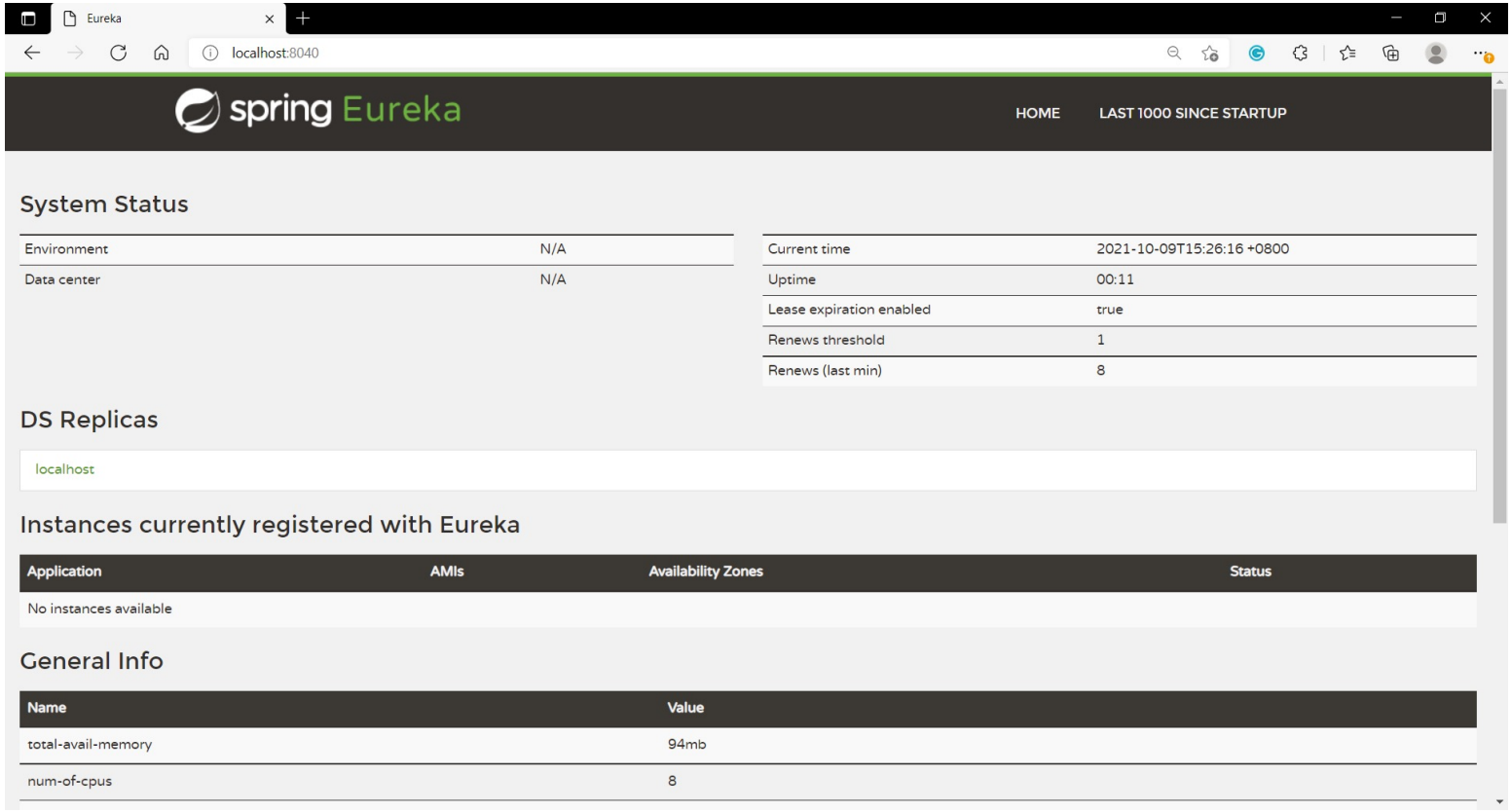
Microservice Demo Eureka

- EurekaApplication

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

Microservice Demo Eureka

REliable, INtelligent & Scalable Systems



Microservice Demo Book Service

- application.yaml

```yaml
spring:
  application:
    name: book-service

eureka:
  instance:
    prefer-ip-address: true
    ip-address: localhost
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://localhost:8040/eureka

server:
  port: 11130
```

