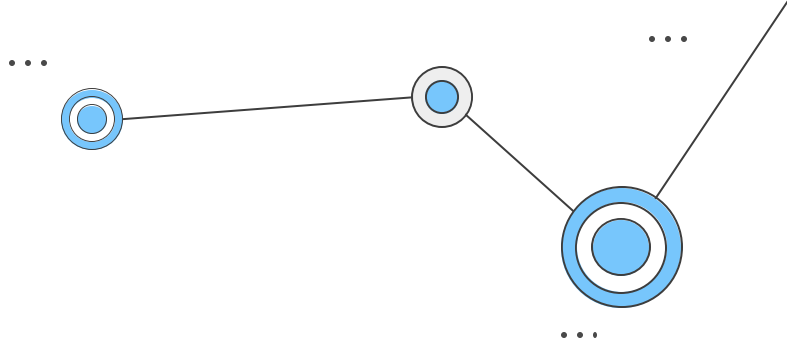
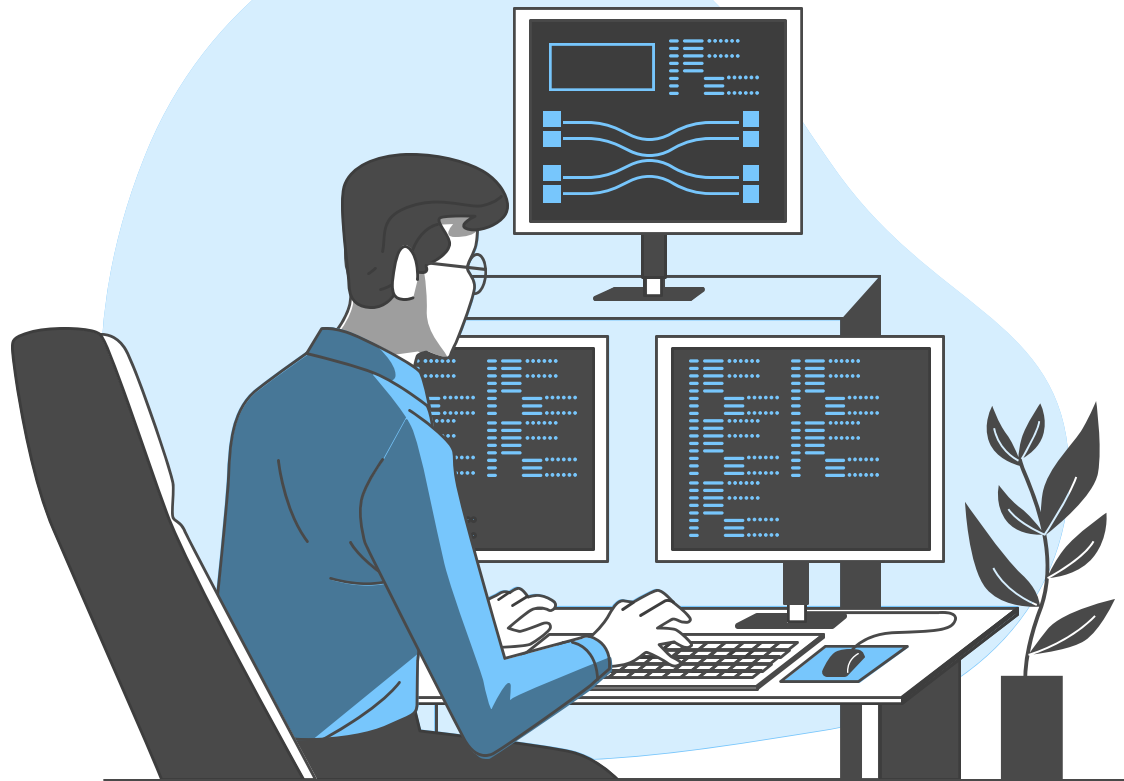




PUC Minas



Laboratório de Desenvolvimento de Software

Prof. Dr. João Paulo Aramuni

Sumário

- Arquitetura de microsserviços
 - Arquitetura de microsserviços
 - Modelos de arquitetura REST
 - Princípios e boas práticas de serviços REST

Sumário

- Arquitetura de microsserviços
 - **Arquitetura de microsserviços**
 - Modelos de arquitetura REST
 - Princípios e boas práticas de serviços REST

Arquitetura de microserviços

- **Por que** aprender a arquitetura de **Microserviços** para o desenvolvimento?
 - Embora os conceitos fundamentais por trás dos microserviços não sejam novos, a aplicação dessa arquitetura é atual, e sua adoção tem sido motivada em parte pelos desafios de:
 - Escalabilidade
 - Falta de eficiência
 - Velocidade lenta de desenvolvimento
 - Dificuldade em adotar novas tecnologias

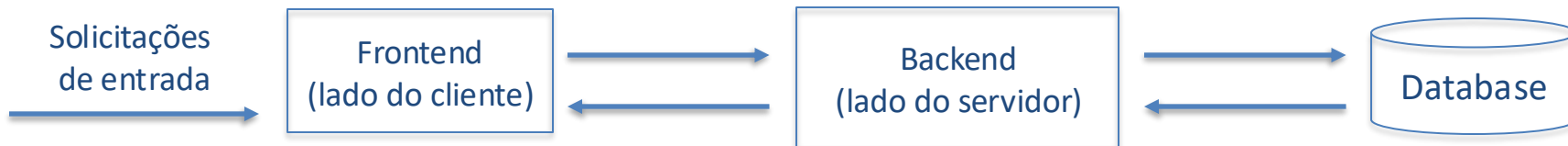


Arquitetura de microsserviços

- Quando sistemas complexos de software são implantados como uma grande aplicação maciça (monolítica), a dificuldade para se desenvolver aumenta e problemas como **retrabalho** começam a assolar a equipe.
- Com a arquitetura de microsserviços, uma aplicação pode ser facilmente **escalada**, a **produtividade** e a velocidade do desenvolvedor aumentam dramaticamente e tecnologias antigas podem facilmente ser trocadas pelas mais recentes.

Arquitetura de microsserviços

- Quase todas as aplicações de software escritas atualmente podem ser divididas em três elementos distintos: lado frontend, lado backend e algum tipo de armazenamento de dados. A chamada **arquitetura de três camadas**:

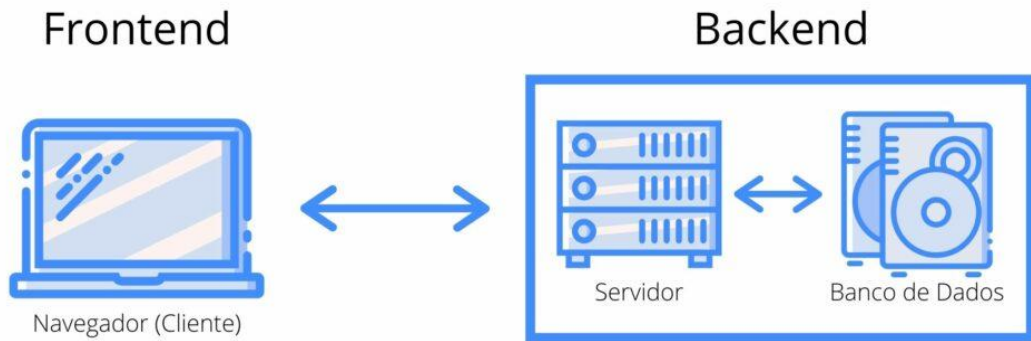


Arquitetura de microsserviços

- A maioria das aplicações coloca o **frontend** e o **backend** em um código-base (ou repositório), onde é armazenado e executado todo código do lado do cliente e do backend como um arquivo executável, com um database separado.
- Outras aplicações **separam** todo código frontend do código backend e os armazenam como executáveis lógicos separados, acompanhados por um database externo.

Arquitetura de microsserviços

- Independentemente da forma como essas três camadas são divididas ou combinadas, a própria aplicação é considerada a **soma desses três elementos distintos**.



Arquitetura de microsserviços

- Nos primeiros estágios, quando uma empresa é jovem, suas aplicações são simples e o número de desenvolvedores que contribuem para o código-base é pequeno.
- À medida que a empresa cresce, mais desenvolvedores são contratados, novos recursos são acrescentados à aplicação, com **três consequências significativas**:



Arquitetura de microsserviços

- I) Primeiro vem um aumento da carga de trabalho (**workload**) operacional, associado à execução e manutenção da aplicação. Isso leva à contratação de engenheiros operacionais (DevOps), que assumem a maioria das tarefas operacionais, como aquelas relacionadas ao hardware e monitoramento dos serviços.
- II) A segunda, é um simples resultado matemático: acrescentar novos recursos à sua aplicação aumenta o número de linhas de código e a **complexidade** da aplicação.

Arquitetura de microsserviços

- III) A terceira é o necessário dimensionamento da aplicação. O aumento do tráfego resulta em significativas demandas de escalabilidade e desempenho sobre a aplicação, exigindo que mais servidores hospedem a aplicação.
- Muitos sistemas acabam não sobrevivendo à essas consequências, necessitando serem migrados de arquitetura ou até mesmo descontinuados (mortos). Aqueles que ainda respiram por aparelhos, são chamados de **sistemas legados**.

