

Curso

Aplicações JAVA com SPRING BOOT



Prof. Msc. Antonio B. C. Sampaio Jr
engenheiro de software & professor

@abctreinamentos
@amazoncodebr

www.abctreinamentos.com.br
www.amazoncode.com.br



CONTEÚDO PROGRAMÁTICO



- UNIDADE 1 – INTRODUÇÃO
- UNIDADE 2 – FUNDAMENTOS DO SPRING BOOT
- UNIDADE 3 – PERSISTÊNCIA DE DADOS NO SPRING BOOT
- UNIDADE 4 – PROJETO WEB NO SPRING BOOT
- UNIDADE 5 – PROJETO REST API NO SPRING BOOT
- UNIDADE 6 – PROJETO REST API NO SPRING BOOT COM REACTJS

CONTEÚDO PROGRAMÁTICO



- UNIDADE 7 – PROJETO REST API NO SPRING BOOT
COM THYMELEAF
- UNIDADE 8 – PROJETO REST API NO SPRING BOOT
COM MICROSERVIÇOS
- UNIDADE 9 – PROJETO FINAL
- EXTRAS

PROJETOS DO CURSO



- 1º Projeto Spring Boot – Impressão de Mensagens
- 2º Projeto Spring Boot – Impressão de Mensagens na WEB
- 3º Projeto Spring Boot – Aplicação Servidor Público
- 4º Projeto Spring Boot – Aplicação Servidor Público na WEB
- 5º Projeto Spring Boot – Aplicação Servidor Público no SGBD MYSQL
- 6º Projeto Spring Boot – Aplicação Servidor Público no MONGO DB

PROJETOS DO CURSO



- 7º Projeto Spring Boot – Aplicação Servidor Público WEB
- 8º Projeto Spring Boot – Aplicação Servidor Público REST API e MySQL
- 9º Projeto Spring Boot – Aplicação Servidor Público REST API com REACT
- 10º Projeto Spring Boot – Aplicação Servidor Público/Curso REST API – Monolítico
- 11º Projeto Spring Boot – Aplicação Servidor Público REST API - Microserviços

UNIDADE 8

PROJETO REST API NO SPRING BOOT COM MICROSSERVIÇOS

ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

ARQUITETURA DE MICROSERVIÇOS (MSA)

ARQUITETURA MSA NO SPRING BOOT

PRINCIPAIS ANOTAÇÕES

SPRING BOOT STARTERS

PROJETO PRÁTICO

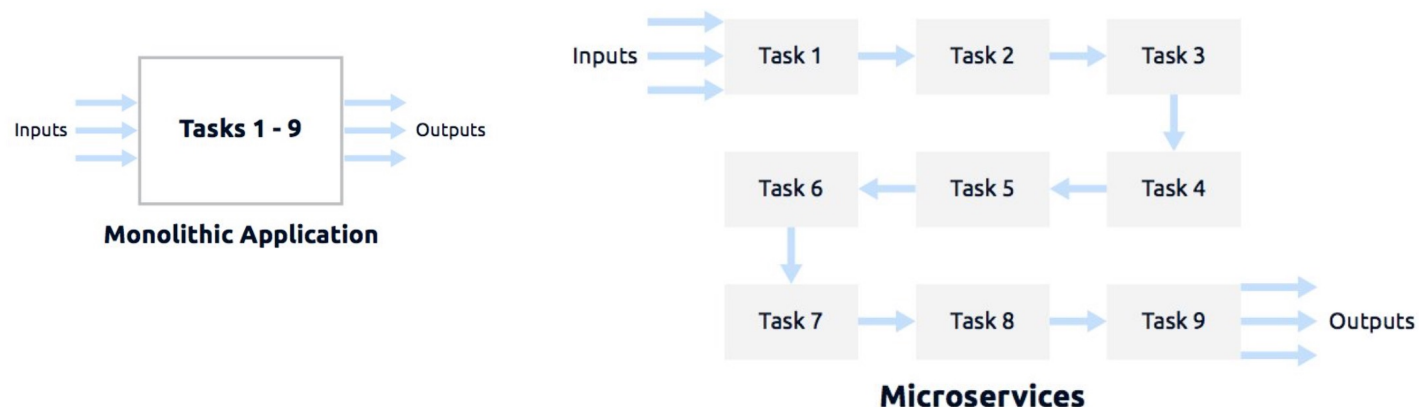
11º Projeto Spring Boot – Aplicação Servidor
Público/Curso REST API com MICROSERVIÇOS

ARQUITETURA
ORIENTADA A
SERVIÇOS

ARQUITETURA ORIENTADA A SERVIÇOS

- **Definição**

- Arquitetura Orientada a Serviços (*SOA - Service-Oriented Architecture*) é uma abordagem de design de software que permite a criação de sistemas distribuídos e interoperáveis por meio da composição de serviços independentes. Nessa arquitetura, os serviços são unidades lógicas de funcionalidade que são projetadas para serem independentes, autônomas e reutilizáveis.



ARQUITETURA ORIENTADA A SERVIÇOS

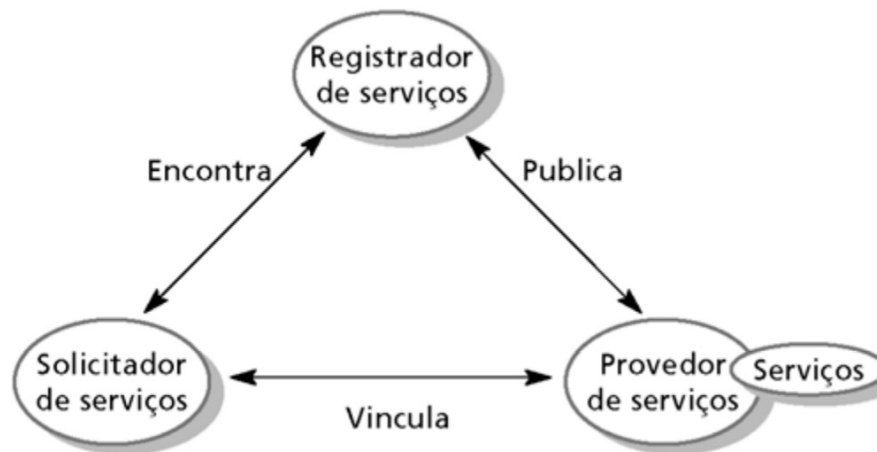
- **Principais Características SOA**

- **Serviços** - Os serviços são os principais blocos de construção da arquitetura. Eles encapsulam funcionalidades específicas do sistema e podem ser invocados por outros componentes por meio de interfaces bem definidas.
- **Composição** - A composição de serviços permite a criação de novas funcionalidades combinando serviços existentes. Isso permite a reutilização de serviços e a construção de sistemas complexos a partir de componentes menores e independentes.
- **Interoperabilidade** - A arquitetura SOA promove a interoperabilidade entre sistemas heterogêneos. Os serviços são projetados para serem independentes de plataformas e tecnologias específicas, permitindo que eles se comuniquem e cooperem entre si.

