

Tópicos em Frameworks

Micro Serviços - Cássio Seffrin 2019

<https://github.com/cassioseffrin/projetoAulaSpring>

Micro Serviços

CARACTERÍSTICAS

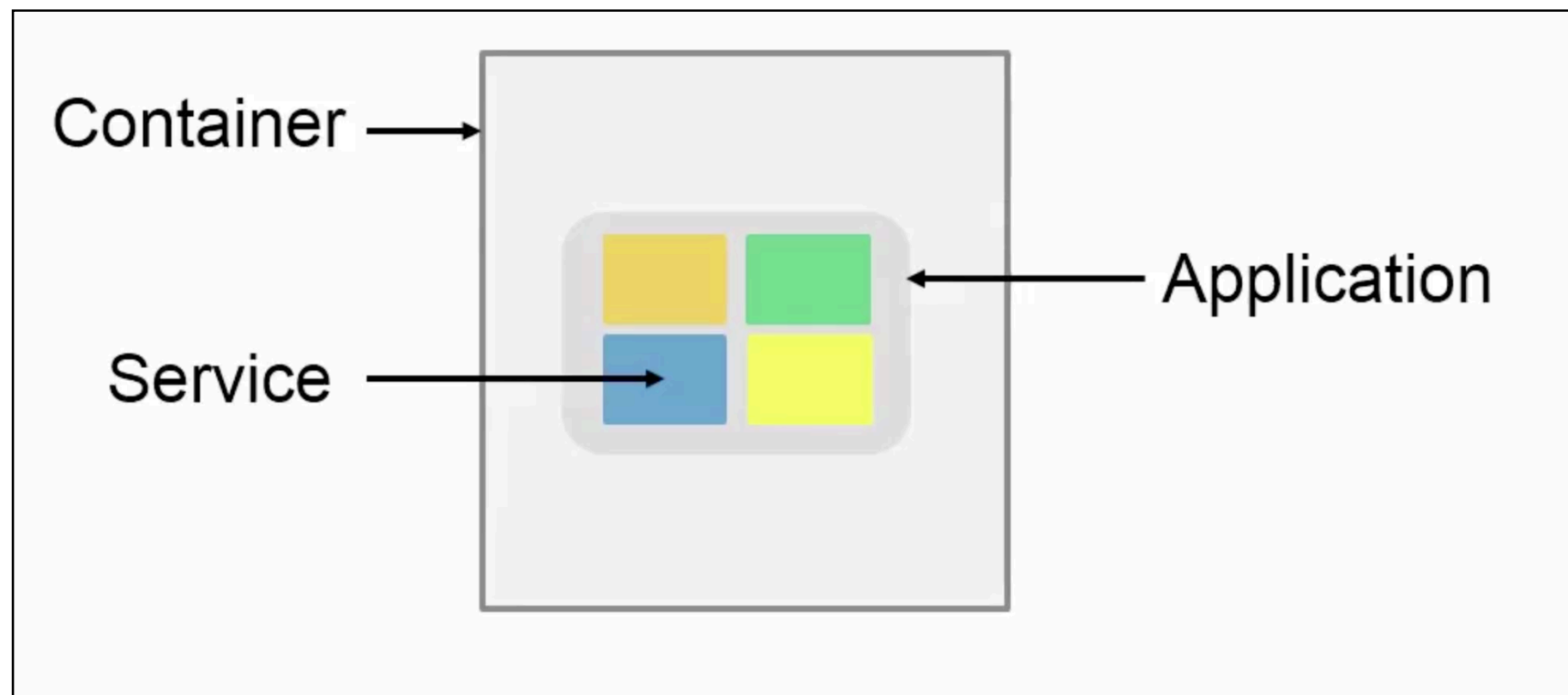
- Pequenos serviços independentes organizados em torno dos recursos de negócios
- O APIS expõe o serviço e seus recursos
- Os serviços interagem em diferentes processos
- Os armazenamentos de dados são focados e específicos para o serviço. Cada serviço tem seu próprio data-source
- Sem estado
- Tolerantes a Falha

Micro serviços

BENEFÍCIOS

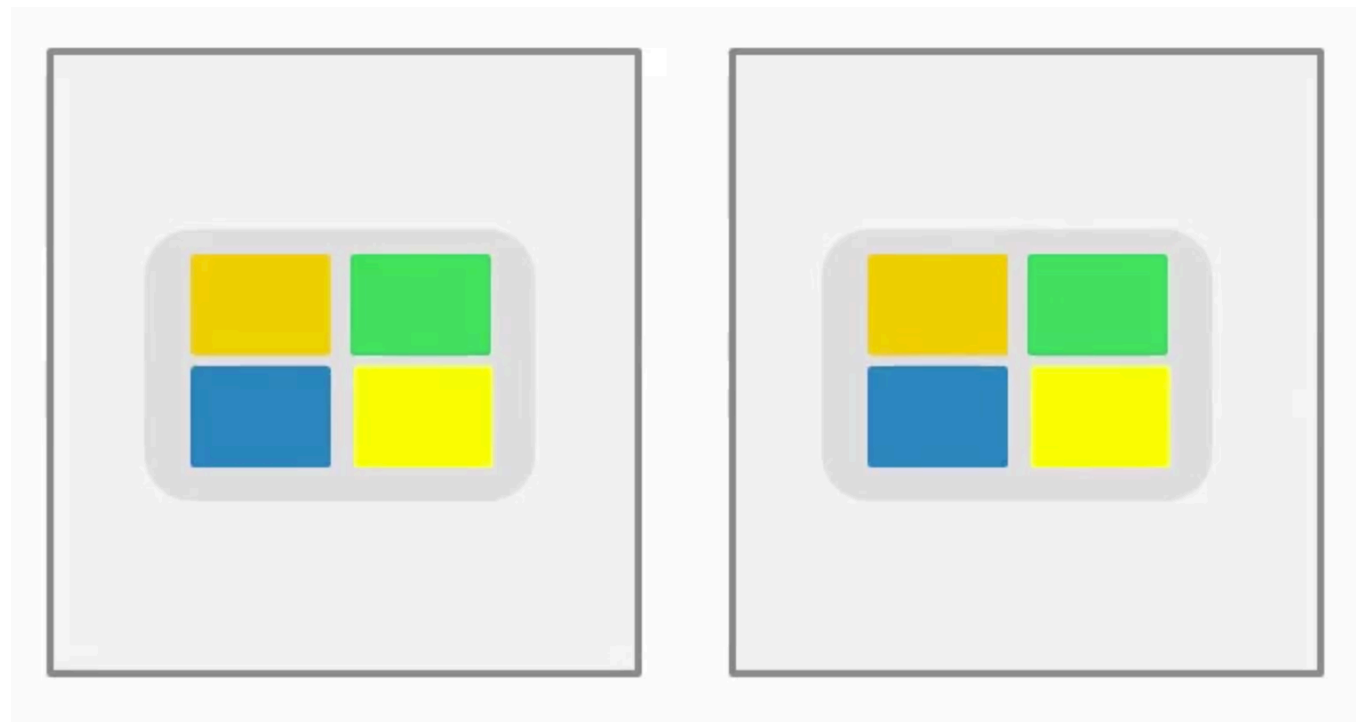
- Separa os recursos de negócios
- Os serviços são fracamente acoplados e modulares
- Altamente escalável e substituível
- Gerenciamento de serviços independente
- Suporta processamento simultâneo
- Permite equipes menores e menor tempo de colocação no mercado