Arquitetura de microsserviços

Arquitetura **Monolítica**

- À medida que a empresa cresce e o número de engenheiros ultrapassa a ordem de grandeza de centenas, tudo começa a ficar mais complicado.
- A complexidade da aplicação cresce juntamente à quantidade de linhas de código adicionada e centenas (quando não milhares) de testes precisam ser escritos para garantir a integridade do código.

Arquitetura de microsserviços

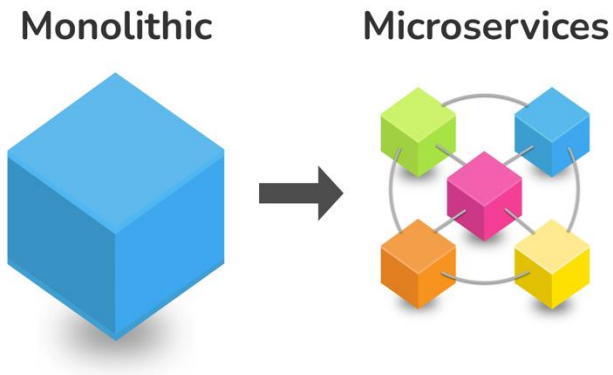
Arquitetura **Monolítica**

- O desenvolvimento e a implantação tornam-se um **pesadelo**, o teste passa a ser um fardo e a implantação das correções mais cruciais é adiada, aumentando rapidamente a **defasagem técnica**.
- É exatamente nesse contexto que a empresa começa a se preocupar em contratar um Arquiteto de Sistemas, ou um Engenheiro de Confiabilidade (SRE) para sanar essas dores.

Arquitetura de microsserviços

Arquitetura **Monolítica**

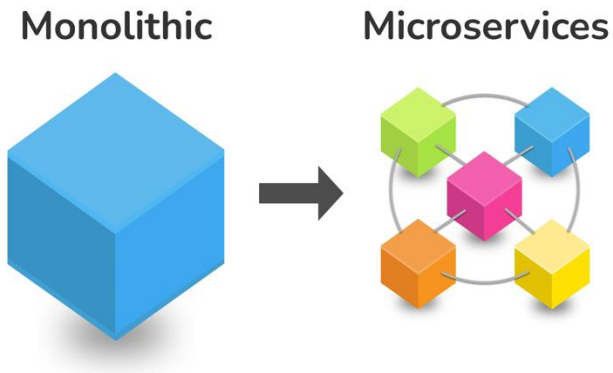
- Aplicações cujo ciclo de vida se encaixa nesse padrão, são carinhosamente conhecidas pela comunidade de software como **MONÓLITOS**.



Arquitetura de microsserviços

Arquitetura **Monolítica**

- A razão pela qual a maioria dos monólitos é suscetível a esses problemas é a natureza de um monólito ser diretamente oposta à **escalabilidade**.
- Escalabilidade requer **simultaneidade** e **segmentação**.
 - Processar em paralelo e;
 - Dividir para conquistar.



Arquitetura de microsserviços

Arquitetura **Monolítica**

- Uma forma simples, porém cara, de se adiar os problemas de escalabilidade causados por uma arquitetura monolítica, é **a compra de mais hardware com mais potência.**
- **Para pensar:** O que é mais barato/caro?
 - I) Contratar um arquiteto de **software** por R\$ 20,000.00 mensais ou;
 - II) Contratar mais recursos de **hardware** da Amazon por R\$ 20,000.00 mensais.



Arquitetura de microsserviços



AWS4GB

\$56/mo

Add to Cart

- ☒ 2 vCPU
- ☒ 4 GB RAM
- ☒ 80 GB SSD
- ☒ 4000 GB Bandwidth
- ☒ SSH and SFTP Access
- ☒ Git Integration
- ☒ NodeJS
- ☒ Cron Job Manager
- ☒ Multiple PHP Versions
- ☒ Free Website Migration

[View All Features](#)

AWS8GB

\$96/mo

Add to Cart

- ☒ 2 vCPU
- ☒ 8 GB RAM
- ☒ 160 GB SSD
- ☒ 5000 GB Bandwidth
- ☒ SSH and SFTP Access
- ☒ Git Integration
- ☒ NodeJS
- ☒ Cron Job Manager
- ☒ Multiple PHP Versions
- ☒ Free Website Migration

[View All Features](#)

AWS16GB

\$176/mo

Add to Cart

- ☒ 4 vCPU
- ☒ 16 GB RAM
- ☒ 320 GB SSD
- ☒ 6000 GB Bandwidth
- ☒ SSH and SFTP Access
- ☒ Git Integration
- ☒ NodeJS
- ☒ Cron Job Manager
- ☒ Multiple PHP Versions
- ☒ Free Website Migration

[View All Features](#)

AWS32GB

\$336/mo

Add to Cart

- ☒ 8 vCPU
- ☒ 32 GB RAM
- ☒ 640 GB SSD
- ☒ 7000 GB Bandwidth
- ☒ SSH and SFTP Access
- ☒ Git Integration
- ☒ NodeJS
- ☒ Cron Job Manager
- ☒ Multiple PHP Versions
- ☒ Free Website Migration

[View All Features](#)

Arquitetura de microserviços

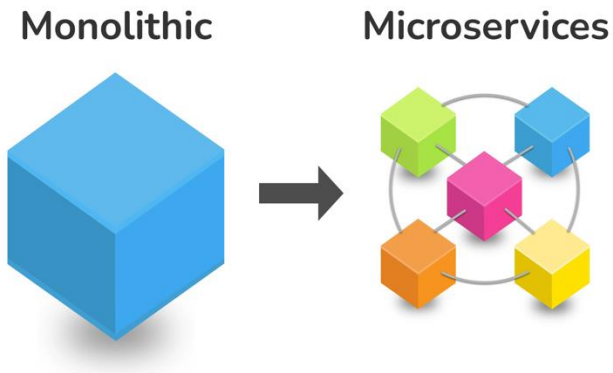
Arquitetura **Microserviços**

- Empresas que executam aplicações em milhares, até mesmo centenas de milhares, de servidores e cujas aplicações tornaram-se monólitos, passaram a enfrentar problemas de **escalabilidade**.
- Em outras palavras, por exemplo, a aplicação funciona bem para 100.000 usuários acessando simultaneamente mas não funciona direito para 1 milhão.

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Estes desafios de escalabilidade foram superados abandonando-se a arquitetura de aplicação monolítica em favor de microsserviços.
- O conceito básico de um microsserviço é simples:
 - Ele é uma pequena aplicação que **executa uma única tarefa** e a faz com eficiência.

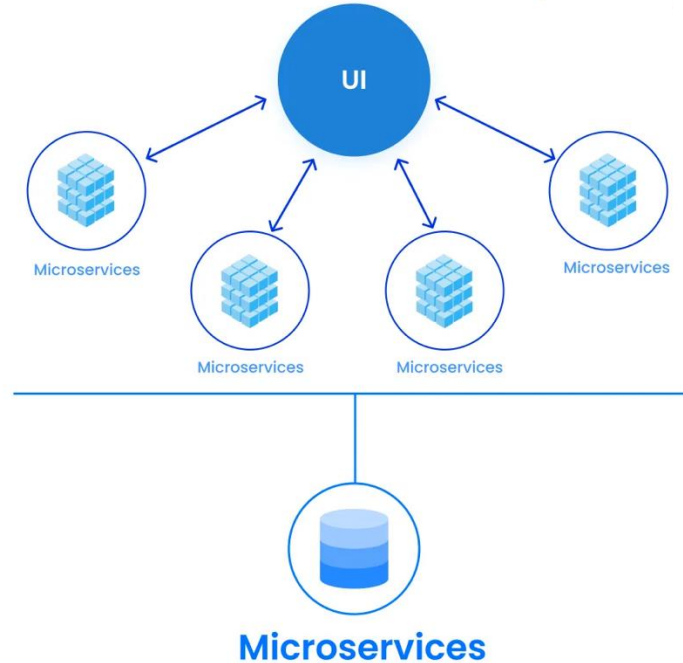
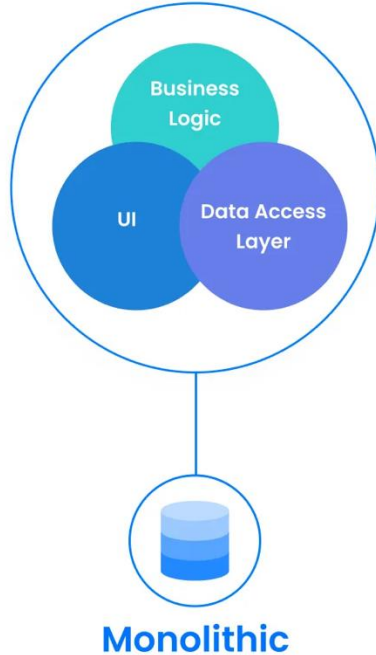


Arquitetura de microsserviços

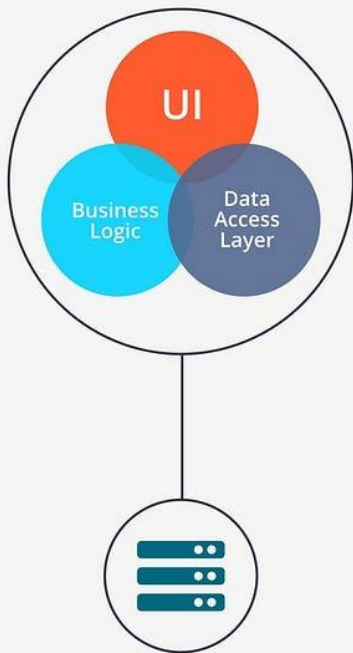
Arquitetura **Microsserviços**

- Um microsserviço é um pequeno componente que pode ser facilmente substituído, e é desenvolvido e implantado de forma **independente**.
- Um microsserviço não é uma ilha e não sobrevive sozinho: é parte de um sistema maior sendo executado e funcionando junto a outros microsserviços para realizar tarefas que normalmente seriam tratadas por uma grande aplicação autônoma.