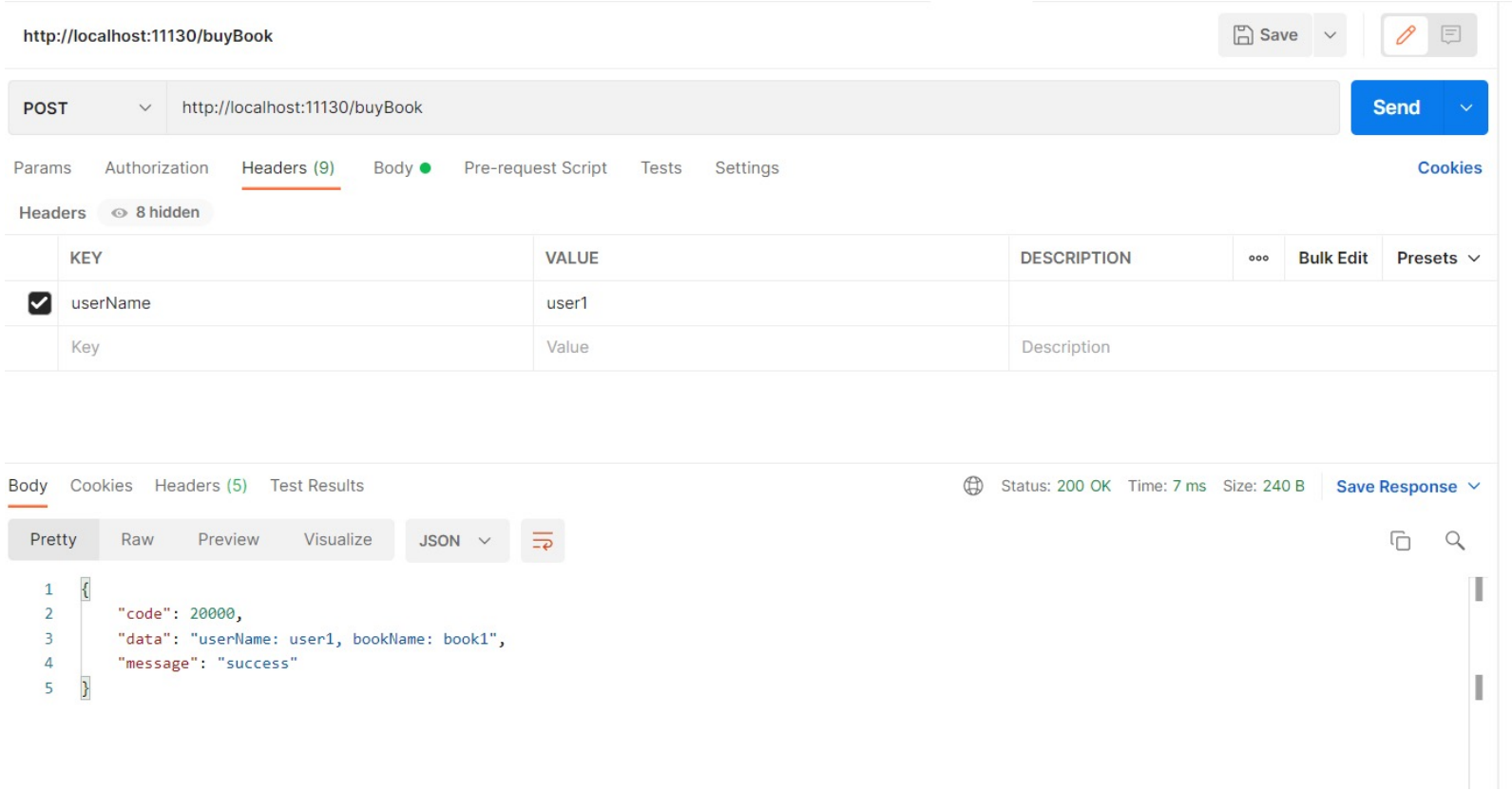
Microservice Demo Book Service

- BookController

```
@Slf4j
@RestController
public class BookController {
    @RequestMapping("/buyBook")
    public ResultDTO checkAuth(@RequestHeader("userName") String userName,
                               @RequestParam("bookName") String bookName){
        log.info("userName: {}, bookName: {}", userName, bookName);
        return ResultDTO.success(String.format("userName: %s, bookName: %s", userName, bookName));
    }
}
```