Monolitico vs micro serviço



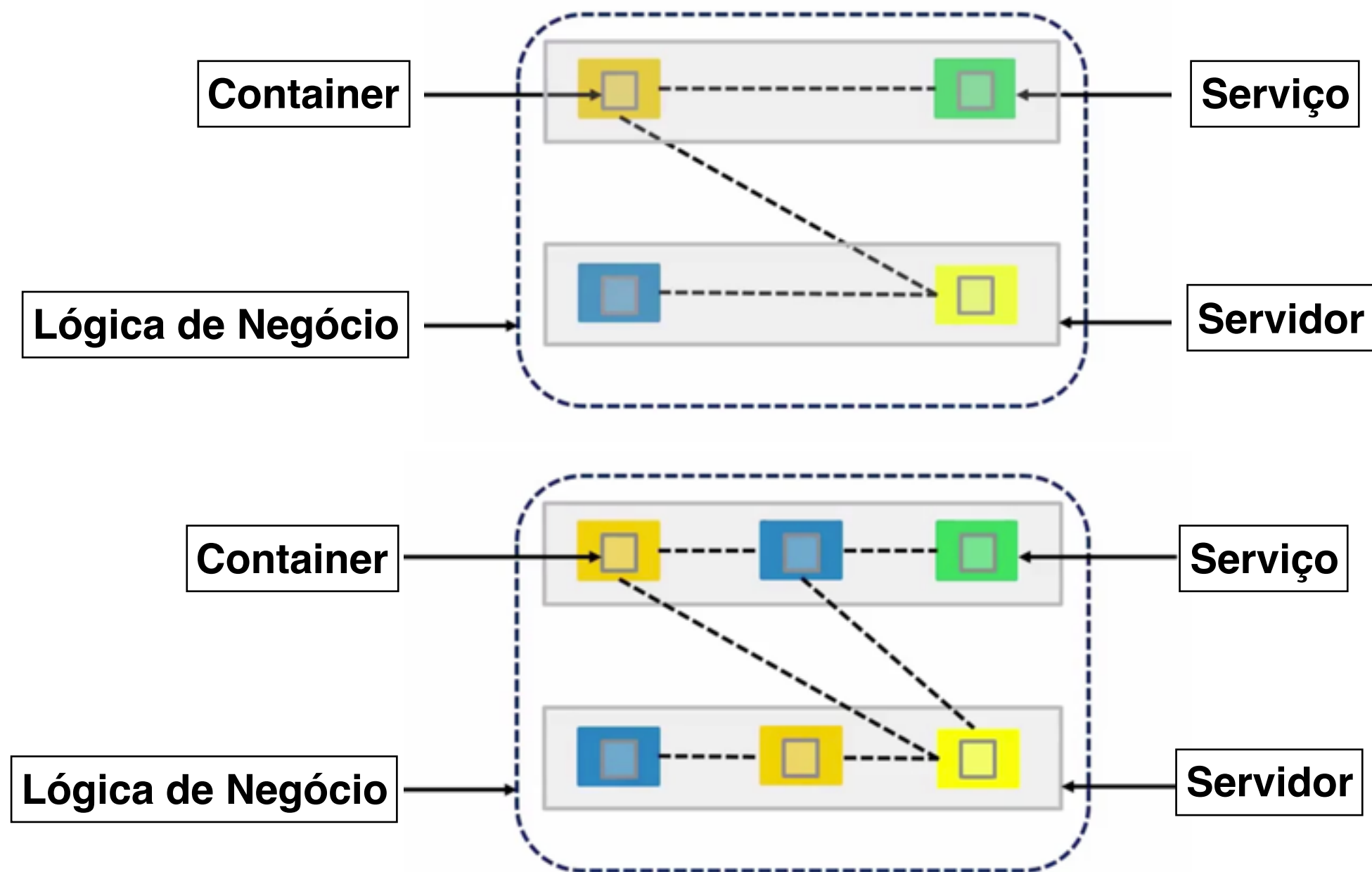
Monolítico como escalar

1. CPU, memória etc..
2. Horizontalmente , dificuldade de manipular sessões



Escalando micro serviços

Definição: Um projeto de software que foca na integração de um grupo de serviços leves e isolados através de comunicação em camadas para formar uma aplicação.



Micro serviços

- A arquitetura de micro serviços sugere que cada serviço deve manipular seus próprios dados. Portanto, qualquer serviço (Serviço A) dependente de dados pertencentes a outro serviço (serviço B) deve acessar esses dados não fazendo chamadas diretas ao banco de dados, mas através da API fornecida pelo segundo serviço (serviço B).
- Então, Como lidar com relacionamentos de chave estrangeira?

Micro serviços

- É possível usar um banco de dados compartilhado para vários micro serviços.
Os padrões para gerenciamento de dados de microsserviços:
<http://microservices.io/patterns/data/database-per-service.html>
- Micro serviços mais autônomos. Nessa situação, você deve duplicar alguns dos seus dados entre vários micro serviços. Você pode compartilhar os dados com chamadas da API entre micro serviços ou com mensagens assíncronas. Depende da sua infraestrutura e frequência de alteração dos dados. Se não estiver mudando com frequência, você deve duplicar os dados com eventos assíncronos.

CORBA, RPC, RMI, EJB ...MICROSERVICES REST

Micro serviços

O Spring Rest Template provê um a forma bem fácil de chamar APIs REST

```
RestTemplate rest = new RestTemplate();  
String url = "http://localhost:8080/aluno/1";  
AlunoDTO aluno = rest.getForObject(url, AlunoDTO.class);
```

Inicializa ou DI

URL do serviço

executa chamda da URL, provendo uma Classe, o RestTemplate tomará cuidado da conversão

Micro serviços

REST Template

Método tradicional usando RestTemplate

```
@GetMapping("/pegarDadosFinanceiro/{id}")
public @ResponseBody AlunoDTO pegarDados(@PathVariable Long id) {
    RestTemplate rest = new RestTemplate();
    String url = "http://localhost:8080/aluno/" + id;
    AlunoDTO aluno = rest.getForObject(url, AlunoDTO.class);
    return aluno;
}
```

Micro serviços

Spring Cloud Feign

- Cliente REST Declarativo da Netflix
- Chamando serviços REST usando as bibliotecas Feign
- O que é o Feign ?
- Alternativa para o REST Template
- Permite fazer chamadas REST sem implementação de código
- Spring Cloud prove um wrapper para usar o Feign

Micro serviços

Spring Cloud Feign

Como funciona?