ARQUITETURA ORIENTADA A SERVIÇOS

- **Principais Características SOA**

- **Descoberta e Registro** - A arquitetura SOA geralmente envolve um mecanismo de descoberta e registro de serviços, onde os serviços são registrados e disponibilizados para outros componentes que desejam consumi-los. Isso permite a localização dinâmica de serviços em tempo de execução.



ARQUITETURA ORIENTADA A SERVIÇOS

- **Principais Características SOA**

- **Granularidade** - Os serviços podem ter diferentes granularidades, desde serviços granulares que executam tarefas específicas até serviços mais abrangentes que oferecem funcionalidades mais complexas. A escolha da granularidade depende dos requisitos e das necessidades do sistema.

ARQUITETURA ORIENTADA A SERVIÇOS

- **Principal Objetivo**

- A arquitetura SOA tem como objetivo principal promover a reutilização, modularidade, flexibilidade e interoperabilidade de sistemas. Ela permite que organizações desenvolvam sistemas distribuídos mais flexíveis, escaláveis e adaptáveis, integrando componentes de software existentes e facilitando a evolução e a manutenção dos sistemas ao longo do tempo.

ARQUITETURA ORIENTADA A SERVIÇOS

- Qual a relação com os Microserviços?
 - A relação entre SOA e microserviços é estreita, pois os microserviços são uma abordagem específica de implementação da arquitetura orientada a serviços.
 - Tanto SOA quanto microserviços têm como objetivo principal criar sistemas distribuídos e modularizados, mas existem diferenças significativas em sua abordagem e implementação.

ARQUITETURA ORIENTADA A SERVIÇOS

- **Quais as Principais Diferenças SOA e Microserviços?**
 - **Granularidade** - SOA geralmente envolve serviços de granularidade mais grossa, onde cada serviço pode oferecer funcionalidades abrangentes e complexas. Por outro lado, os microserviços são caracterizados por serviços de granularidade fina, onde cada microserviço é responsável por uma tarefa específica e bem definida.
 - **Composição** - Em SOA, os serviços são compostos para criar funcionalidades complexas. Essa composição pode ser feita por meio de orquestração ou coreografia de serviços. Já nos microserviços, a composição é realizada por meio da comunicação entre microserviços independentes, onde cada um tem sua própria lógica de negócio.

ARQUITETURA ORIENTADA A SERVIÇOS

- **Quais as Principais Diferenças SOA e Microserviços?**
 - **Escopo e Autonomia** - SOA geralmente envolve sistemas monolíticos divididos em serviços, onde os serviços ainda estão fortemente acoplados e compartilham recursos e infraestrutura. Nos microserviços, cada microserviço é autônomo e independente, possuindo sua própria base de código, infraestrutura e banco de dados. Os microserviços podem ser implantados, dimensionados e atualizados independentemente.
 - **Tecnologias e Protocolos** - SOA é mais flexível em termos de tecnologias e protocolos utilizados para comunicação entre serviços, permitindo a interoperabilidade entre sistemas heterogêneos. Nos microserviços, é comum o uso de protocolos HTTP/REST para comunicação síncrona entre microserviços.

ARQUITETURA ORIENTADA A SERVIÇOS

- **Quais as Principais Diferenças SOA e Microserviços?**
 - **Evolução e Escalabilidade** - Os microserviços são projetados para facilitar a evolução e a escalabilidade dos sistemas, permitindo que novos microserviços sejam adicionados ou modificados sem afetar os outros. SOA também permite a evolução, mas pode ser mais desafiador lidar com a escalabilidade e a flexibilidade devido ao acoplamento entre os serviços.

ARQUITETURA DE MICROSSERVIÇOS

ARQUITETURA DE MICROSERVIÇOS

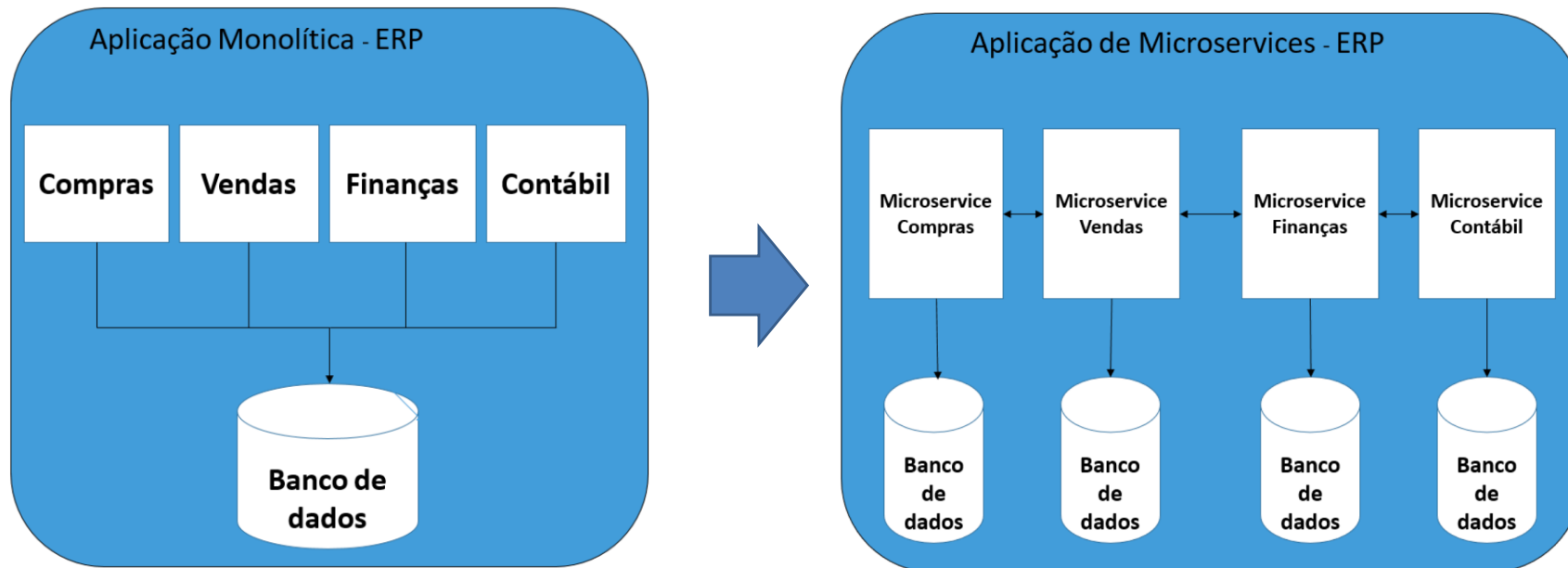
- **Definição**

- A MSA (*Microservices Architecture*) é uma abordagem arquitetural que enfatiza a construção de sistemas de software como um conjunto de microserviços independentes e autônomos, oferecendo benefícios como escalabilidade, agilidade no desenvolvimento e tolerância a falhas.