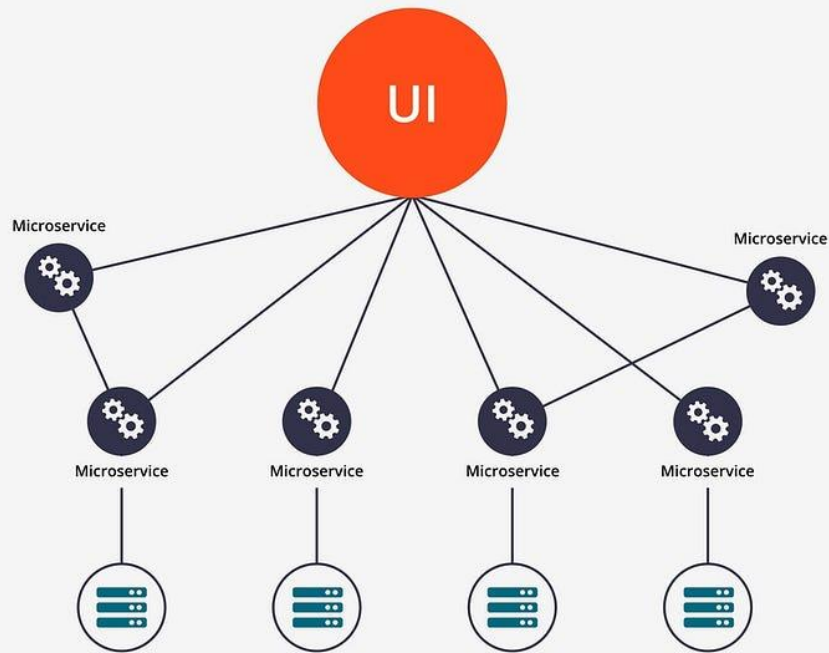
Sistema Monolítico vs Sistema com Microserviços



Sistema Monolítico vs Sistema com Microserviços



Monolithic Architecture



Microservice Architecture

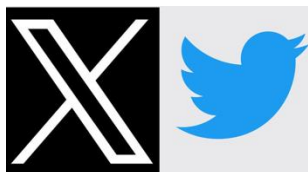
Arquitetura de microsserviços

Arquitetura **Microsserviços**

- O objetivo da arquitetura de microsserviços é construir um conjunto de pequenas aplicações, cada uma responsável por executar uma função (ao contrário da forma tradicional de construir **uma aplicação que faz tudo**), e permitir que cada microsserviço seja autônomo, independente e autossuficiente.
- Leitura sugerida:
 - <https://martinfowler.com/microservices/>

Arquitetura de microsserviços

- Nos últimos anos, o setor de tecnologia tem testemunhado uma rápida mudança na arquitetura aplicada em **sistemas distribuídos**, o que tem levado os gigantes da indústria como **Netflix**, **Twitter**, **Amazon**, **eBay** e **Uber** a abandonar a construção de aplicações monolíticas e adotar a arquitetura de microsserviços.



Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Competição por recursos de **hardware**:
 - Em uma aplicação monolítica, todos os recursos de hardware são direcionados para todas as funções existentes no código.
 - Com isso, caso uma parte do sistema esteja sobrecarregada, todo o sistema pode ser comprometido e **parar de funcionar**.
 - Como por exemplo durante uma promoção relâmpago na Black Friday ou durante a venda de ingressos para a libertadores 2024 na Arena MRV.

Arquitetura de microsserviços

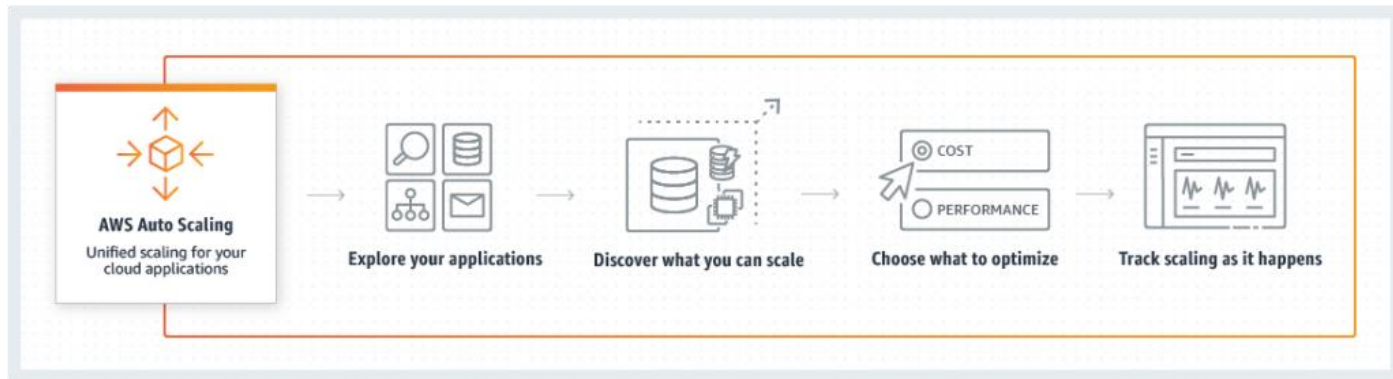
Arquitetura **Microsserviços**

- Competição por recursos de **hardware**:
 - Em uma aplicação com microsserviços, os recursos de hardware podem ser divididos e designados para aquelas funções que exigem maior capacidade de processamento, memória RAM ou recursos de rede.
 - Para pensar: Qual dessas funções gastaria mais recursos de hardware?
 - `cadastrarUsuario()`;
 - `logar()`;
 - `comprarIngresso()`;

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Competição por recursos de **hardware**:
 - A AWS (Amazon Web Services) já possui maneiras eficientes para auto escalar a aplicação durante picos de acesso.



Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Competição por recursos de **hardware**:
 - Hoje, a AWS gera mais receita para a Amazon do que todo o seu site de e-commerce (venda de livros, eletrodomésticos, etc.)
 - Aluguel de hardware/servidores de nuvem é um dos negócios mais rentáveis do mundo atualmente.

Leitura sugerida:

- <https://aws.amazon.com/pt/blogs/aws-brasil/dez-dicas-para-otimizacao-de-custos-na-aws-2020/>

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Empresas que adotam a arquitetura de microsserviços geralmente o fazem depois de terem construído uma aplicação e encontrado desafios de escalabilidade.
- Elas começam com uma aplicação monolítica e depois dividem o monólito em microsserviços.

Arquitetura de microsserviços

- Susan J. Fowler, autora do livro *“Microsserviços prontos para a produção”*, trabalhou como Engenheira de Confiabilidade (SRE) na gigantesca migração da arquitetura monolítica da **Uber** para mais de 1000 microsserviços.



Arquitetura de microserviços

Arquitetura **Microserviços**

- A decisão de migrar para microserviços deve ser sempre um esforço que envolva toda a empresa. Além disso, é necessário ter pessoas no time que conheçam microserviços (você =D).
- Antes de chegarmos na prática (*show me the code*), vejamos as **etapas** para se dividir um monólito e os **elementos** que compõem uma arquitetura de microserviços (APIs, endpoints, etc).

Arquitetura de microserviços

Arquitetura **Microserviços**

- A primeira etapa para se dividir um monólito é **identificar os componentes que devem ser escritos como serviços independentes** (talvez essa seja a parte mais difícil do processo).
- A regra geral para identificar componentes é localizar com precisão as principais funcionalidades gerais do monólito e então dividir essas funcionalidades em pequenos componentes independentes.

Arquitetura de microserviços

Arquitetura **Microserviços**

- Os microserviços devem ser tão simples quanto possível, ou então a empresa correrá o risco de substituir um monólito por vários monólitos menores, que sofrerão dos mesmos problemas à medida que a empresa crescer.
- Depois que as funções-chave tiverem sido transformadas em microserviços, o próximo passo é alterar a estrutura organizacional de modo que os microserviços tenham sua própria equipe de engenharia de sistemas.