Definir interfaces para o código do cliente REST;

- Anotar interface como `@FeignClient` ;
- Anotar assinaturas de métodos com anotações do Spring MVC;
- Outras implementações como JAX/RS são possíveis, porém SpringMVC é mais indicado;

O Spring Cloud o implementará em tempo de execução

- Escaneia as interfaces
- Implementa automaticamente o código para chamar o serviço REST e processar resposta.

Micro serviços

Spring Cloud Feign

Anotar a aplicação com @EnableFeignClients

```
@SpringBootApplication
@EnableFeignClients("br.edu.cassio")
public class EscolaFinanceiroApplication {

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

}
```

Micro serviços

Feign Interface

usar o client-id do eureka,
(mais conveniente)

Criar uma interface (nao classe)

```
@FeignClient(url = "localhost:8080")
public interface EscolaMatriculaProxy {

    @GetMapping("/aluno/{id}")
    public AlunoDTO pegarDados(@PathVariable("id") Long id);

}
```

Criar uma classe Controller

```
@Autowired
private EscolaMatriculaProxy escolaMatriculaProxy;

@GetMapping("/pegarDadosFinanceiro-feign/{id}")
public @ResponseBody AlunoDTO pegarDadosFeign(@PathVariable Long id) {
    AlunoDTO aluno = escolaMatriculaProxy.pegarDados(id);
    return aluno;
}
```

Micro serviços

Spring Cloud Feign

Dependências Spring Initializr

- Spring Web
- Cloud Bootstrap
- Lombok
- MySQL Driver
- Spring Data JPA
- Config Client

Micro serviços

Spring Cloud Feign

Ou adicionar as seguintes dependências no pom.xml

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


Micro serviços

Feign

Ribbon e Eureka e Feign, onde se encaixam?

- O exemplo anterior - URL codificado:
`@FeignClient (url = "localhost:8080 “)`
 - ... use um "ID de cliente, client-id" Eureka:
`@FeignClient (name="msacademico")`
 - O Ribbon é ativado automaticamente
- Eureka dá ao nosso aplicativo todos os “clientes” que correspondem ao client-id fornecido - O Ribbon aplica balanceamento de carga automaticamente - Feign trata do código.

Micro serviços

Eureka

- Serviço Descoberta
- Eureka Server
- Descoberta do cliente

Micro serviços

Eureka

- Descoberta de Serviços: As arquiteturas de micro serviço resultam em um grande número de chamadas entre serviços
- É desafiador para configurar
- Como um aplicativo pode encontrar facilmente todas as outras dependências de tempo de execução?
- Configuração manual
- A descoberta de serviços prove um single lookup service (único serviço de 'pesquisa')
- Os clientes se registram, descobrem outros registrantes.
- Soluções: Eureka, PaceMaker, Etcd, Zookeeper, SmartStack, etc.

Micro serviços

Eureka descoberta de serviço cliente/servidor

- Parte do Spring Cloud Netflix testada pelo Netflix
- Eureka fornece um servidor de 'pesquisa'.
Geralmente tornado altamente disponível executando várias cópias
As cópias replicam o estado dos serviços registrados.
- Os Serviços "Cliente" são registrados no Eureka
Fornece metadados no host, porta, URL do indicador de integridade, etc.
- Os Serviços de Cliente enviam sinais (heartbeats) para o Eureka
Eureka remove os serviços sem heartbeats.

Micro serviços

Servidor Eureka

É uma simples aplicação spring boot web com uma dependência e uma anotação

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
</dependency>
```

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

Micro serviços

Servidor Eureka - múltiplos servidores

- Tipicamente vários servidores Eureka devem ser executados simultaneamente.
Caso contrário, você receberá avisos nos logs
Os servidores Eureka se comunicam entre si para compartilhar o estado
Fornece alta disponibilidade
- Cada o servidor deve saber a URL dos outros
Pode ser fornecido pelo server Config
Server One (JAR), vários perfis (profiles)



Micro serviços

Servidor Eureka - múltiplos servidores

Opções de configuração comuns para o Eureka Server:

<https://github.com/Netflix/eureka/wiki/Configuring-Eureka>

application

```
server:
  port: 8090

eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url: http://localhost:8091/eureka, http://localhost:8092/eureka

  instance:
    status-page-url-path: ${management.context Path}/info
    health-check-url-path: ${management.contextPath/health
    hostname: localhost
```

bootstrap.yml

```
spring:
  application:
    name: eureka-server
```

Micro serviços

Spring Boot application registrado com Eureka

Dependencia:

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>
```

```
Somente a localização do Eureka requer configuração explícita  
@SpringBootApplication  
@EnableDiscoveryClient  
public class Application {  
  .....  
}
```

application.yml:

```
spring:  
  application:  
    name: msacademico
```

application.properties:

```
#application.properties  
eureka.client.serviceUrl.defaultZone: http://server: 9000/eureka/
```

Default fallback.
Qualquer instância do Eureka (geralmente
algumas URLs separadas por vírgula)

Micro serviços

@EnableDiscoveryClient

Registra automaticamente o cliente no servidor Eureka
Registra o nome, host e porta do aplicativo


Usando valores do Spring Environment.
Mas pode ser substituído.

Atribua a seu aplicativo `spring.application.name`
Torna este aplicativo uma "instância" e um "cliente"

Ele pode localizar outros serviços do cliente

Service ID (Eureka VIP)
Corresponde a
`spring.application.name`

```
@Autowired
DiscoveryClient clientes;
@GetMapping("/servicos")
public @ResponseBody String getURI() {
    List<ServiceInstance> lstRecursosDaInstancia = clientes.getInstance("MSACADEMICO");
    if (lstRecursosDaInstancia != null && lstRecursosDaInstancia.size() > 0) {
        ServiceInstance instancia = lstRecursosDaInstancia.get(0);
        return instancia.getInstanceId();
    }
    return "servico nao encontrado";
}
```



Micro serviços

Resumo

- **Passive Service Discovery**
 - Ter serviços se registrar e encontrar outros automaticamente
- **Spring Cloud Eureka Server**
 - Mantém registros, compartilha informações sobre outros registrantes
Sincroniza-se com outros servidores
- **Spring Cloud Eureka Client**
 - Conecta-se ao servidor para registrar-se e obter informações sobre outros clientes.