ARQUITETURA DE MICROSERVIÇOS

- Monolítico => Microsserviços



ARQUITETURA DE MICROSERVIÇOS

- Por que é tão Popular?



**Desire for faster
changes**



**Need for greater
availability**



**Looking for fine-
grained scaling**



**Compatible with
a DevOps
mindset**

ARQUITETURA DE MICROSERVIÇOS

- **Caso de Sucesso**



Traditional development model

Hundreds of engineers

Microservices architecture

- Small teams
- End-to-end development
- Hundreds of microservices

ARQUITETURA DE MICROSERVIÇOS

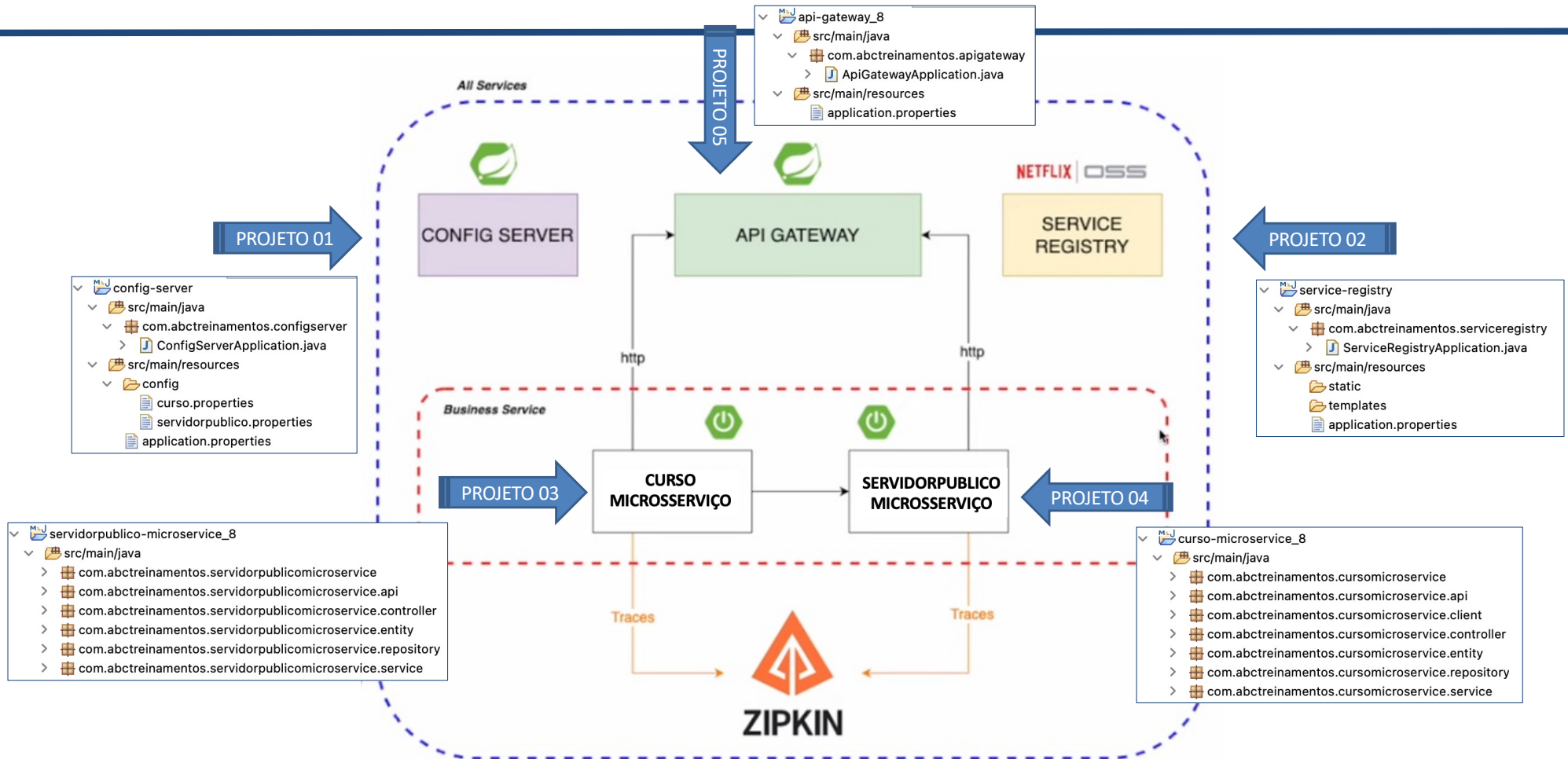
- Caso de **InSucesso**



<https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90>

ARQUITETURA MSA
NO SPRING BOOT

ARQUITETURA MSA NO SPRING BOOT



PROJETO 01 - CONFIG SERVER

- **Definição**

- O **ConfigServer** é um componente do Spring Cloud que faz parte da infraestrutura de gerenciamento de configuração do Spring Boot. Ele é responsável por fornecer um repositório centralizado para armazenar as configurações de uma aplicação distribuída.

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

PROJETO 01 - CONFIG SERVER

- **Benefícios**

- Existem vários benefícios ao utilizar o **ConfigServer** do Spring Boot:
 - **Centralização das configurações:** Todas as configurações são armazenadas em um local centralizado, facilitando o gerenciamento e a atualização das configurações em uma aplicação distribuída.
 - **Separação das configurações do código:** As configurações não precisam estar embutidas no código do aplicativo, permitindo que elas sejam modificadas sem a necessidade de recompilar ou reiniciar o aplicativo.
 - **Suporte a múltiplos ambientes:** O ConfigServer permite que você defina perfis de configuração para diferentes ambientes, como desenvolvimento, teste e produção. Isso facilita a personalização das configurações para cada ambiente.