Arquitetura de microserviços

Arquitetura **Microserviços**

- A criação de um **ecossistema** de microserviços é fundamental nesse processo. Quando um monólito é dividido em microserviços, as responsabilidades da **infraestrutura** organizacional em prover uma **plataforma estável** para que os microserviços sejam desenvolvidos e mantidos crescem drasticamente.
- As equipes de infra devem fornecer às equipes de microserviços uma **infra estável** que abstraia a maiores das complexidades das interações entre os microserviços.

Arquitetura de microsserviços

Empresas pequenas, que ainda não tenham passado por problemas de escalabilidade, que não possuem áreas e times bem definidos, que não contrataram bons DevOps e Arquitetos de Nuvem e de Sistemas em seus times, não terão capacidade operacional suficiente para manter um ecossistema de microsserviços.



Arquitetura de microsserviços

Arquitetura **Microsserviços**

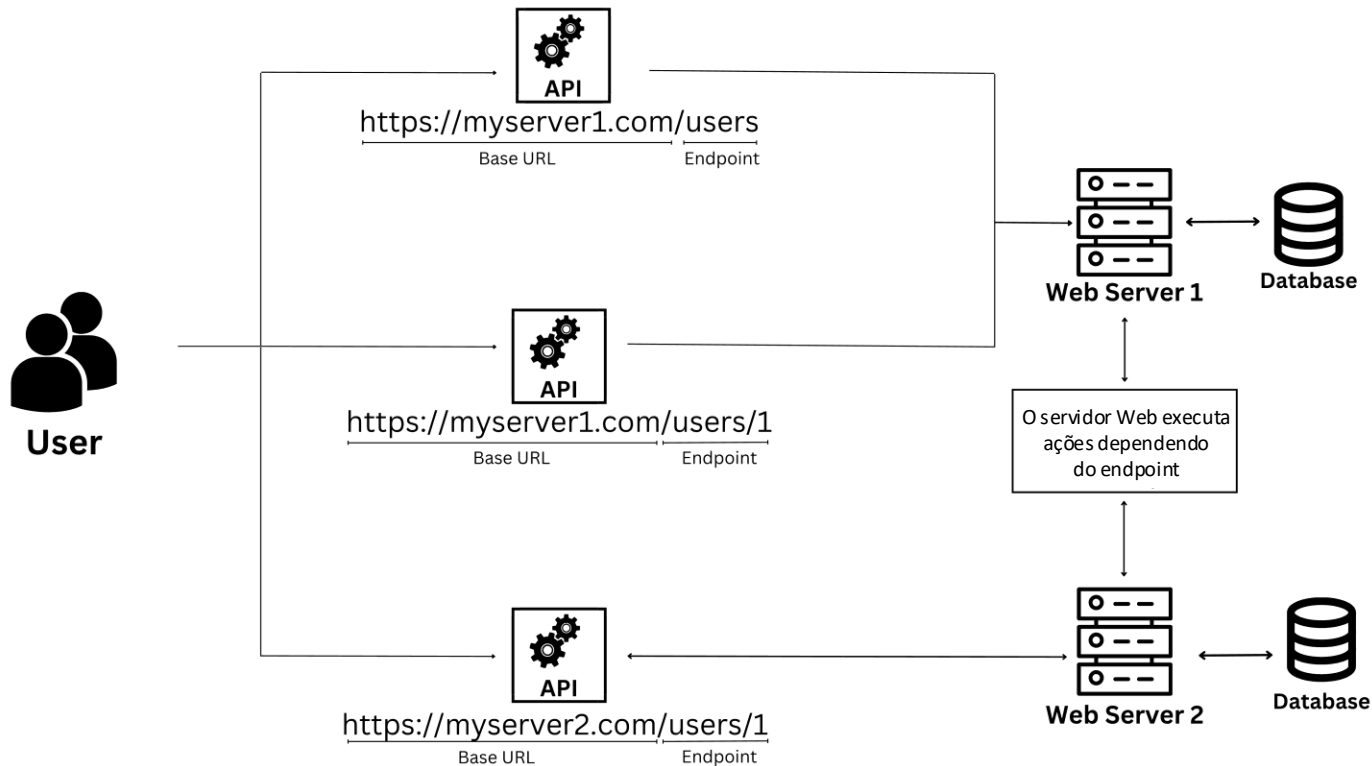
- Vejamos agora os **elementos** que compõem uma arquitetura de microsserviços simples.
- Todo microsserviço terá três componentes (muito parecido com o modelo de três camadas):
 - Um elemento frontend, algum código de backend que faça o trabalho pesado e uma maneira de armazenar ou obter dados relevantes.

Arquitetura de microserviços

Arquitetura **Microserviços**

- O elemento frontend de um microserviço não é a típica aplicação de frontend, mas uma API (interface de programação de aplicação) com endpoints estáticos (pontos de acesso).
- APIs de microserviço bem projetadas permitem que os microserviços interajam de forma fácil e eficaz, **enviando solicitações aos endpoints** de APIs relevantes.

Arquitetura de microsserviços



Arquitetura de microsserviços

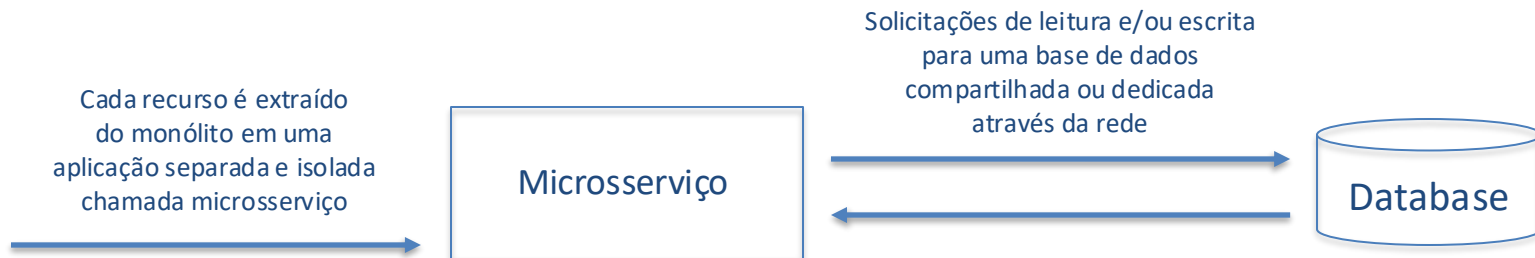
Arquitetura **Microsserviços**

- Um exemplo simples:
 - um microsserviço responsável pelos dados do cliente pode ter um endpoint do tipo *get_customer_information* (obter informações do cliente) para o qual outros serviços podem **enviar solicitações** para autorizar as informações de um cliente específico e um endpoint *delete_customer_information* (apagar informações do cliente) que os serviços podem usar para apagar as informações de um cliente.

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Esses endpoints são separados apenas na arquitetura e na teoria, mas não na prática, pois eles convivem lado a lado como parte do código de um backend que processa todas as solicitações.



Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Para nosso exemplo de um microsserviço que é responsável pelos dados do cliente, uma **solicitação enviada** para o endpoint *get_customer_information* dispararia uma tarefa que processa a solicitação de entrada, determina quaisquer filtros ou opções específicas aplicadas à solicitação, **obtem as informações** de um database, formata as informações e as **devolve para o cliente** (microsserviço) que as solicitou.

Arquitetura de microserviços

Arquitetura **Microserviços**

- Em outras palavras: “Querido endpoint X, preciso de Y.”.
- Se você obtém o que deseja, não precisa entender como o microserviço foi construído. Não precisa saber nem mesmo em qual linguagem de programação ele foi escrito.

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- Escrever um microsserviço dá ao desenvolvedor bastante liberdade.
- O Dev fica livre para escrever a lógica interna de seus microsserviços da forma que desejar. Ele pode usar, por exemplo, **qualquer linguagem de programação**.

Arquitetura de microsserviços

Arquitetura **Microsserviços**

- A maioria dos microsserviços armazena algum tipo de dados, seja na memória (talvez usando um cache) ou em um database externo. Se os dados relevantes forem armazenados na memória, **não será preciso fazer uma chamada de rede extra para um database externo** e o microsserviço poderá facilmente retornar quaisquer dados relevantes para um cliente.
- Caso contrário, o microsserviço deverá aguardar a resposta para a sua solicitação.

Arquitetura de microserviços

Arquitetura **Microserviços**

- Esta arquitetura é necessária para que os microserviços funcionem bem em conjunto.
- O paradigma desta arquitetura requer que um conjunto de microserviços funcione junto para executar as ações que antes existiam na forma de uma grande aplicação, portanto há certos elementos dessa arquitetura que precisam ser padronizados para que um conjunto de microserviços interaja de forma bem-sucedida e eficiente.

Arquitetura de microserviços

Arquitetura **Microserviços**

- Os endpoints da API dos microserviços devem ser padronizados em toda a organização. Isso não quer dizer que todos os microserviços devam ter os mesmos endpoints específicos, mas o **tipo de endpoint** deve ser o mesmo.
- Vejamos então um tipo muito comum de endpoint de API para microserviços, as chamadas API **REST**.