Micro serviços - hands on

Novo Projeto com Spring Initializr

- Spring Boot Actuator
- Eureka Discovery Client
- Spring Web

Modificar a classe Application class. Adicionar @EnableDiscoveryClient.

bootstrap.yml

```
spring:  
  application:  
    name: sujeito
```

application.yml

```
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8090/eureka/  
  
server:  
  port: 8091  
  
adjetivos: linda, feia, fabulosa, feiosa, deslumbrante, perfumada
```

Micro serviços - hands on

Criar um controller para adjetivo, artigo, substantivo, sujeito e verbo

```
@RestController
public class Controller {
    @Value("${adjetivos}")
    public String adjetivos;

    @GetMapping("/")
    public @ResponseBody String getWord() {
        String[] arr = adjetivos.split(",");
        int i = (int) Math.round(Math.random() * (arr.length - 1));
        return arr[i];
    }
}
```

Micro serviços - hands on

Criar um controller para montar a sentença toda

```
@RestController
public class Controller {

    @Autowired
    DiscoveryClient clientes;

    @GetMapping("/frase")
    public @ResponseBody String getFrase() {
        return getFracaoFrase("SUJEITO") + " " + getFracaoFrase("VERBO") + " " +
            getFracaoFrase("ARTIGO") + " " + getFracaoFrase("SUBSTANTIVO") + " "
            + getFracaoFrase("ADJETIVO") + ".";
    }

    public String getFracaoFrase(String microServico) {
        List<ServiceInstance> listaServicos = clientes.getInstances(microServico);
        if (listaServicos != null && listaServicos.size() > 0) {
            URI uri = listaServicos.get(0).getUri();
            if (uri != null) {
                return (new RestTemplate()).getForObject(uri, String.class);
            }
        }
        return null;
    }
}
```

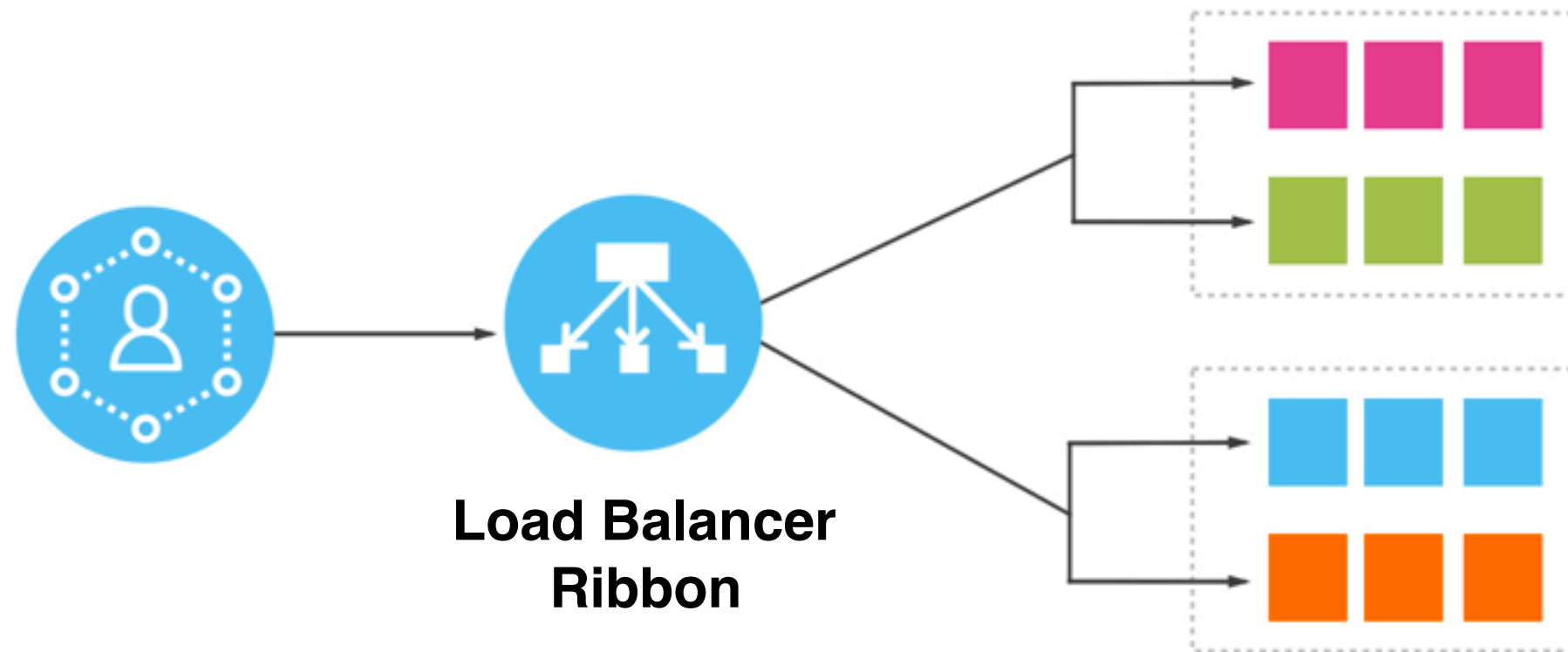
Micro serviços

Spring Cloud Ribbon

Balanceamento de carga no cliente

Tradicionalmente o balanceamento de carga fica no lado do servidor

Distribuindo trafego entre alguns servidores
software apache nginx HA Proxy ou (hardware F5, NSX, BigIP)

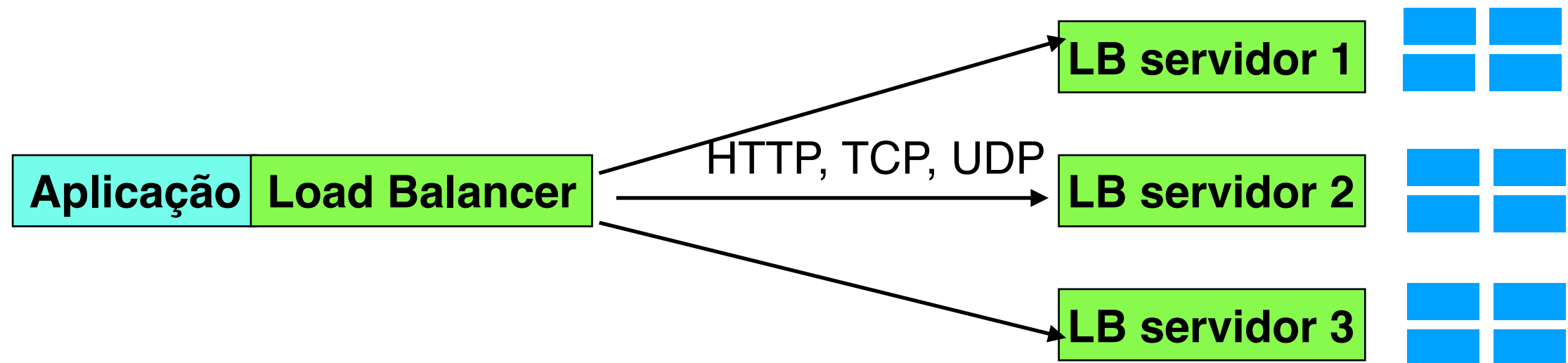


Micro serviços

Spring Cloud Ribbon

Balanceador de carga no lado do cliente

- O balanceador de carga no lado do cliente seleciona qual servidor chamar
- Com base em alguns critérios
- Parte do software cliente.
- O servidor ainda pode empregar seu próprio balanceador de carga