PROJETO 01 - CONFIG SERVER

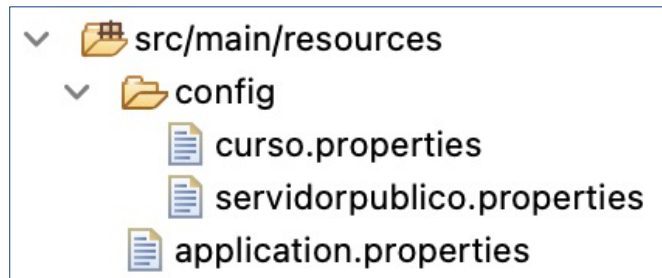
- **Benefícios**

- Existem vários benefícios ao utilizar o **ConfigServer** do Spring Boot:
 - **Recarregamento dinâmico das configurações:** Com o ConfigServer, você pode alterar as configurações em tempo de execução sem reiniciar o aplicativo. Isso permite que você faça ajustes ou atualizações nas configurações sem interromper o serviço.
 - **Segurança das configurações:** O ConfigServer do Spring Boot suporta recursos de segurança, como autenticação e autorização, para garantir que apenas as pessoas autorizadas tenham acesso às configurações sensíveis.

PROJETO 01 - CONFIG SERVER

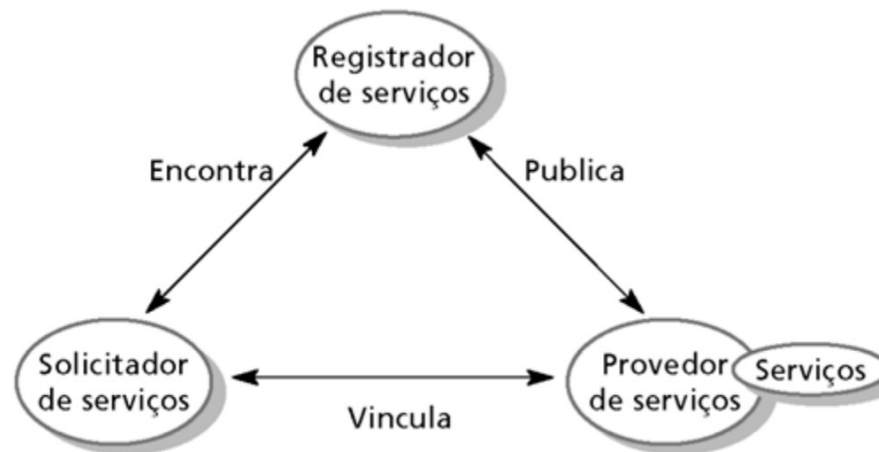
- **Benefícios**

- O **ConfigServer** é responsável por fornecer um repositório centralizado de configurações para os serviços distribuídos. Ele não está diretamente envolvido na interação entre solicitadores e provedores de serviços. Sua função é disponibilizar e gerenciar as configurações dos serviços de forma centralizada, permitindo que os serviços acessem suas configurações conforme necessário.



PROJETO 01 - CONFIG SERVER

- Resumo
- O ConfigServer atua como um componente de gerenciamento de configurações na arquitetura SOA, enquanto o solicitador de serviços, registrador de serviços e provedor de serviços são elementos relacionados à interação entre os componentes da arquitetura SOA.



PROJETO 02 – SERVICE REGISTRY

- **Definição**

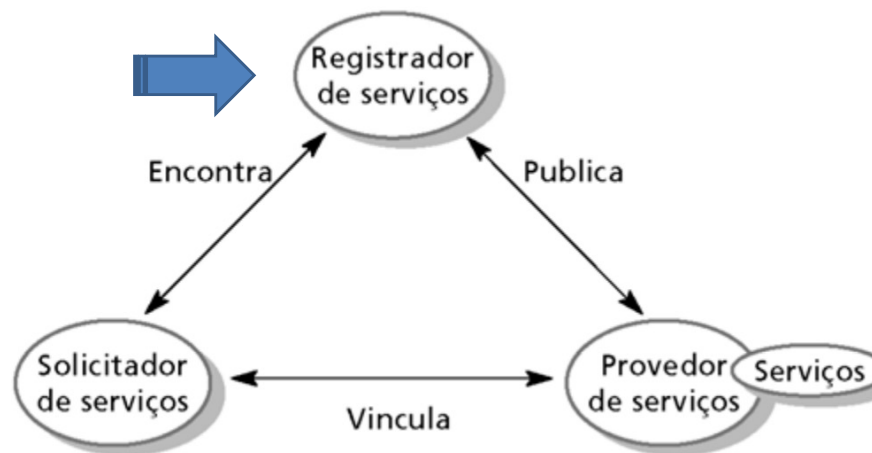
- O Service Registry (**Registrador de Serviços**) é um componente fornecido pelo Spring Cloud que desempenha um papel importante na arquitetura de microsserviços. Ele é responsável por manter um registro centralizado de todos os serviços disponíveis em um ambiente distribuído.

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

PROJETO 02 – SERVICE REGISTRY

- **Definição**

- O **Service Registry** permite que os microserviços se registrem e se descubram de maneira dinâmica, facilitando a comunicação entre eles. Ele atua como um catálogo de serviços, onde os microserviços podem registrar informações como nome, endereço, porta, protocolo, versão e outras metainformações relevantes.




PROJETO 02 – SERVICE REGISTRY


- **Resumo**

- Ao usar o Service Registry, os microsserviços podem se tornar independentes de localização e configuração estática. Em vez de precisarem conhecer explicitamente o endereço de cada serviço com o qual desejam se comunicar, eles podem consultar o Service Registry para obter as informações necessárias. Isso permite que os microsserviços sejam escaláveis, resilientes e flexíveis, pois podem se adaptar a mudanças na topologia e nas configurações dos serviços.
- O Spring Cloud oferece uma implementação de Service Registry chamada Eureka, que é uma solução de registro e descoberta de serviços. O Eureka fornece um servidor de registro onde os microsserviços podem se registrar e uma interface de descoberta que permite que os microsserviços localizem e se comuniquem entre si de forma dinâmica.