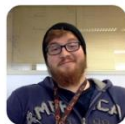
Sumário

- Arquitetura de microsserviços
 - Arquitetura de microsserviços
 - **Modelos de arquitetura REST**
 - Princípios e boas práticas de serviços REST

Modelos de arquitetura REST

- APIs REST são frequentemente usadas em etapas técnicas e projetos referentes à **processos seletivos** da área de desenvolvimento de software.
- Por isso é um assunto que importa não somente para a profissão mas também para ingressar no mercado como Dev Jr.

leoscalco/ Processolclinic



API REST em django referente ao processo seletivo para desenvolvedor BackEnd da Iclinic



1

Contributor



0

Issues



0

Stars

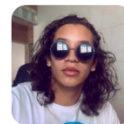


0

Forks



cleefsouza/**starwars-resistance**



API REST, Processo Seletivo - Phoebus Tecnologia



1

Contributor



0

Issues



0

Stars



1

Fork



Modelos de arquitetura REST

- Os microsserviços interagem entre si via **RPCs** (remote procedure calls) que são chamadas por meio de **redes** projetadas para se parecerem com e se comportarem exatamente como chamadas locais de procedimentos.
- Os **protocolos** usados dependem das escolhas de arquitetura e suporte organizacional, assim como dos endpoints usados. Um microsserviço com endpoint do tipo **REST**, provavelmente irá interagir com outros microsserviços via **HTTP**.

Modelos de arquitetura REST

- O que são APIs **REST** (Transferência Representacional de Estado)?
 - **REST** é um estilo arquitetônico que define um conjunto de princípios para projetar sistemas distribuídos e construir microserviços.
 - Este modelo foi proposto por Roy Fielding em sua tese de doutorado em 2000.



Modelos de arquitetura REST

- O que são APIs **REST** (Transferência Representacional de Estado)?
 - O objetivo do REST é criar sistemas que sejam escaláveis, flexíveis, e que possam ser entendidos e mantidos a longo prazo.
- Os princípios do REST incluem o uso de recursos identificados por **URIs**, a representação de recursos, a manipulação de recursos através de métodos do protocolo **HTTP** padrão (**GET**, **POST**, **PUT**, **DELETE**), e a independência de estado entre o cliente e o servidor.

Modelos de arquitetura REST

- O que são APIs **REST** (Transferência Representacional de Estado)?
 - O termo "**RESTful**" é frequentemente usado para descrever serviços ou APIs que aderem aos princípios da arquitetura REST.
 - Uma **API RESTful**, por exemplo, é uma API que utiliza os métodos HTTP de maneira adequada, usa URIs para identificar recursos, e fornece representações desses recursos.
 - O termo "RESTful" é uma maneira informal de indicar que uma implementação segue as melhores práticas e diretrizes da arquitetura REST.

Modelos de arquitetura REST

- Existem várias formas de implementar uma arquitetura RESTful e vários modelos possíveis.
- Entre os modelos mais utilizados estão: I) Modelo de servidor único (Single-Server); II) Modelo Cliente-Servidor; III) Modelo Cliente-Servidor com Camadas (Layered Architecture); Modelo Stateless (Sem estado) e Modelo HATEOAS (Hypermedia As The Engine Of Application State).

Modelos de arquitetura REST

- Dentre os modelos possíveis, focaremos no modelo **Stateless**.
- A statelessness, nesse contexto, significa que cada solicitação do cliente para o servidor **contém todas as informações necessárias para entender e processar a solicitação**, sem depender de nenhum estado armazenado no servidor entre as solicitações.

Sumário

- Arquitetura de microsserviços
 - Arquitetura de microsserviços
 - Modelos de arquitetura REST
 - **Princípios e boas práticas de serviços REST**

Princípios e boas práticas de serviços REST

- Existem várias razões pelas quais a statelessness é considerada uma prática recomendada em arquiteturas RESTful:
 - **Escalabilidade:** A ausência de estado no servidor torna mais fácil escalar, pois cada solicitação é independente e não requer informações de estado do servidor.
 - **Simplicidade:** A statelessness simplifica a lógica de servidor, uma vez que cada solicitação é autocontida e não há necessidade de rastrear o estado da sessão do cliente no servidor.

Princípios e boas práticas de serviços REST

- Existem várias razões pelas quais a statelessness é considerada uma prática recomendada em arquiteturas RESTful:
 - **Interoperabilidade:** A statelessness facilita a interoperabilidade entre clientes e servidores, pois cada solicitação contém todas as informações necessárias para sua execução.
 - **Tolerância a Falhas:** A statelessness melhora a tolerância a falhas, uma vez que cada solicitação é independente e não depende do estado persistente no servidor.

Princípios e boas práticas de serviços REST

- Embora a statelessness seja um princípio fundamental, é importante notar que não é uma regra estrita em todos os casos.
- Em algumas situações, pode haver exceções em que é necessário armazenar algum estado no servidor para atender a requisitos específicos do aplicativo.

Vamos à prática!

- Vamos então construir nossa primeira **API RESTful** utilizando Java 17 e Spring Boot.
- O passo a passo leva menos de 5 minutos! Esse é o grande benefício de se utilizar um **framework**.



Vamos à prática!

- **Spring Framework:** É um framework abrangente para o desenvolvimento de aplicativos Java empresariais. Ele fornece uma variedade de módulos e funcionalidades para tratar de aspectos como injeção de dependência, AOP (Programação Orientada a Aspectos), transações, segurança, entre outros.
- **Spring Boot:** É construído sobre o Spring Framework e é projetado para simplificar significativamente o processo de configuração e desenvolvimento de aplicativos Spring. O Spring Boot visa facilitar o desenvolvimento de aplicativos Java com configurações mínimas.

Vamos à prática!

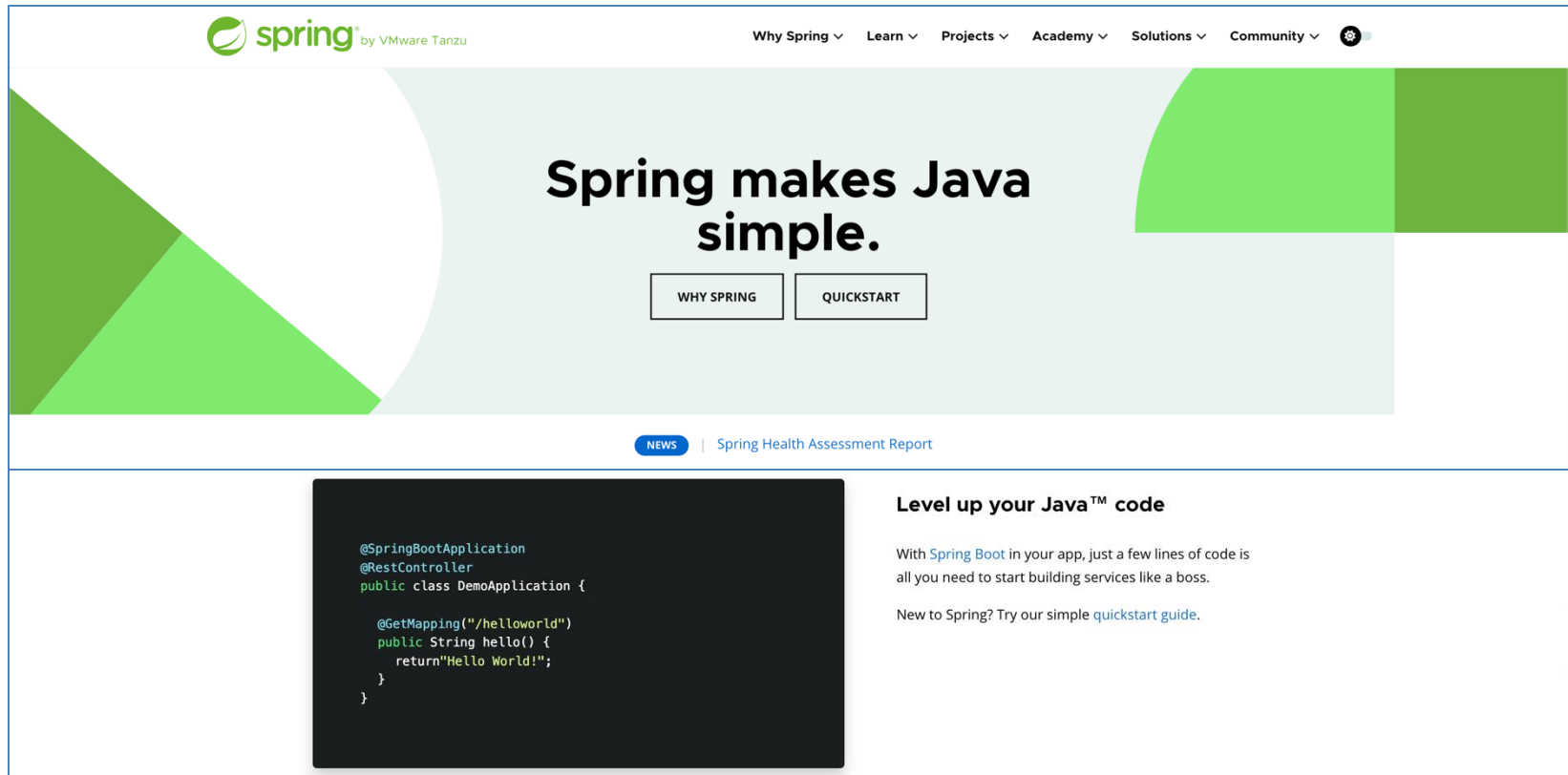
- **Spring Framework:**

- <https://spring.io/>
- <https://spring.io/quickstart>
- <https://spring.io/projects/spring-framework>

- **Spring Boot:**

- <https://spring.io/projects/spring-boot>
- <https://docs.spring.io/spring-boot/docs/current/reference/html/>

Vamos à prática!