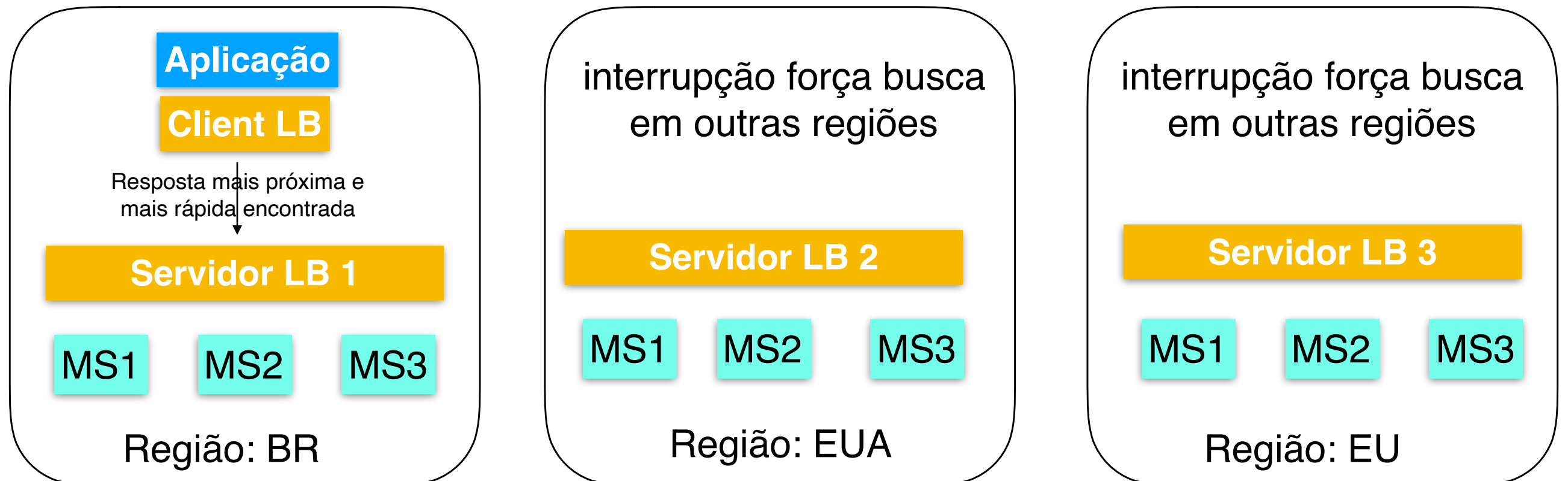
Micro serviços

Spring Cloud Ribbon

Por quê?

Nem todos os servidores são iguais

- Alguns podem estar indisponíveis (falhas)
- Alguns podem ser mais lentos que outros (desempenho)
- Alguns podem estar mais distantes do que outros (regiões)



Micro serviços

Spring Cloud Ribbon

Spring Cloud Netflix Ribbon - Outra parte da família Netflix OSS (Open Source Software)

- Balanceador de carga no lado do cliente
- Integra-se automaticamente com a descoberta de serviço (Eureka)
- Resiliência de falha integrada (Hystrix)
- Armazenamento em cache / em lote
- Vários protocolos (HTTP, TCP, UDP)
- O Spring Cloud fornece uma API Wrapper fácil para o uso da Ribbon

Micro serviços

Spring Cloud Ribbon

- Lista ou Pool de possíveis servidores
- Lista filtrada de servidores
- LB
- Ping

Micro serviços

Spring Cloud Ribbon

Lista de Servidores (sem eureka)

- Determina qual é a lista de servidores possíveis para um determinado serviço cliente
 - Estático - Preenchido por configuração (ex application.yml)
 - Dinâmico - Preenchido por Serviço Padrão do Spring Cloud Discovery (Eureka)
- Spring Cloud Default: Eureka quando presente no classpath

Exemplo de servidor “lista Estática”

"academico" e
“financeiro”
Exemplos de client
IDs

```
msacademico:  
  ribbon:  
    listOfServers: acad1.com, acad2.com  
msfinanceiro:  
  ribbon:  
    listOfServers: finan1.com, finan2.com
```

application.yml

Micro serviços

Spring Cloud Ribbon

Usando Ribbon com Spring Cloud

- Técnica de baixo nível
- Acesse o LoadBalancer diretamente

Exemplo de “client-id”

```
@GetMapping("/findAlunoById/{id}")  
public @ResponseBody AlunoDTO findAluno(@PathVariable int id) {  
    ServiceInstance instance = loadBalancer.choose("msacademico");  
    URI uriAluno = URI.create(String.format("http://%s:%s/aluno/findById?id=%s",  
instance.getHost(), instance.getPort(), id));  
    RestTemplate rest = new RestTemplate();  
    AlunoDTO a = rest.getForObject(uriAluno, AlunoDTO.class);  
    return a;  
}
```

Instância selecionada
pelo balanceador de
carga

Micro serviços

Spring Cloud Ribbon

Ping

- Utilizado para testar se o servidor está ativo ou inativo
- Padrão do Spring Cloud - delegar ao Eureka para determinar se o servidor está up ou down

Micro serviços

Spring Cloud Ribbon

Load Balancer

- O Load Balancer é o componente real que roteia as chamadas para os servidores na lista filtrada
- Várias estratégias disponíveis, mas geralmente utilizam o componente Rule para tomar as decisões reais
- O padrão da Spring Cloud: `ZoneAwareLoadBalancer`

Micro serviços

Spring Cloud Ribbon

Rule

- A Regra é o único módulo de inteligência que toma as decisões sobre fazer uma chamada ou não.
- Padrão da Spring Cloud: `ZoneAvoidanceRule`

Micro serviços

Spring Cloud Ribbon

Configurando o Spring Cloud Starter pom.xml

```
<properties>
  <java.version>1.8</java.version>
  <spring-cloud.version>Hoxton.RC2</spring-cloud.version>
</properties>

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


Micro serviços

Spring Cloud Ribbon

Anteriormente, descrevemos os padrões. E se você quiser alterá-los? Declare uma configuração separada com o bean de substituição.

Substitui a estratégia "Ping" ao chamar clientes "msacademico"

```
@Configuration
@RibbonClient(name = "msacademico", configuration =
AulaConfig.class)
public class MainConfig {

}
```

```
@Configuration
public class AulaConfig {
    @Bean
    public IPing ribbonPing(IClientConfig config) {
        return new PingUrl();
    }
}
```

Maiores detalhes: <http://netflix.github.io/ribbon/ribbon-core-javadoc/index.html>

Micro serviços

Spring Cloud Ribbon

Subindo os serviços fora do eclipse com maven.