Microservice Demo Book Service

REliable, INtelligent & Scalable Systems



Microservice Demo User Authorize

- application.yaml

```yaml
spring:
  application:
    name: user-auth

eureka:
  instance:
    prefer-ip-address: true
    ip-address: localhost
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://localhost:8040/eureka

server:
  port: 11230
```

Microservice Demo User Authorize

- UserinfoController

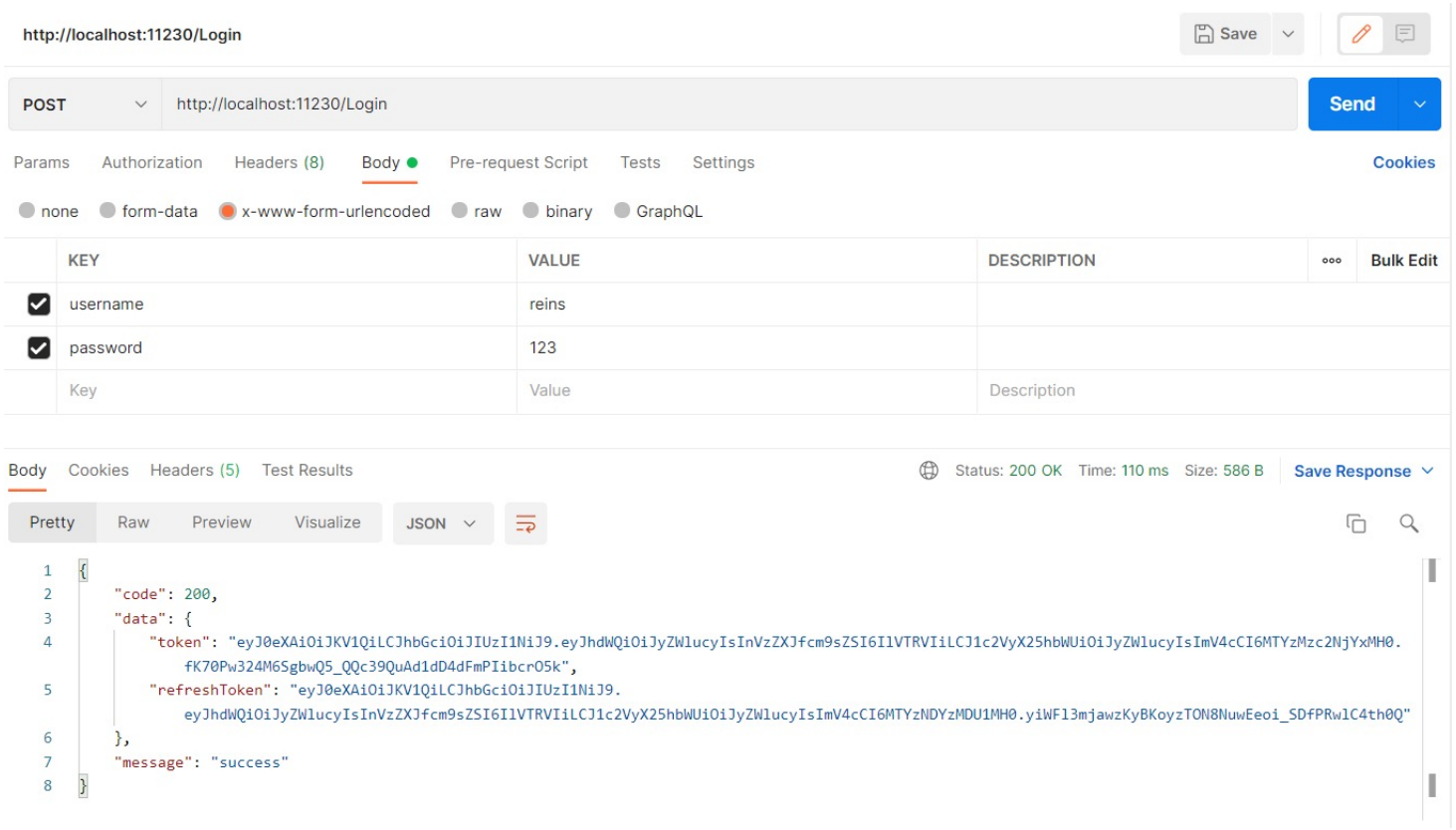
```
@ResponseBody
@PostMapping("/Login")
public Response login(Userinfo userinfo){
    return  userinfoService.verify(userinfo);
}
```