The screenshot shows the Spring website homepage. At the top left is the Spring logo with the text "by VMware Tanzu". To the right is a navigation bar with links: "Why Spring", "Learn", "Projects", "Academy", "Solutions", and "Community", each followed by a dropdown arrow. A settings gear icon is on the far right. The main hero section has a light blue background with abstract green and white shapes. It features the headline "Spring makes Java simple." in large black font. Below the headline are two buttons: "WHY SPRING" and "QUICKSTART". A "NEWS" button is located below the hero section, followed by a link to "Spring Health Assessment Report". The bottom section is divided into two parts. On the left is a dark code editor showing Java code for a Spring Boot application. On the right is a text area with the heading "Level up your Java™ code" and two paragraphs of text.

spring by VMware Tanzu

Why Spring ▾ Learn ▾ Projects ▾ Academy ▾ Solutions ▾ Community ▾ ⚙️

Spring makes Java simple.

WHY SPRING QUICKSTART

NEWS | [Spring Health Assessment Report](#)

```
@SpringBootApplication
@RestController
public class DemoApplication {

    @GetMapping("/helloworld")
    public String hello() {
        return "Hello World!";
    }
}
```

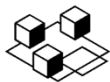
Level up your Java™ code

With [Spring Boot](#) in your app, just a few lines of code is all you need to start building services like a boss.

New to Spring? Try our simple [quickstart guide](#).

Vamos à prática!

What Spring can do



Microservices

Quickly deliver production-grade features with independently evolvable microservices.



Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.



Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.



Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.



Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.




Batch

Automated tasks. Offline processing of data at a time to suit you.

Vamos à prática!

- 1º) Acesse: <https://start.spring.io/>
 - Configure conforme a imagem a seguir.
 - Project **Maven** // Java **17** // Spring Boot **3.2.1**
 - Dependências: **Spring Web** (Para construir a API **RESTful**)
 - Por fim, clique em **Generate!**

Vamos à prática!



Project
☐ Gradle - Groovy ☐ Gradle - Kotlin
☒ Maven

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 3.3.0 (SNAPSHOT) ☐ 3.2.2 (SNAPSHOT) ☒ 3.2.1 ☐ 3.1.8 (SNAPSHOT)
☐ 3.1.7

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

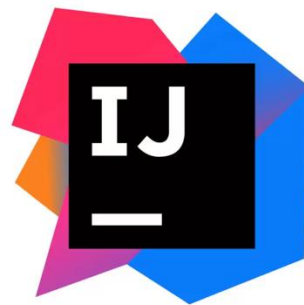
Java ☐ 21 ☒ 17

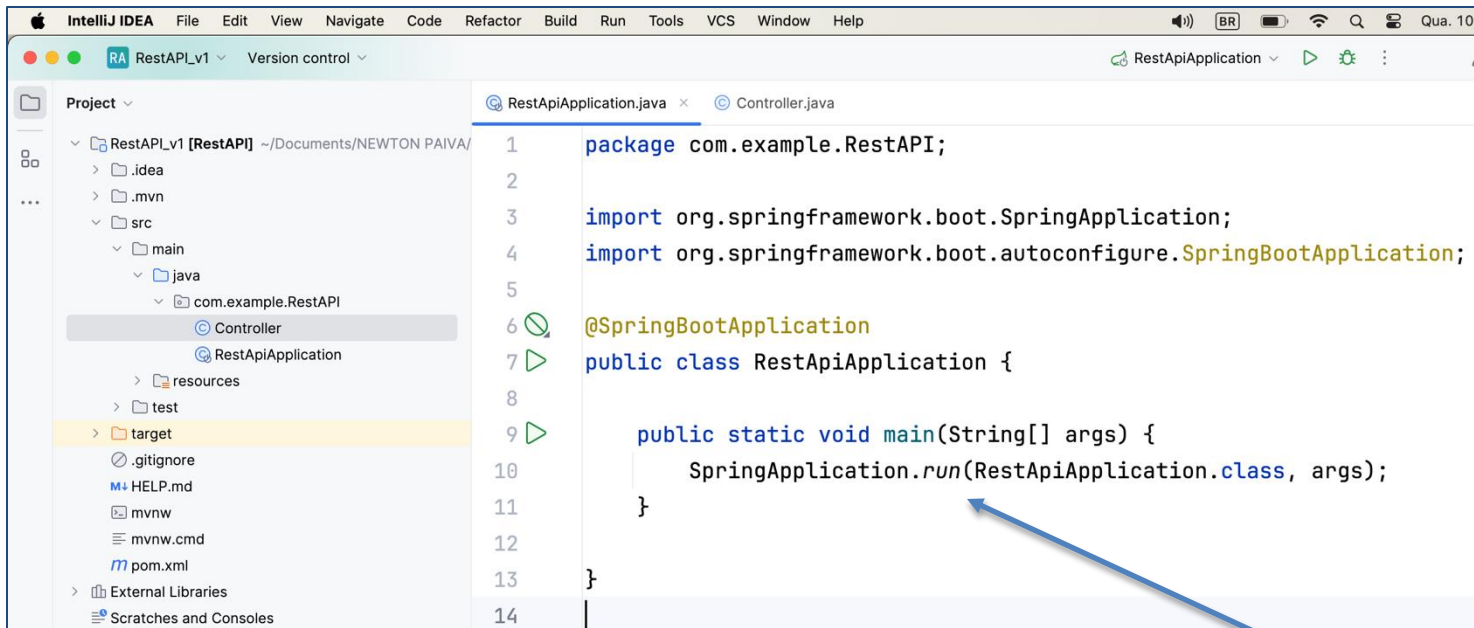
Dependencies

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

Vamos à prática!

- 2º) Depois de clicar em Generate, conclua o download do projeto. Neste exemplo, o projeto se chama **RestAPI**.
- Feito o download, abra o projeto com o seu **IntelliJ** IDEA.





O Spring Boot já criou para você uma classe **Application** com o método **main** startando a aplicação Spring.

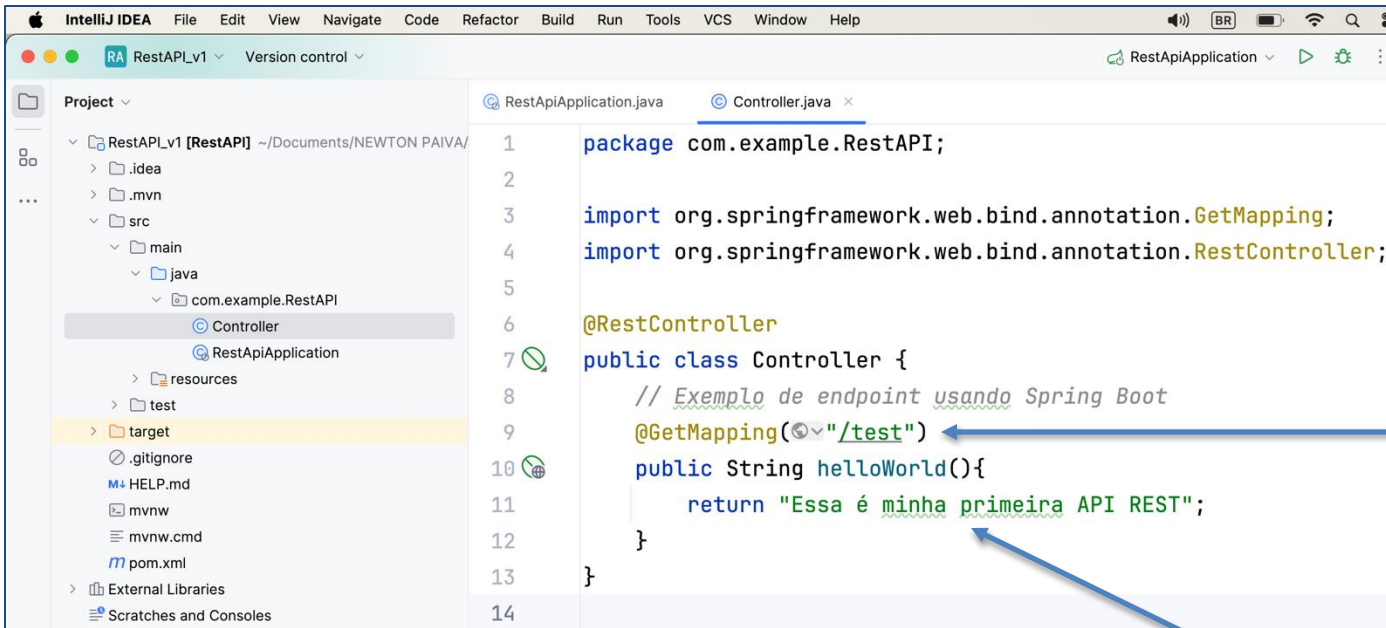
Porém a API REST ainda não está pronta.