```
mvn spring-boot:run -Dspring.profiles.active=artigos
mvn spring-boot:run -Dspring.profiles.active=verbos
mvn spring-boot:run -Dspring.profiles.active=sujeitos
mvn spring-boot:run -Dspring.profiles.active=substantivos
mvn spring-boot:run -Dspring.profiles.active=adjetivos

mvn spring-boot:run -Dspring.profiles.active=eureka-server
mvn spring-boot:run -Dspring.profiles.active=frase
```

Subindo os serviços fora do java -jar

```
java -jar nomeservico.jar -Dspring.profiles.active=frase
```

Micro serviços

Spring Cloud Ribbon

Usando o LB Ribbon - parte 1

pom.xml

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

FraseApplication.java

```
@SpringBootApplication
@EnableEurekaClient
public class FraseApplication {
    public static void main(String[] args) {
        SpringApplication.run(FraseApplication.class, args);
    }
    @Bean
    @LoadBalanced
    RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

Micro serviços

Spring Cloud Ribbon

Usando o LB Ribbon - parte 2

Controller.java

```
@RestController
public class Controller {
    @Autowired
    RestTemplate template;
    @GetMapping("/frase")
    public @ResponseBody String getFrase() {
        return getFracaoFrase("SUJEITO") + " " + getFracaoFrase("VERBO") +
        " " + getFracaoFrase("ARTIGO") + " "
            + getFracaoFrase("SUBSTANTIVO") + " " +
        getFracaoFrase("ADJETIVO") + ".";
    }
    public String getFracaoFrase(String microServico) {
        return template.getForObject("http://" + microServico,
String.class);
    }
}
```

Micro serviços

Spring Cloud Ribbon

Usando o LB Ribbon - parte 2

Pra rodar multiplas instancias de clientes adicionar no application.yml

```
instance:
  instanceId: ${spring.cloud.client.hostname}:${spring.application.name}:${spring.application.instance_id:${random.value}}

server:
# port: 8095
port: ${PORT:${SERVER_PORT:0}}
```

Micro serviços

Spring Cloud Ribbon

consideracoes...

Atualize a página do navegador de frases <http://localhost:9006/frase>. Quando tomar conhecimento do novo serviço, o loadbalancer distribuirá a carga entre os dois serviços e, na metade do tempo, sua frase terminará com um dos “adjetivos” definidos em cada serviço.

Pare um dos serviços e atualize a página do navegador de frases várias vezes. Você verá que ele falha na metade do tempo, pois uma das instâncias não está mais disponível. De fato, como o balanceador de carga padrão é baseado em um algoritmo round-robin, a falha ocorre a cada segunda vez que o serviço de substantivos é usado. Se você continuar atualizando por tempo suficiente, verá que as falhas eventualmente param quando o cliente da faixa de opções é atualizado com a lista de servidores revisada da Eureka.

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

O problema: falha em cascata

Ter um grande número de serviços como dependências pode levar a 'falhas em cascata'

Sem mitigar isso, os micro serviços são uma receita para um certo desastre!

O que acontece se este serviço falhar?

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

- Hystrix - Parte do Netflix OSS
- Leve, fácil de usar e fornecido pela Spring Cloud OSS.
- Detecta condições de falha e “abre” o circuito para impedir novas chamadas .
Hystrix Default - 20 falhas em 5 segundos
- Identifica "fallback" - o que fazer em caso de falha de dependência de serviço semelhante a um catch block, porem mais sofisticado
os fallbacks podem ser encadeado
- Automaticamente "se fecha" após o intervalo
- Hystrix Default -5 segundos.

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

- O Padrão do Hystrix - Circuit Breaker (Disjuntor)
- Considere um disjuntor doméstico "Observa" um circuito
- Quando ocorre uma falha (excesso de fluxo de corrente), ele "abre o circuito (desconecta o circuito) ;
- Depois que o problema é resolvido, você pode fechar manualmente o disjuntor girando o interruptor.
- Evita falhas em cascata, ou seja, sua casa queima.

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

pom.xml

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

Habilitar o Hystrix

```
@SpringBootApplication  
@EnableHystrix  
public class Application{  
    ...  
}
```

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

Exemplo @HystrixCommand envolver métodos num circuit breaker:

```
@Component
public class LojaIntegracao {

    @HystrixCommand (fallbackMethod = "lojaDefault")
    public Object getStores (Map<String, Object> parameters) {
        //alguma coisa que pode falhar
    }

    public Object lojaDefault(Map<String, Object> parameters) {
        return "alguma coisa util";
    }
}
```

Com base em falhas recentes, o Hystrix irá chamar um desses dois métodos

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

O comportamento do Custom Properties Failure/Recovery é altamente customizável

Pode também usar os seguintes parametros: **@Properties** e **@HystrixProperty**

```
@HystrixCommand ( fallbackMethod = "fraseDefault",  
    commandProperties = { Taxa de falha superior a 20% no período de 10 segundos, circuito aberto  
        @HystrixProperty name="circuitBreaker.errorThresholdPercentage", value="20"),  
        @HystrixProperty name="circuitBreaker.sleepWindowInMilliseconds", value="1000")  
    })  
    Após 1 segundo, tente fechar o circuito  
  
    public Object fraseDefault(...){  
        // seu código  
    }
```

<https://github.com/Netflix/Hystrix/tree/master/hystrix-contrib/hystrix-javanica#configuration>

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

O comando pode ser chamado de várias maneiras

- Synchronously - chamar executar e thread (padrao).
- Asynchronously - chamar numa thread separada (queue), retornando um futuro. Lide com o futuro quando quiser
Semelhante ao Spring @Async annotation
- Reactively Inscreva-se (subscribe), e obtenha um listener (Observable)

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

Ex: Asynchronous Command Execution

Ter método que retorna o Futuro.

Envolver o resultado em AsyncResult

```
@HystrixCommand(...)
public Future<Adjetivo> getAdjetivo (Map<String, Object> parameters) {
    return new AsyncResult<Store> () {
        @Override
        public Adjetivo invoke () {
            //fazer algo que pode falhar
        };
    }
}
```

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

Ex: **Reactive Command Execution**

Ter método de retorna um Observable

Envolver o resultado em um Observable

```
@HystrixCommand(...)
public Observable<Adjetivo> getAdjetivo (Map<String, Object> parameters)
{
    return new Observable Result<Store> () {

        @Override
        public Adjetivo invoke () {

        }

    }
}
```

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

- Modificar o sistema para usar o Feign em todas chamadas
- Construir uma interface FraseService (interface) e uma classe FraseServiceImpl que agrupa as chamadas para os clientes Feign. Apenas para organizar e melhorar a legibilidade do código.
- Abra o POM. Adicione outra dependência para o spring-cloud-starter-netflix-hystrix.
- Edite a classe de configuração principal do aplicativo e @EnableHystrix.
- Modificar o FraseServiceImpl para usar o Hystrix. Decidimos que não é necessário ter um adjetivo em nossa frase se o serviço de adjetivo estiver falhando; portanto, modifique o serviço getAdjetivo para executar dentro de um Comando Hystrix.