PROJETO 02 – SERVICE REGISTRY


-  **spring Eureka**
 - O **Eureka** é um serviço de registro e descoberta de serviços desenvolvido pela **Netflix** e amplamente utilizado em arquiteturas baseadas em microsserviços. Ele faz parte do ecossistema do Spring Cloud, que é uma família de projetos do Spring que facilitam o desenvolvimento de aplicativos distribuídos.
 - O **Eureka** fornece um servidor de registro onde os microsserviços podem se registrar e um cliente que permite que os microsserviços descubram e se comuniquem uns com os outros de forma dinâmica. Ele oferece recursos de balanceamento de carga e failover, permitindo que os microsserviços sejam escaláveis e resilientes.

PROJETO 02 – SERVICE REGISTRY

-  **spring Eureka**
 - Os microsserviços que desejam se registrar no Eureka fornecem informações, como nome, endereço, porta e metainformações adicionais, para que possam ser descobertos por outros microsserviços. O Eureka mantém um registro atualizado dos serviços disponíveis e fornece uma interface RESTful para consultas de descoberta.
 - Ao utilizar o Eureka, os microsserviços podem se tornar independentes de configurações estáticas e conhecer apenas a localização do servidor Eureka. Isso permite que os serviços sejam adicionados, removidos ou alterados sem afetar diretamente outros microsserviços, pois eles podem consultar o Eureka para descobrir os serviços disponíveis.

PROJETO 02 – SERVICE REGISTRY

-  **spring Eureka**
 - Em resumo, o Eureka é um serviço de registro e descoberta de serviços usado em arquiteturas de microsserviços para facilitar a comunicação e a descoberta dinâmica de serviços. Ele desempenha um papel fundamental na criação de sistemas distribuídos escaláveis e resilientes.



[HOME](#) [LAST 1000 SINCE STARTUP](#)

System Status

Environment	test	Current time	2023-05-27T11:36:28 -0300
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	2

DS Replicas

localhost

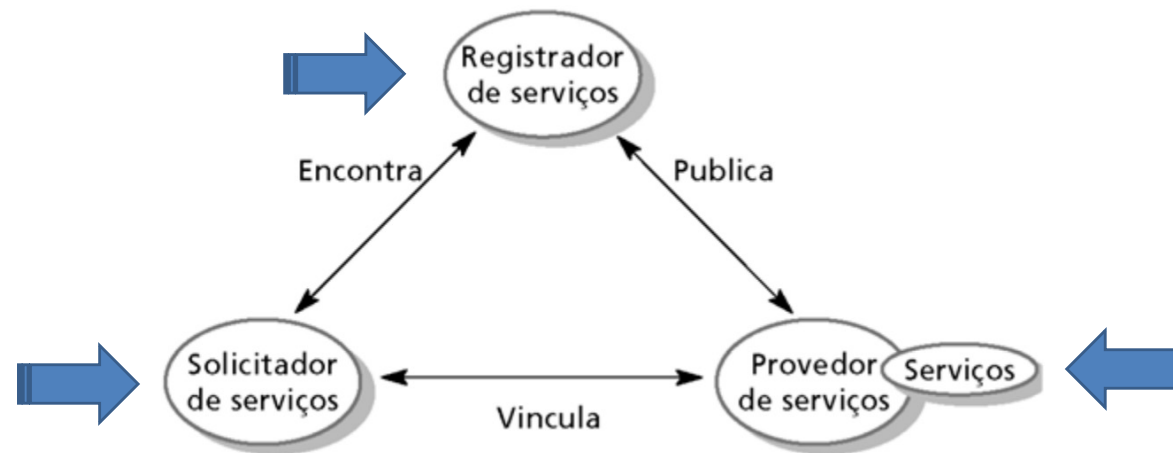
Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CURSO	n/a (1)	(1)	UP (1) - 192.168.0.201:curso:8082
SERVIDORPUBLICO	n/a (1)	(1)	UP (1) - 192.168.0.201:servidorpublico:8081

PROJETOS 03 E 04 – MICROSERVIÇOS

- **Definição**

- Os microserviços podem desempenhar os três papéis abaixo, dependendo do contexto em que são utilizados.



PROJETOS 03 E 04 – MICROSERVIÇOS

- **Solicitador de Serviços (*Service Consumer*)**
 - Os microsserviços podem atuar como solicitadores de serviços ao consumir funcionalidades fornecidas por outros serviços. Nesse papel, eles enviam solicitações para serviços específicos, buscando resultados ou executando operações em nome do usuário ou de outros componentes. Os microsserviços que consomem serviços são responsáveis por invocar os serviços corretos e tratar as respostas recebidas.

```
@FeignClient(name = "servidorpublico",url="http://192.168.0.201:8090")
public interface ServidorPublicoServiceClient {

    @GetMapping("/servidorpublico/listarServidores/{idcurso}")
    public List<ServidorPublico> findAllByCursoId(@PathVariable("idcurso") Long idcurso);
}
```

PROJETOS 03 E 04 – MICROSERVIÇOS

- **Registrador de Serviços (*Service Registry*)**
 - O registrador de serviços é um componente que mantém um registro centralizado de todos os serviços disponíveis na arquitetura SOA. Geralmente, o registrador de serviços é utilizado para registrar e atualizar informações sobre os serviços, como nome, localização, protocolo, versão e outras metainformações relevantes. Os microserviços, ao serem iniciados, podem se registrar no registrador de serviços para tornar suas funcionalidades disponíveis para outros componentes.

PROJETOS 03 E 04 – MICROSERVIÇOS

- **Provedor de Serviços (*Service Provider*)**
 - Os microsserviços também podem atuar como provedores de serviços ao implementar e disponibilizar funcionalidades específicas. Nesse papel, eles expõem suas funcionalidades por meio de interfaces ou endpoints para que possam ser consumidos pelos solicitadores de serviços. Os microsserviços provedores de serviços são responsáveis por implementar a lógica de negócio e fornecer as respostas adequadas às solicitações recebidas.

```
@RestController
@RequestMapping("servidorpublico")
public class ServidorPublicoController implements ServidorpublicoAPIRest{
    //Recebimento da chamada via Microsserviço
    @GetMapping("/listarServidores/{idcurso}")
    public List<ServidorPublico> findAllServidores(@PathVariable("idcurso") Long idcurso) {
        List<ServidorPublico> servidores = servidorService.findAllByCursoId(idcurso);
        return servidores;
    }
}
```

PROJETO 05 - GATEWAY

- **Definição**

- Um **Gateway** é um componente responsável por receber todas as solicitações de um cliente (como um navegador da web ou um aplicativo móvel) e encaminhá-las para os serviços apropriados. É uma peça fundamental na arquitetura de microsserviços, onde diferentes serviços são distribuídos e executados independentemente.

```
@SpringBootApplication
@EnableDiscoveryClient
public class ApiGatewayApplication {

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


PROJETO 05 - GATEWAY

- **Definição**

- Um **Gateway** atua como um ponto de entrada único para o sistema, permitindo o roteamento e a filtragem de solicitações. Ele pode ser usado para lidar com várias tarefas, como autenticação, autorização, balanceamento de carga, cache, monitoramento, registro, entre outros.
- O conceito de Gateway no Spring Boot refere-se a um componente intermediário que permite o roteamento e a filtragem de solicitações de entrada, fornecendo uma interface única para acessar os serviços distribuídos de um sistema.

PROJETO 05 - GATEWAY

- **Definição**

- No Spring Boot, um Gateway é implementado usando o projeto **Spring Cloud Gateway**, que fornece uma solução flexível e escalável para criar gateways em arquiteturas de microsserviços. O Spring Cloud Gateway permite **definir rotas**, filtros e manipular as solicitações e respostas de acordo com as necessidades do sistema.