Vamos à prática!

- 3º) Crie uma classe **Controller** e utilize as annotations:

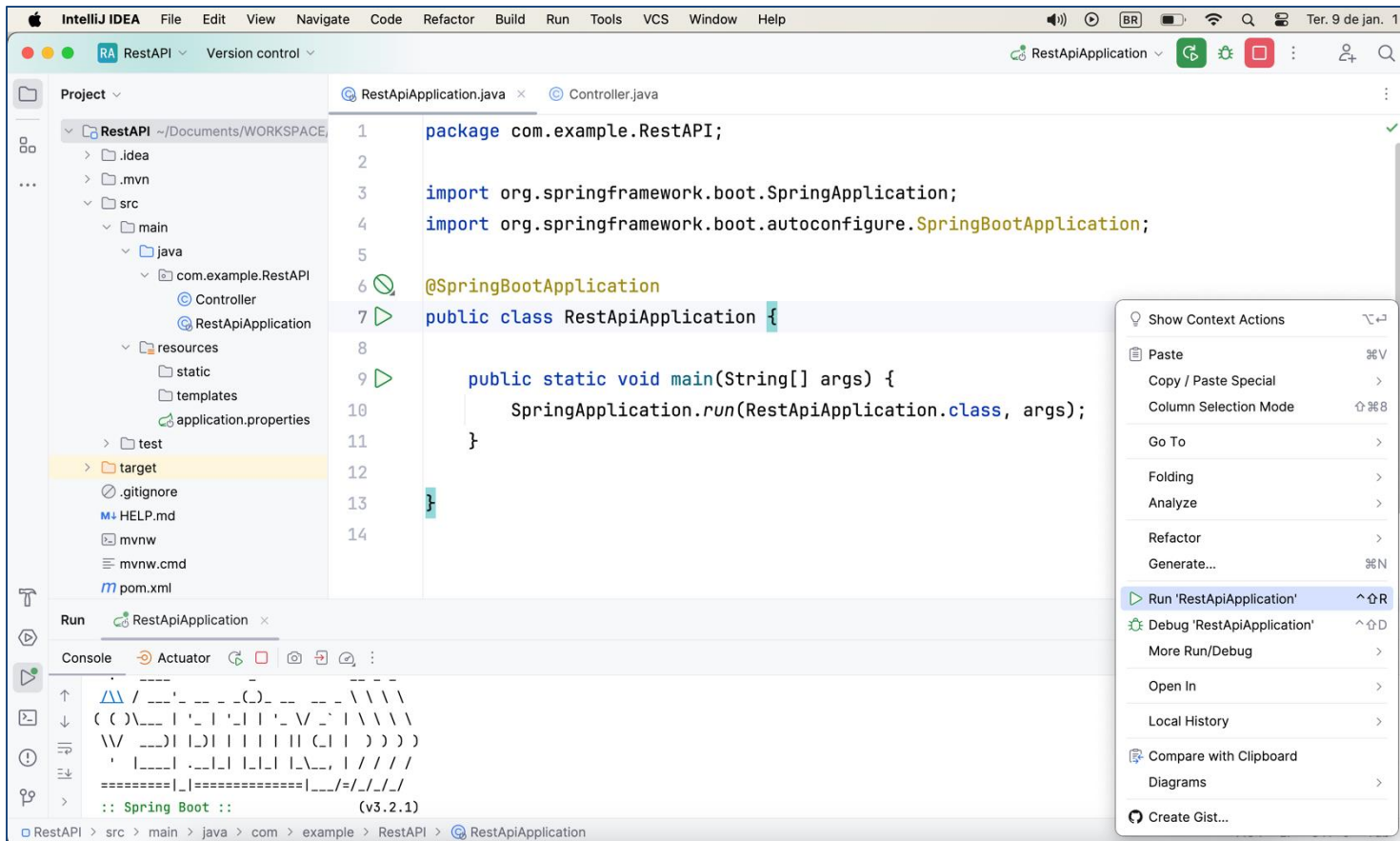
```
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;
```

- A anotação **@RestController** é uma versão especializada da anotação **@Controller** do Spring, que indica que a classe é um controlador que lida com solicitações HTTP e produz uma resposta HTTP adequada para a web.
- A anotação **@GetMapping** é uma forma especializada de **@RequestMapping** que associa solicitações HTTP do tipo GET a métodos específicos em um controlador. Neste caso, o método **HelloWorld()** será chamado quando uma solicitação **GET** for feita para a rota ("/test").



O valor dentro de `@GetMapping("/test")` especifica a **URI** (Uniform Resource Identifier) ou caminho da solicitação que acionará esse método.

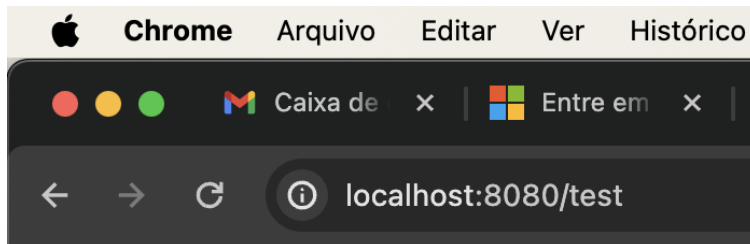
Essa mensagem irá aparecer no browser quando fizermos uma chamada **HTTP** do tipo **GET**



4°) Execute a aplicação.

Vamos à prática!

- 5º) Abra o browser de sua preferência e faça uma chamada HTTP do tipo GET:
 - <http://localhost:8080/test>
 - **localhost** pois estamos rodando o projeto em um servidor local.
 - 8080 é a porta padrão para servidores web de desenvolvimento.
 - /test é o caminho ou URL que configuramos como a nossa rota de teste.



Essa é minha primeira API REST

Vamos à prática!

- Muito bem! Podemos agora construir novas APIs REST para diversas finalidades diferentes, como por exemplo:
 - Consultar o clima na sua cidade:
 - <https://advisor.climatempo.com.br/>
 - Reproduzir músicas do Spotify diretamente nas suas aplicações:
 - <https://developer.spotify.com/documentation/web-api>
 - Recuperar dados do GitHub:
 - <https://docs.github.com/pt/rest?apiVersion=2022-11-28>
 - Recuperar dados sobre personagens, quadrinhos e eventos da Marvel:
 - <https://developer.marvel.com/>
 - Analisar dados com inteligência artificial do Google:
 - <https://cloud.google.com/apis>



Entenda Melhor

A Climatempo disponibiliza através de uma API a previsão do tempo e outros dados meteorológicos em tempo real. Nunca foi tão fácil integrar dados ao seu negócio!



Uptime de 99.8%

Estrutura completamente dedicada a garantir estabilidade e um uptime de 99,8%.



API REST

Facilitamos a integração dos dados requisitados em qualquer aplicação.



Documentação

Documentos completos e de fácil entendimento para você saber utilizar nossa API de previsão do tempo.



Suporte ao desenvolvimento

Nossa equipe estará pronta para ajudar e tirar dúvidas sobre a integração dos dados.



Atualizações

Trabalho contínuo de melhorias do produto através de feedbacks dos nossos usuários.



A melhor previsão

Hoje a Climatempo é a principal empresa privada de meteorologia da América e possuímos o melhor índice de acertos da previsão do tempo.

Web API

Overview

Getting started

Concepts

Tutorials

How-Tos

REFERENCE

Albums

Artists

Audiobooks

Categories

Chapters

Episodes

Genres

Markets

Player

Playlists

Web API

Retrieve metadata from Spotify content, control playback or get recommendations

Spotify Web API enables the creation of applications that can interact with Spotify's streaming service, such as retrieving content metadata, getting recommendations, creating and managing playlists, or controlling playback.

Getting started

This is where the magic begins! The following steps will help you to get started with your journey towards creating some awesome music apps using the API:

1. Log into the [dashboard](#) using your Spotify account.
2. [Create an app](#) and select "Web API" for the question asking which APIs are you planning to use. Once you have created your app, you will have access to the app credentials. These will be required for API [authorization](#) to obtain an [access token](#).
3. Use the [access token](#) in your [API requests](#).

REST API

API Version: 2022-11-28 (latest)

Início rápido

Sobre a API REST

Usando a API REST

Autenticação

Guides

Acções

Atividade

Aplicativos

Cobrança

Branches

Verificações

Sala de aula

Verificação de código

Códigos de conduta

Crie integrações, recupere dados e automatize seus fluxos de trabalho com a API REST do GitHub.

Visão geral

Guia de Início Rápido

Comece por aqui [Exibir tudo](#) →

Sobre a API REST

Obtenha orientação sobre a documentação da API REST.

Introdução à API REST

Saiba como usar a API REST do GitHub.