Microservice Demo User Authorize

- UserinfoService

```
public Response verify(Userinfo userinfo) {
    Response response = new Response();
    QueryWrapper<Userinfo> queryWrapper = new QueryWrapper<>();
    queryWrapper.eq("username", userinfo.getUsername());

    if(usernameExist(userinfo.getUsername())){
        Userinfo _userinfo = userinfoMapper.selectOne(queryWrapper);
        if(Objects.equals(userinfo.getPassword(), _userinfo.getPassword())) {
            response.setCode(200);
            response.setMessage("success");
            response.setData(createTokenPair(_userinfo));
        } else …
    } else …
    return  response;
}
```

Microservice Demo User Authorize



Microservice Demo Gateway

- application.yaml

```yaml
server:
  port: 8080
  error:
    include-message: always
spring:
  cloud:
    gateway:
      globalcors:
        cors-configurations:
          '[/**]':
            allowedOrigins: "*"
            allowedMethods:
              - GET
              - POST
  application:
    name: gateway
```

Microservice Demo Gateway

- application.yaml

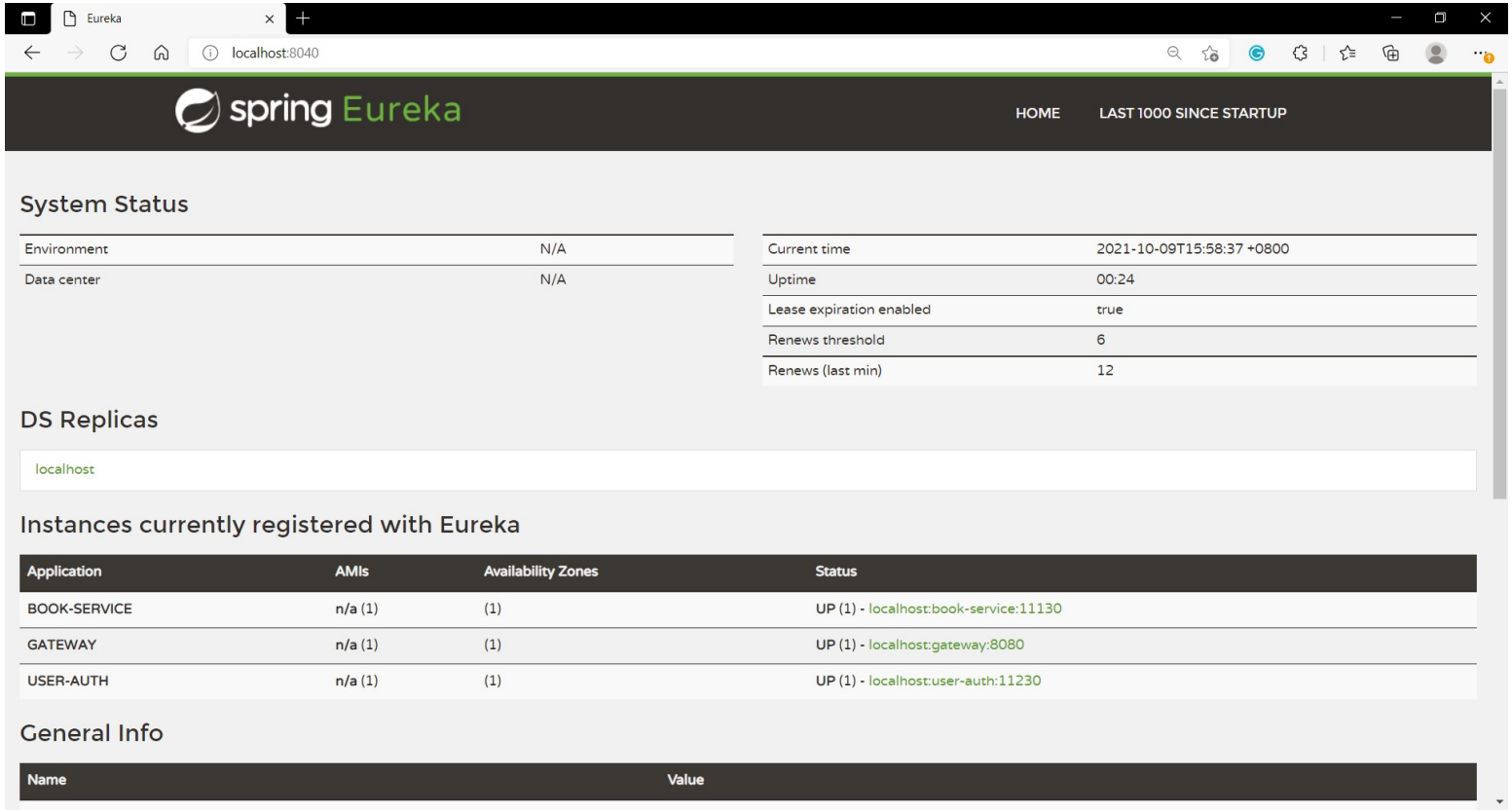
```
eureka:
  instance:
    prefer-ip-address: true
    ip-address: localhost
  client:
    registerWithEureka: true
    fetchRegistry: true
    serviceUrl:
      defaultZone: http://localhost:8040/eureka
    eureka-service-url-poll-interval-seconds: 10
```