Micro serviços

Spring Cloud Hystrix - Circuit Breaker

- Estabeleça um método de fallback que retorne um novo adjetivo("hystrix fallback").
- O aplicativo deve funcionar da mesma forma que antes, embora a chamada "Adjetivo" agora esteja passando pelo Hystrix.
- Para testar, pare o serviço Adjetivo. Atualize <http://localhost:9006/frase>. A frase deve aparecer sem um adjetivo.
- Reinicie o serviço adjetivo. Quando o registro Eureka estiver completo e o circuito fechado, o servidor de sentenças exibirá novamente os adjetivos.
- Abrir a URL : <http://localhost:9006/hystrix/>
- informar no Hystrix Dashboard: localhost:9006/actuator/hystrix.stream
- clicar em Monitor Stream

Micro serviços

Spring Cloud Zuul - API Gateway

- Porque usar um API Gateway?
- Spring Cloud Netflix Zuul
- Cache

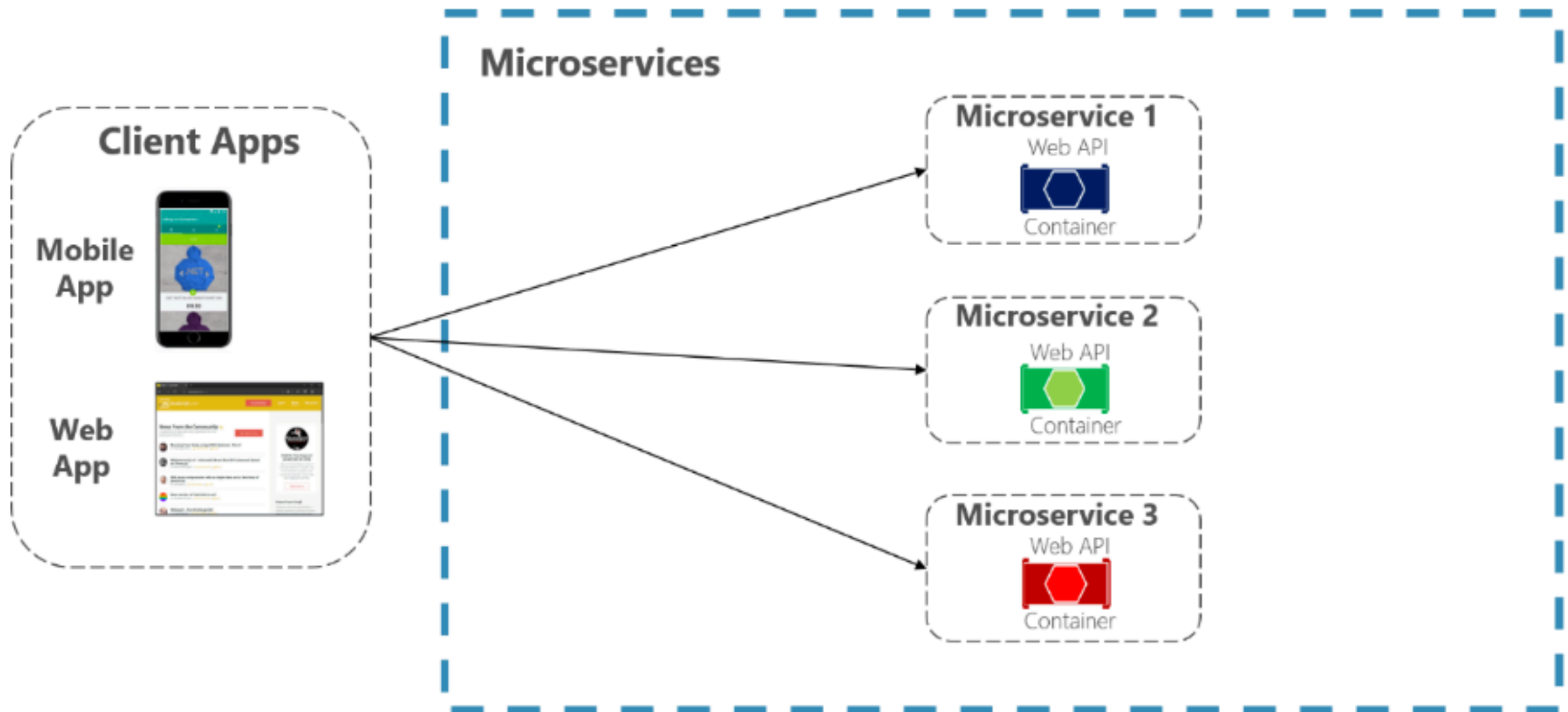
Micro serviços

Spring Cloud Zuul - API Gateway

- Acessando de uma internet publica
- API interna exposta
- Segurança
- CORS (cross origin request sharing)
- Múltiplas chamadas