Autenticação na API REST

Você pode se autenticar na API REST para acessar mais pontos de extremidade e ter um limite de taxa mais alto.

Popular

Limites de taxa para a API REST

Saiba mais sobre os limites de taxa da API REST, como não ultrapassá-los e o que fazer se você excedê-los.

Solucionar problemas do API REST

Saiba como diagnosticar e resolver problemas comuns de API REST.

Scripts com a API REST e o JavaScript

Escreva um script usando o SDK do Octokit.js para interagir com a API REST.

Novidades [Exibir tudo →](#)

Upcoming changes to repository insights

November 29

GitHub Actions – Enforcing workflow scope when creating a release

November 02

Additional repository metrics available via GraphQL API

October 24

developer.marvel.com

SIGN IN

JOIN

MARVEL

MARVEL UNLIMITED
SUBSCRIBE

NEWS

COMICS

CHARACTERS

MOVIES

TV SHOWS

GAMES

VIDEOS

MORE

DEVELOPER PORTAL

How-Tos

Interactive Documentation

Get a Key


Help


News and Updates

CREATE AWESOME STUFF
WITH THE WORLD'S
GREATEST COMIC API

The Marvel Comics API allows developers everywhere to access information about Marvel's vast library of comics—from what's coming up, to 70 years ago.


GET STARTED






API Documentation »

Read the documentation and rules of the road for the Marvel



Test Calls »

Use the interactive test page to explore and test API calls.



Get a Key »

Generate your API key and start making cool products.

cloud.google.com/apis?hl=pt-br

Google Cloud

Informações Gerais Soluções Produtos Preços Recursos

🔍

📄

Documentos Suporte

Português -...

Console

⋮

Entre em contato conosco

APIs de IA e machine learning

APIs do Cloud

Visão geral

APIs de IA e machine learning

APIs do Compute

APIs de armazenamento e banco de dados

APIs de rede

APIs de análise de dados

APIs de ferramentas de gestão

APIs operacionais

APIs de segurança e identidade

APIs de infraestrutura gerenciada

API Vertex AI

Treine modelos personalizados de machine learning de alta qualidade com o mínimo de experiência e esforço.

APIs do modelo de fundação

Modelos grandes multitarefa pré-treinados, como o PaLM 2, que podem ser ajustados ou personalizados para tarefas específicas usando a Vertex AI.

API Speech-to-Text

Usa um reconhecimento de fala rápido e preciso para converter áudio em texto em mais de 125 idiomas e variantes.

API Cloud Natural Language

Analisa a estrutura e o significado do texto, incluindo análise de sentimento, reconhecimento de entidades e anotações no texto.

API Vision

Integra rotulagem de imagens, detecção de rostos, logotipos e paisagens, reconhecimento óptico de caracteres (OCR, na sigla em inglês) e detecção de conteúdo explícito nos apps.

API Cloud Translation

Traduz texto de um idioma para outro.

<https://console.cloud.google.com/apis/dashboard>

Google Cloud

Projeto vie

Pesquise (/) recursos, documentos, produtos e muito mais

Pesquisa

APIs e serviços

APIs e serviços

+ ATIVAR APIS E SERVIÇOS

APIs e serviços ativados

Biblioteca

Credenciais

Tela de permissão OAuth

Contratos de uso de página

1 hora 6 horas 12 horas **✓ 1 dia** 2 dias 4 dias 7 dias 14 dias 30 dias

Tráfego

▲ Não há dados disponíveis no período selecionado.

UTC-3 9 de jan. 06:00 12:00

Erros

▲ Não há dados disponíveis no período selecionado.

UTC-3 9 de jan. 06:00 12:00

Latência mediana

▲ Não há dados disponíveis no período selecionado.

UTC-3 9 de jan. 06:00 12:00

Gerenciador de API do
Google

<https://github.com/public-apis/public-apis>

README.md

Update README.md

4 months ago

README

MIT license

Public APIs

A collective list of free APIs for use in software and web development

Status

Number of Categories 51

Number of APIs 1427

Tests of push & pull failing

Validate links failing

Tests of validate package passing

The Project

[Contributing Guide](#) • [API for this project](#) • [Issues](#) • [Pull Requests](#) • [License](#)

Alternative sites for the project (unofficials)

[Free APIs](#) • [Dev Resources](#) • [Public APIs Site](#) • [Apihouse](#) • [Collective APIs](#)


☆ 275k stars

👁 4k watching

🍴 30.8k forks

Report repository

Contributors 1,262



+ 1,248 contributors

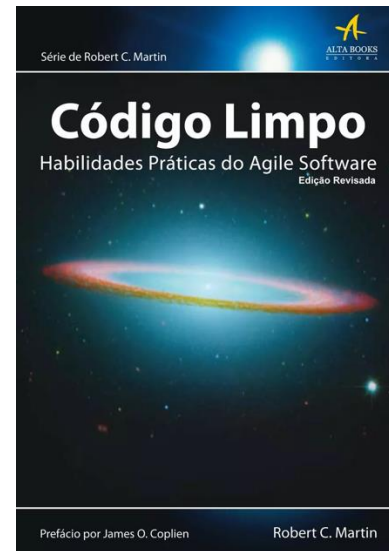
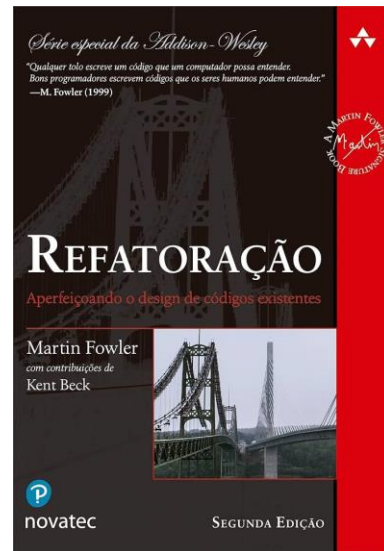
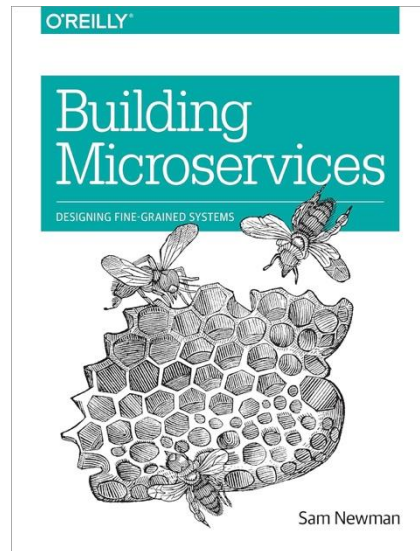
Languages

Python 96.8%

Shell 3.2%

Um super catálogo de APIs

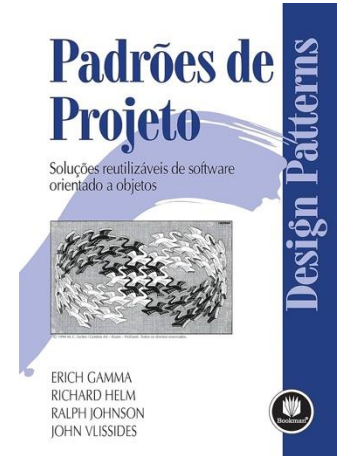
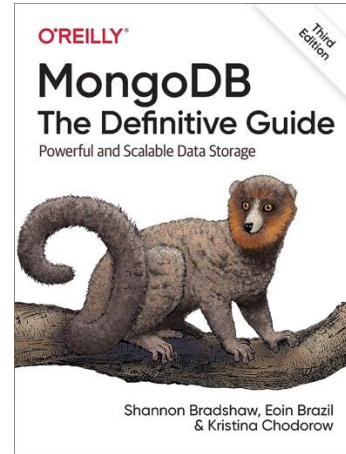
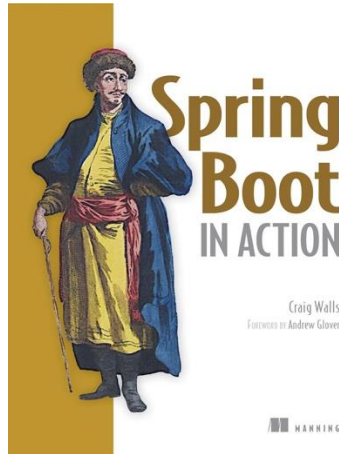
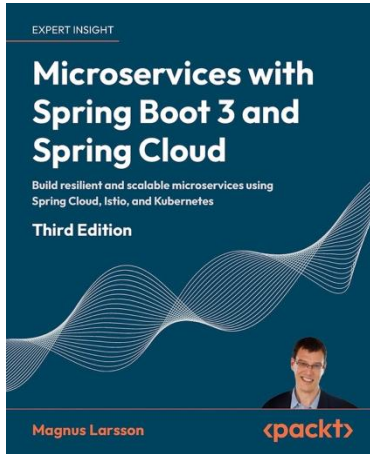
Referências básicas



Referências complementares



Outras referências



Obrigado!

Dúvidas?

joaopauloaramuni@gmail.com



[GitHub](#)



[LinkedIn](#)



[Lattes](#)

...