Microservice Demo Gateway

- MainGateway

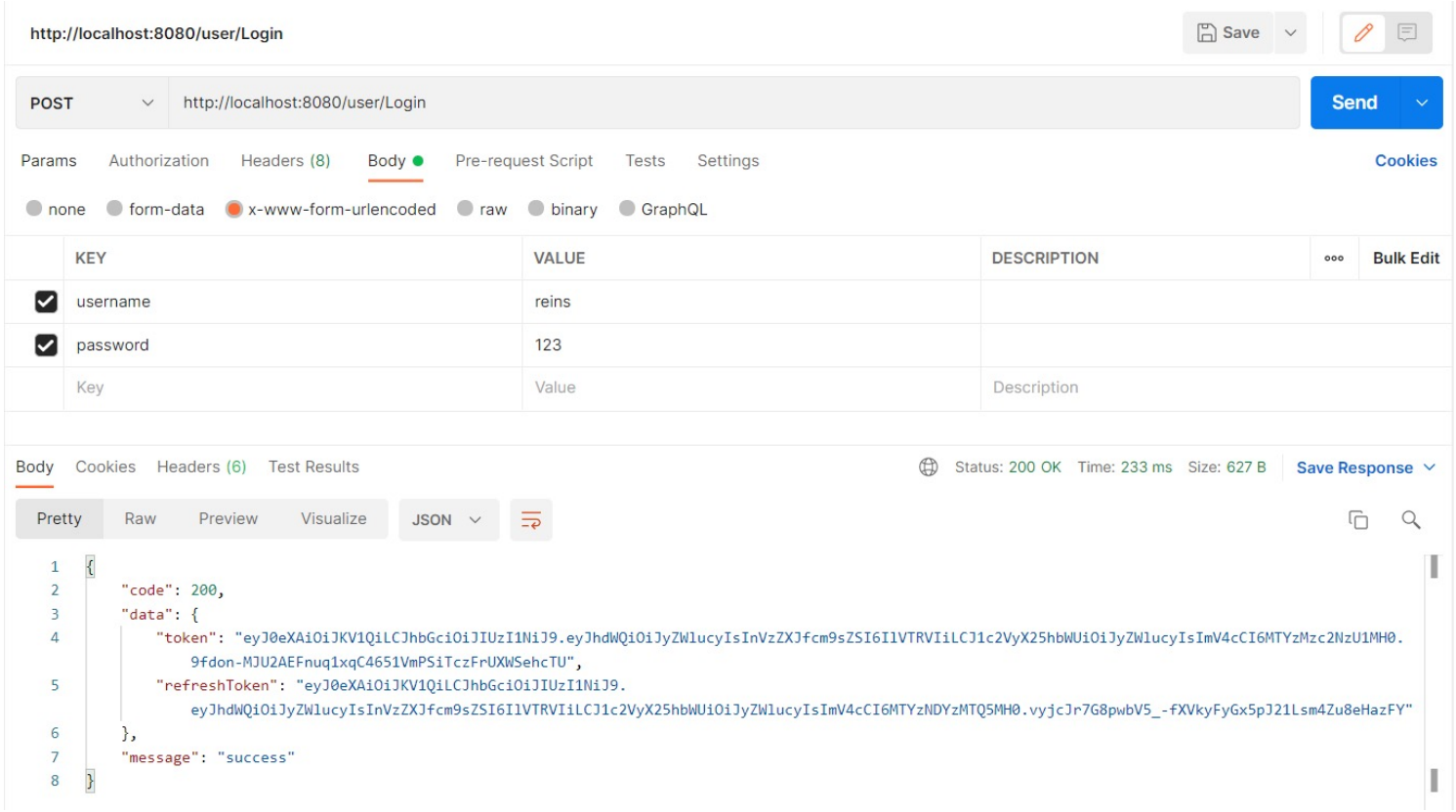
```
@SpringBootApplication
@EnableDiscoveryClient
@EnableEurekaClient
public class MainGateway {
    public static void main(String[] args) {
        SpringApplication.run(MainGateway.class, args);
    }
    @Autowired
    JwtCheckFilter jwtCheckFilter;
    @Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(r -> r.path("/book/**")
                        .filters(f -> f.rewritePath("/book","").filter(jwtCheckFilter))
                        .uri("lb://BOOK-SERVICE")
                ).route(r->r.path("/user/**")
                        .filters(f->f.rewritePath("/user",""))
                        .uri("lb://USER-AUTH")
                )
                .build();
    }
}
```