Micro serviços

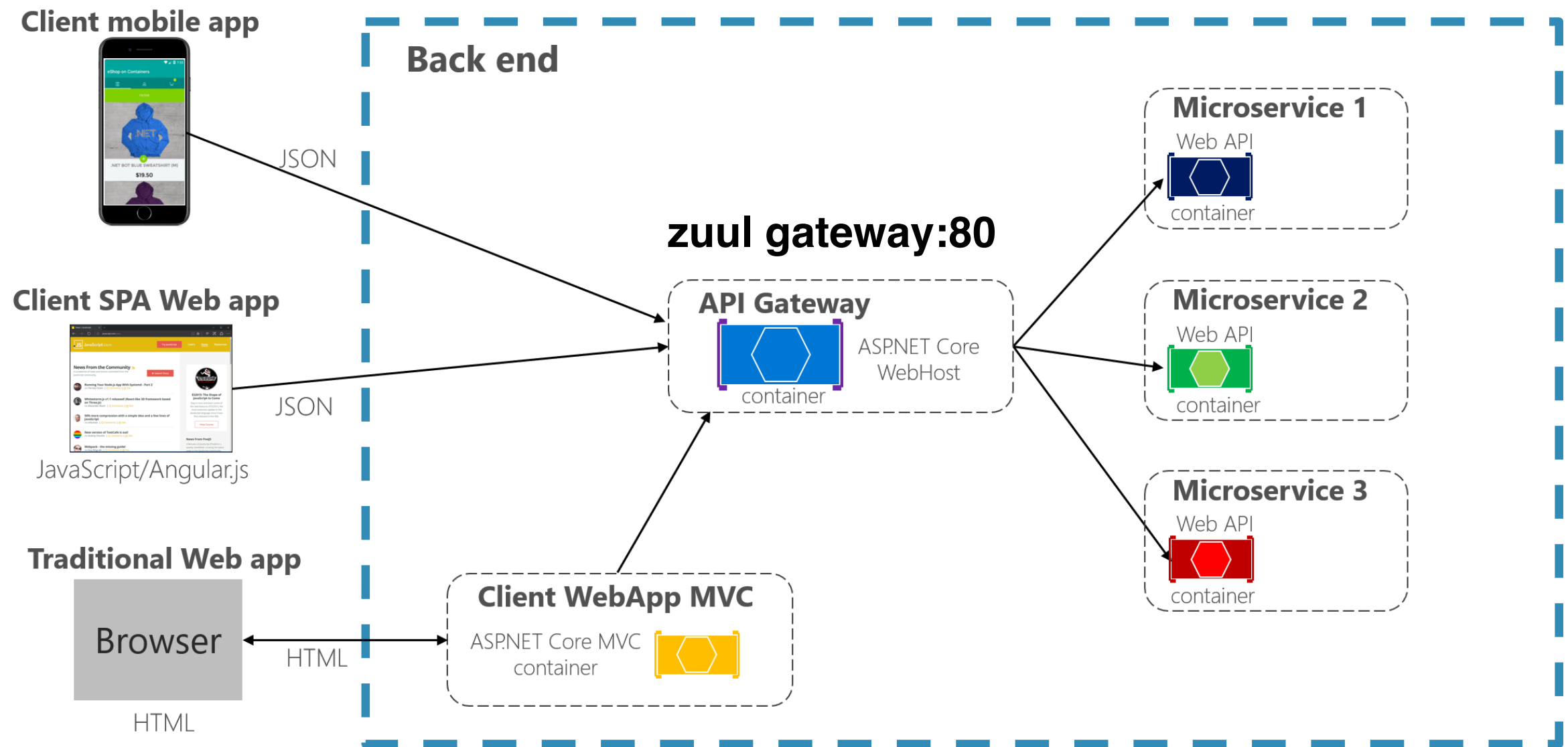
Spring Cloud Zuul - API Gateway



Fonte: <https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>

Micro serviços

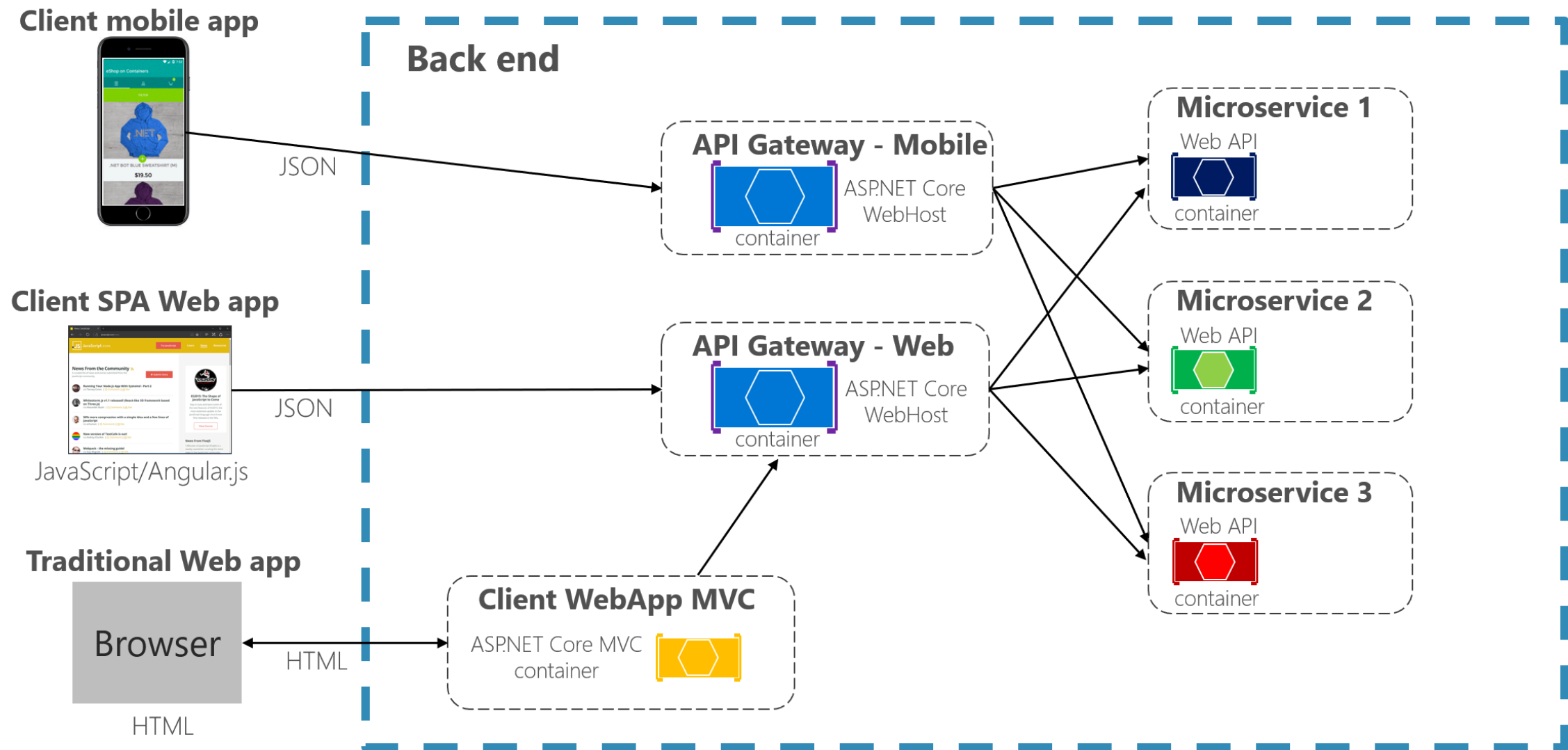
Spring Cloud Zuul - API Gateway



Fonte: <https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>

Micro serviços

Spring Cloud Zuul - API Gateway



Fonte: <https://docs.microsoft.com/pt-br/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern>

Micro serviços

Spring Cloud Zuul - API Gateway

- Zuul Features serviços podem ser excluídos
zuul.ignored-services:
- adicionar um prefixo: zuul.prefix: /api

```
zuul:  
  prefix: /api  
  ignored-services: endereco  
  routes:  
    financeiro:  
      path: msfinanceiro/**  
    academico:  
      path: msacademico/**
```

Micro serviços

Spring Cloud Zuul - API Gateway

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

```
spring:  
  application:  
    name: web-gateway  
  
server:  
  port: 8888  
  
eureka:  
  client:  
    serviceUrl:  
      defaultZone: http://localhost:8090/eureka/  
  
zuul:  
  prefix: /api  
  ignored-services: endereco  
  routes:  
    msfinanceiro:  
      path: msfinanceiro/**  
    msacademico:  
      path: msacademico/**
```


Micro serviços

Spring Cloud Zuul - API Gateway

```
@SpringBootApplication
@EnableZuulProxy
public class WebGatewayApplication {

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

    @Bean
    public ShallowEtagHeaderFilter shallowEtagHeaderFilter() {
        return new ShallowEtagHeaderFilter();
    }
}
```

Micro serviços

Spring Cloud Zuul - API Gateway

O Cliente vai chamar:

<http://localhost:8888/msfinanceiro/parcela/parcelasInadimplentes>
<http://localhost:8888/msacademico/aluno>
<http://localhost:8888/msacademico/professor>
<http://localhost/msintegracaofinanceira/integracaoFinanceira/parcelasInadimplentes>

O Zuul vai chamar:

<http://localhost:8081/parcela/parcelasInadimplentes>
<http://localhost:8080/aluno>
<http://localhost:8080/aluno>
<http://localhost:57855/integracaoFinanceira/parcelasInadimplentes>

Micro serviços

Spring Cloud Zuul - API Gateway