```
spring.cloud.gateway.routes[0].id=curso
spring.cloud.gateway.routes[0].uri=lb://curso
spring.cloud.gateway.routes[0].predicates[0]=Path=/curso/**
spring.cloud.gateway.routes[1].id=servidorpublico
spring.cloud.gateway.routes[1].uri=lb://servidorpublico
spring.cloud.gateway.routes[1].predicates[0]=Path=/servidorpublico/**
```

PRINCIPAIS ANOTAÇÕES



@EnableDiscoveryClient

- **Definição**

- Esta anotação é fornecida pelo Spring Cloud para habilitar a descoberta de serviços em um ambiente distribuído.
- Ao adicionar a anotação **@EnableDiscoveryClient** a uma classe de configuração no Spring Boot, será ativada a capacidade de descoberta de serviços do Spring Cloud.
- Essa anotação é comumente usada quando se usa o Service Registry do Spring Cloud, como o Eureka Server, para registrar e descobrir seus microsserviços.
- Quando a anotação **@EnableDiscoveryClient** é adicionada, o aplicativo Spring Boot automaticamente se conecta ao Service Registry configurado (como o Eureka Server) e registra-se como um serviço disponível.



@EnableConfigServer

- **Definição**

- É uma anotação fornecida pelo Spring Cloud para habilitar o Config Server em uma aplicação Spring Boot.
- Ao adicionar a anotação **@EnableConfigServer** a uma classe de configuração no Spring Boot, será ativada a funcionalidade de servidor de configuração do Spring Cloud.
- O Config Server é responsável por fornecer configurações centralizadas para os microserviços em uma arquitetura distribuída.



@EnableEurekaServer

- **Definição**

- É uma anotação fornecida pelo Spring Cloud Netflix para habilitar o servidor Eureka em um aplicativo Spring Boot.
- O Eureka é um serviço de registro e descoberta de serviços que permite que os microsserviços se registrem e localizem uns aos outros em uma arquitetura de microsserviços. Ao adicionar a anotação **@EnableEurekaServer** a uma classe de configuração Spring Boot, será ativada a funcionalidade de servidor Eureka.
- Ao usar o **@EnableEurekaServer**, o aplicativo Spring Boot se torna um servidor Eureka, permitindo que outros microsserviços se registrem e sejam descobertos por meio dele. Isso facilita a comunicação e a descoberta de serviços em uma arquitetura distribuída.



- **Definição**

- Esta anotação é utilizada para habilitar a funcionalidade de clientes Feign. O **Feign** é uma biblioteca do Spring Cloud que simplifica a comunicação entre microsserviços usando uma abordagem de cliente HTTP declarativa.
- As interfaces **Feign** definem as chamadas de serviço entre microsserviços. Essas interfaces seguem uma abordagem declarativa que definem os métodos, os endpoints de destino e os parâmetros da chamada. A tecnologia Feign se encarrega de realizar a chamada HTTP e fornecer uma implementação concreta.



@EnableFeignClients

- Exemplo

```
@FeignClient(name = "servidorpublico",url="http://192.168.0.201:8090")
public interface ServidorPublicoServiceClient {