Microservice Demo Service Register



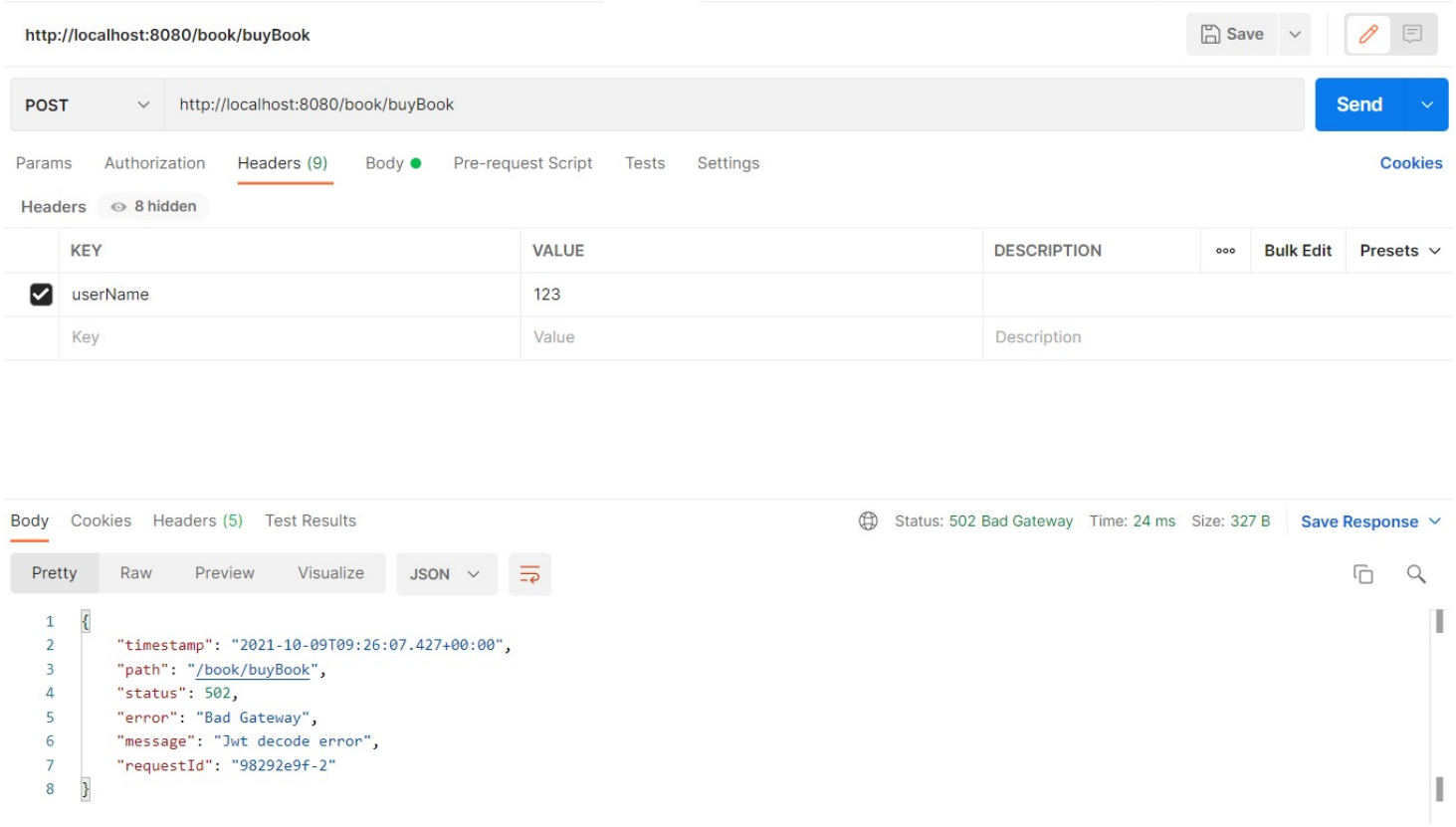
Microservice Demo Gateway

REliable, INtelligent & Scalable Systems



Microservice Demo Gateway

REliable, INtelligent & Scalable Systems



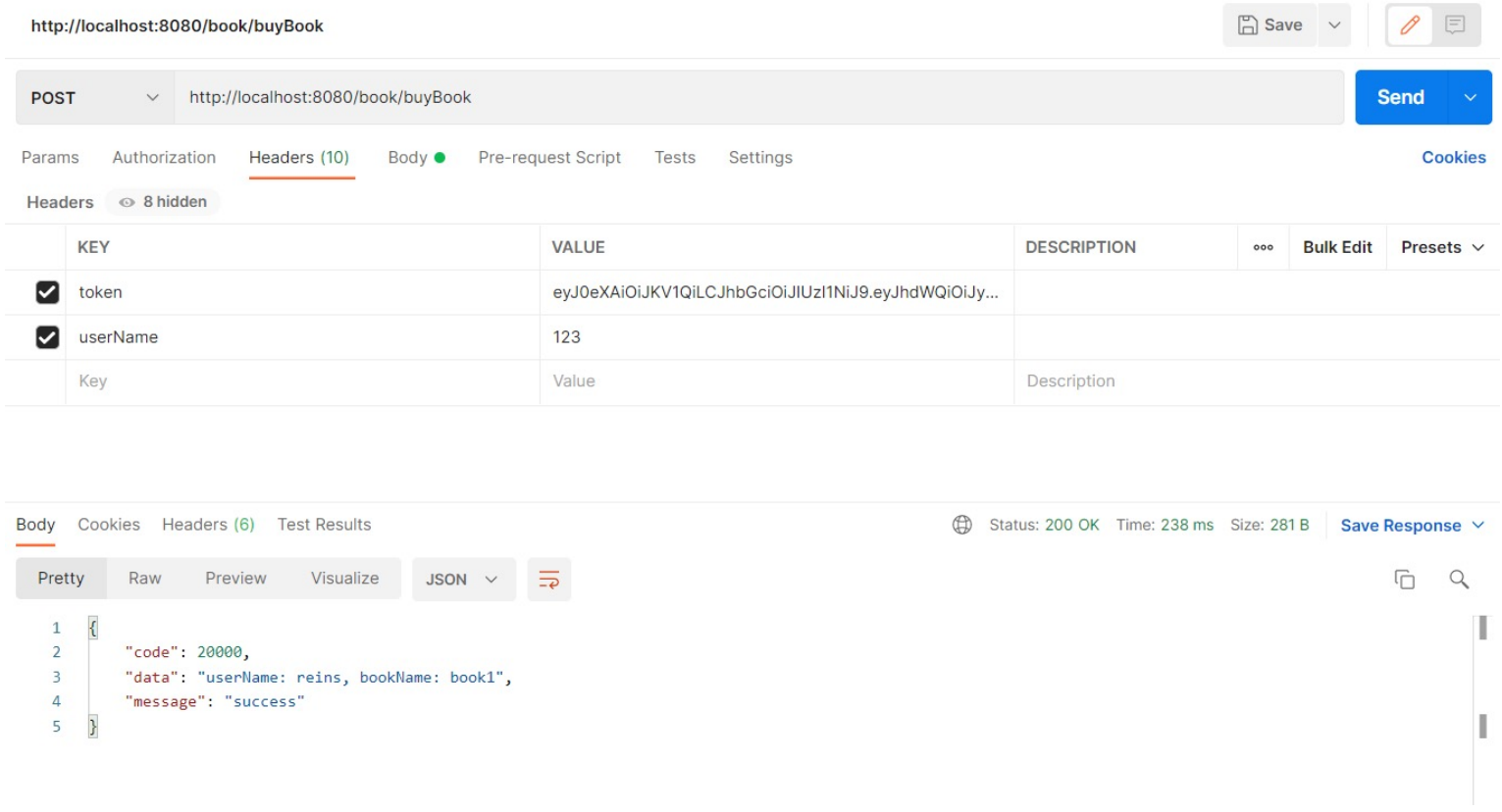
Microservice Demo Gateway

- JwtCheckFilter

```
public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
    String jwtToken = exchange.getRequest().getHeaders().getFirst("token");
    UserInfo userInfo = jwt.parseToken(jwtToken);
    System.out.println(userInfo);
    if(userInfo != null && userInfo.getUserRole()!= null && userInfo.getUsername()!=null) {
        ServerHttpRequest request = exchange.getRequest().mutate()