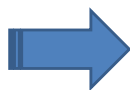
    @GetMapping("/servidorpublico/listarServidores/{idcurso}")
    public List<ServidorPublico> findAllByCursoId(@PathVariable("idcurso") Long idcurso);
}
```


SPRING BOOT STARTERS

MSA NO SPRING BOOT

- **Spring Starters Necessários**

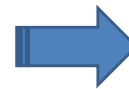
- Abaixo os **Spring Starters** necessários para a criação dos serviços de (1) Config-Server e (2) Service-Registry.



Config Server

SPRING CLOUD CONFIG

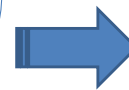
Central management for configuration via Git, SVN, or HashiCorp Vault.



Spring Web

WEB

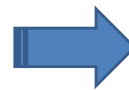
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.



Eureka Discovery Client

SPRING CLOUD DISCOVERY

A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.



Eureka Server

SPRING CLOUD DISCOVERY

spring-cloud-netflix Eureka Server.

(1) CONFIG-SERVER

(2) SERVICE-REGISTRY

MSA NO SPRING BOOT

- **Spring Starter Config Server**


- O Spring Starter Config Server permite criar um servidor de configuração centralizado para os microsserviços. Faz parte do projeto Spring Cloud Config e permite a centralização das configurações dos microsserviços em um único local.
- Ao usar o Spring Starter Config Server, os microsserviços podem acessar as configurações armazenadas no servidor de configuração central por meio de uma API REST ou usando a **integração do Spring Cloud Config Client**.

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



MSA NO SPRING BOOT

- **Eureka Discovery Client (Spring Cloud Netflix)**
 - O Eureka Discovery Client faz parte do projeto Spring Cloud Netflix e é um componente usado para integração de serviços com o Eureka Server, que é um servidor de registro e descoberta de serviços. O Eureka Discovery Client permite que os serviços se registrem no servidor Eureka e consultem as informações de registro de outros serviços.



HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2023-05-27T11:36:28 -0300
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	2

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CURSO	n/a (1)	(1)	UP (1) - 192.168.0.201:curso:8082
SERVIDORPUBLICO	n/a (1)	(1)	UP (1) - 192.168.0.201:servidorpublico:8081



MSA NO SPRING BOOT

- **Spring Starter Eureka Discovery Client**

- Ao adicionar a dependência do Spring Starter Eureka Discovery Client ao projeto Spring Boot, facilmente a aplicação é registrada no servidor Eureka e pode aproveitar recursos como balanceamento de carga, tolerância a falhas e descoberta de serviços.

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



MSA NO SPRING BOOT

- **Spring Starter Eureka Server**

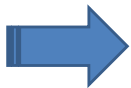
- Ao adicionar a dependência do Spring Starter Eureka Server ao projeto Spring Boot, é simplificada a criação de um servidor de registro e descoberta de serviços em uma arquitetura de microsserviços, proporcionando recursos para facilitar a comunicação dinâmica entre os serviços e garantir a alta disponibilidade e escalabilidade do sistema.

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

MSA NO SPRING BOOT

- **Spring Starters Necessários**

- Abaixo os **Spring Starters** necessários para a criação de microserviços (3) **ServidorPublico** e (4) **Curso** no Spring Boot com acesso ao banco de dados MySQL:

**Zipkin** **OBSERVABILITY**

Enable and expose span and trace IDs to Zipkin.

**Eureka Discovery Client** **SPRING CLOUD DISCOVERY**

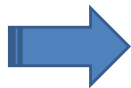
A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

Spring Web **WEB**

Build web, including RESTful, applications using Spring MVC.
Uses Apache Tomcat as the default embedded container.

Spring Data JPA **SQL**

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

**Config Client** **SPRING CLOUD CONFIG**

Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

MySQL Driver **SQL**

MySQL JDBC driver.

**OpenFeign** **SPRING CLOUD ROUTING**

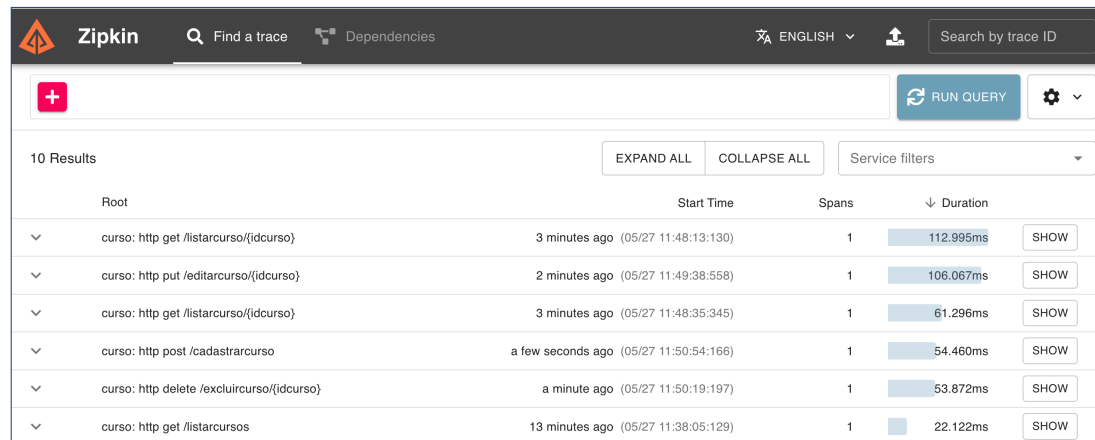
Declarative REST Client. OpenFeign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.



MSA NO SPRING BOOT

- **Zipkin**

- O Zipkin é uma ferramenta de rastreamento distribuído que ajuda a coletar e visualizar informações sobre as solicitações que fluem através de vários microsserviços. Ele permite rastrear a execução de uma solicitação à medida que ela atravessa diferentes microsserviços, fornecendo informações valiosas sobre o tempo de processamento, dependências e possíveis gargalos.



Root	Start Time	Spans	Duration	
curso: http get /listarcurso/{idcurso}	3 minutes ago (05/27 11:48:13:130)	1	112.995ms	SHOW
curso: http put /editarcurso/{idcurso}	2 minutes ago (05/27 11:49:38:558)	1	106.067ms	SHOW
curso: http get /listarcurso/{idcurso}	3 minutes ago (05/27 11:48:35:345)	1	61.296ms	SHOW
curso: http post /cadastrarcursos	a few seconds ago (05/27 11:50:54:166)	1	54.460ms	SHOW
curso: http delete /excluircurso/{idcurso}	a minute ago (05/27 11:50:19:197)	1	53.872ms	SHOW
curso: http get /listarcursos	13 minutes ago (05/27 11:38:05:129)	1	22.122ms	SHOW

MSA NO SPRING BOOT

- **Spring Starter Zipkin**

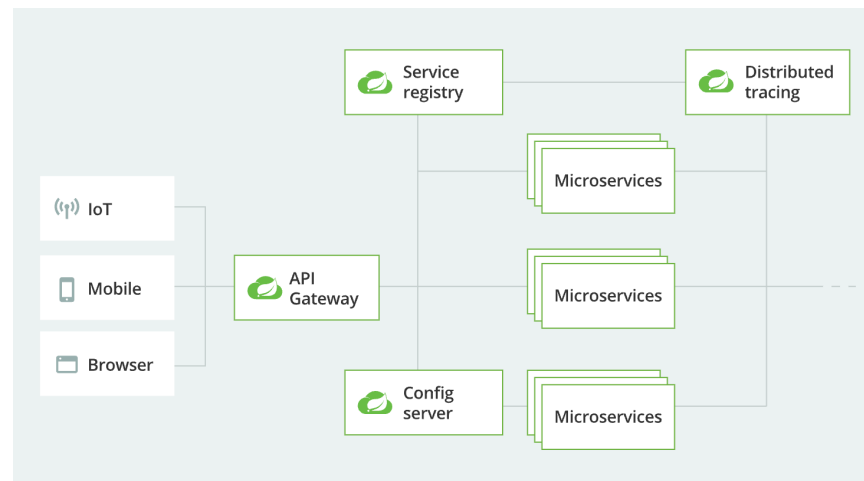
- O Spring Starter Zipkin simplifica a integração do Zipkin em aplicativos Spring Boot. Ele fornece dependências e configurações pré-definidas que permitem aos desenvolvedores adicionar facilmente recursos de rastreamento distribuído aos seus aplicativos. Através do uso de anotações e configurações específicas, é possível registrar e propagar identificadores de rastreamento entre os microsserviços, permitindo que o rastreamento seja acompanhado de ponta a ponta.

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>
<dependency>
  <groupId>io.zipkin.reporter2</groupId>
  <artifactId>zipkin-reporter-brave</artifactId>
</dependency>
```

MSA NO SPRING BOOT

- **Config Client (Spring Cloud)**

- O Config Client é um componente que faz parte do ecossistema do Spring Cloud e é responsável por fornecer recursos para a integração de configurações externas em um aplicativo Spring Boot. Ele permite que os aplicativos obtenham de forma centralizada as configurações necessárias para sua execução, em vez de terem as configurações embutidas em seu código.



MSA NO SPRING BOOT

- **Spring Starter Config Client**

- Com o Config Client, é possível recuperar as configurações do servidor de configuração remoto, que é geralmente fornecido pelo Spring Cloud Config Server. O Config Client fornece uma maneira fácil e flexível de acessar essas configurações usando a abordagem "convenção sobre configuração" do Spring Boot.

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

MSA NO SPRING BOOT

- **Spring Starter OpenFeign**

- Ao adicionar a dependência do Spring Starter OpenFeign ao projeto Spring Boot, facilmente a aplicação pode criar clientes HTTP para se comunicar com serviços remotos.

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

MSA NO SPRING BOOT

- **Spring Starters Necessários**

- Abaixo os **Spring Starters** necessários para a criação do serviço de (5) Api-Gateway.



Gateway SPRING CLOUD ROUTING

Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.

Spring Boot Actuator OPS

Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.



Config Client SPRING CLOUD CONFIG

Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

Zipkin OBSERVABILITY

Enable and expose span and trace IDs to Zipkin.



Eureka Discovery Client SPRING CLOUD DISCOVERY

A REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

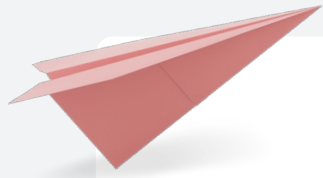
MSA NO SPRING BOOT

- **Spring Starter Gateway**

- O Spring Starter Gateway é uma dependência do Spring Boot que facilita a criação de um gateway de API em uma arquitetura de microsserviços, fornecendo recursos avançados de roteamento, filtragem e manipulação de tráfego para melhorar a segurança, desempenho e resiliência do seu sistema.

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

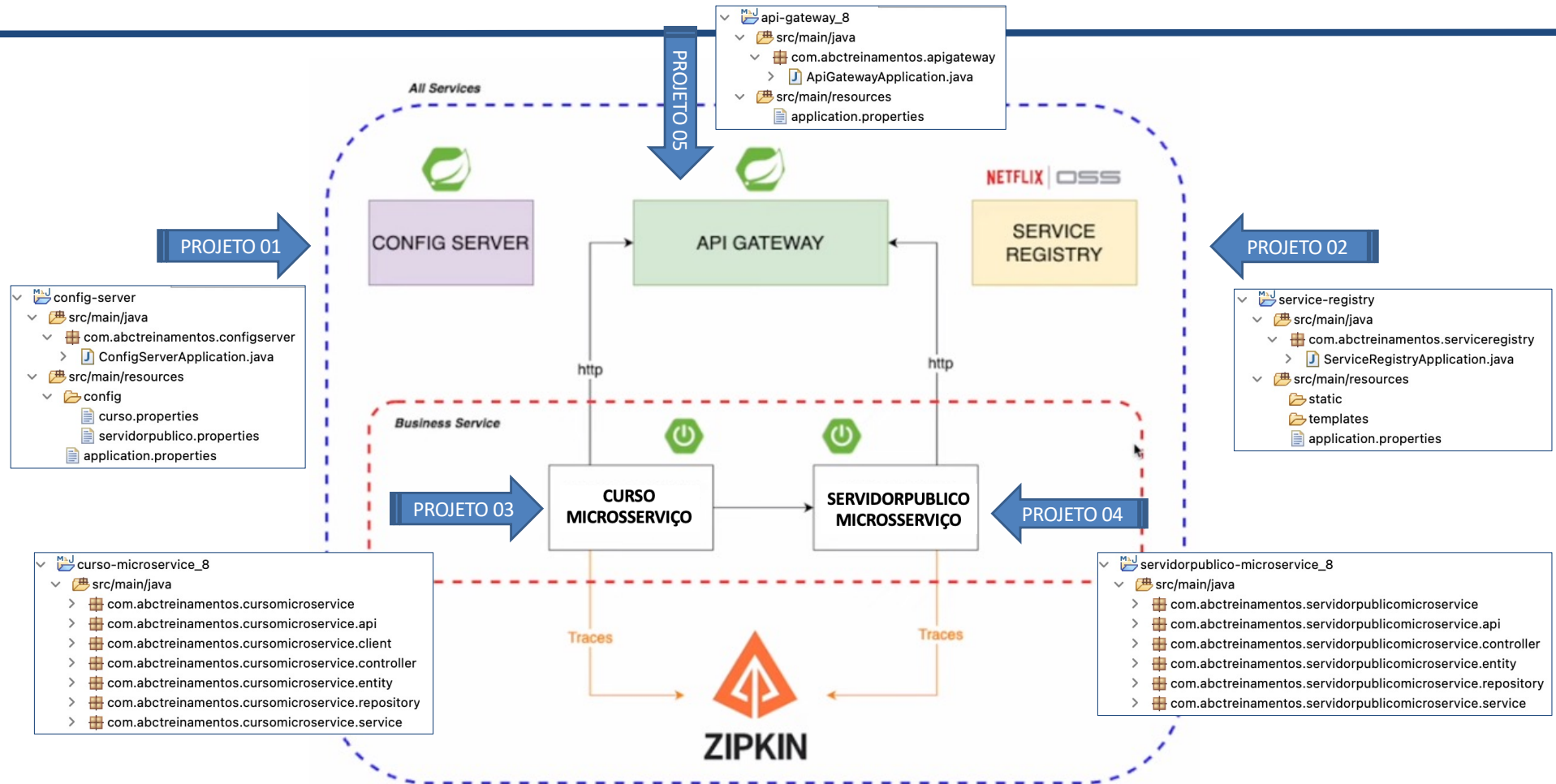
PROJETO PRÁTICO

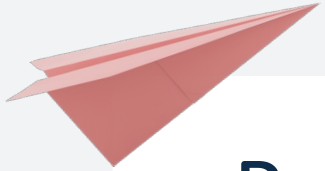


11º Projeto Spring Boot – Aplicação Servidor Público/Curso REST API com MICROSERVIÇOS



ARQUITETURA MSA NO SPRING BOOT





Passos:

- Criar o Projeto 01 (Servidor de Configuração)
- Criar o Projeto 02 (Servidor de Registro)
- Criar o Projeto 03 (Microserviço Servidor Público)
- Criar o Projeto 04 (Microserviço Curso)
- Criar o Projeto 05 (Gateway)



ARQUITETURA DE MICROSERVIÇOS