                .header("userName", userInfo.getUsername())
                .build();
        return chain.filter(exchange.mutate().request(request).build());
    }
    throw new ResponseStatusException(HttpStatus.BAD_GATEWAY, "Jwt decode error");
}
```

Microservice Demo Gateway

REliable, INtelligent & Scalable Systems



作业五(第一部分)

- 请你在大二开发的E-Book系统的基础上，完成下列任务：
  1. 下面两项任务你可以选择一项完成：
     ① 开发一个微服务，输入为书名，输出为书的作者。将此微服务单独部署，并使用netflix-zuul进行路由，在你的E-Book系统中使用该服务来完成作者搜索功能。
     ② 开发一个函数式服务，输入为订单中每种书的价格和数量，输出为订单的总价。将此函数式服务单独部署，并在你的E-Book系统中使用该服务来完成作者搜索功能。

  – 请将你编写的相关代码整体压缩后上传，请勿压缩整个工程提交。

- 评分标准：
  1. 能够正确实现上述搜索/计费功能 。(3分)

References

- Microservices
  - https://spring.io/microservices
- 【微服务】使用spring cloud搭建微服务框架，整理学习资料
  - https://www.cnblogs.com/ztfjs/p/9230374.html
- 一文详解微服务架构
  - https://www.zhihu.com/question/65502802
- Eureka
  - https://github.com/Netflix/eureka
  - https://github.com/Netflix/eureka/wiki/Eureka-at-a-glance
- Service Registration & Discovery
  - https://spring.io/guides/gs/service-registration-and-discovery/
- Routing and Filtering
  - https://spring.io/guides/gs/routing-and-filtering/
- Getting Started with Spring Cloud Gateway
  - https://spring.io/blog/2019/06/18/getting-started-with-spring-cloud-gateway

References

- Serverless
  - https://spring.io/serverless
- Spring Cloud Function GitHub's repository
  - https://github.com/spring-cloud/spring-cloud-function

Thank You!