

Java e Microsserviços



A aplicação que será usada como exemplo no curso é:

Nela, eu tenho 2 Microsserviços, o de cadastro e o de pagamento.

Os dois vão sendo chamados à medida que vou avançando nas etapas de preenchimento do formulário.

E ai passado um tempo (30 s), posso chamar o login, pois os microservices já criaram para mim. O microsserviço acadêmico então será chamado:

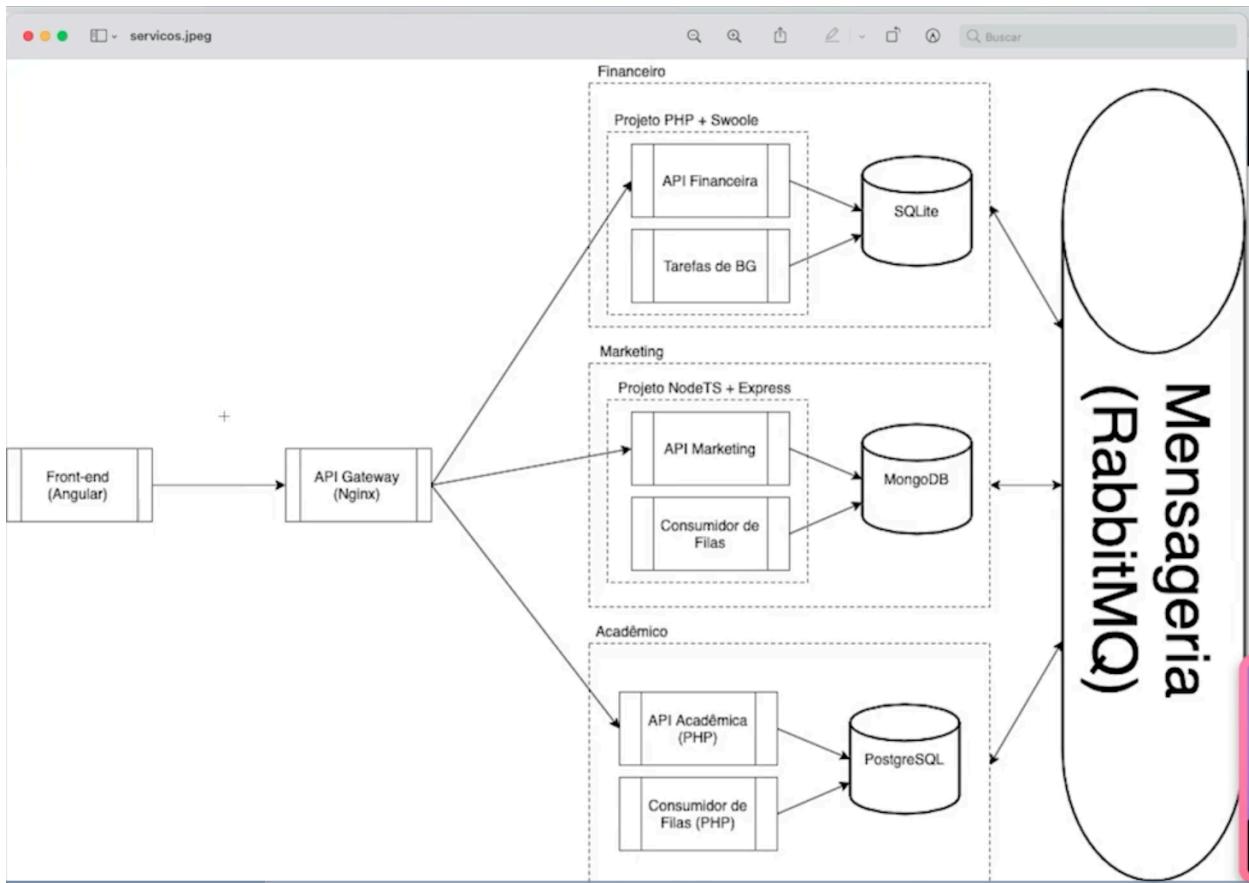
A screenshot of a web browser window titled "FrontEnd". The address bar shows the URL "http://localhost:4200/cursos". The main content area has a blue header with the text "ra". Below the header, there is a section titled "Cursos disponíveis" (Available Courses) containing a list of nine items, each with a small icon and a course name. To the right of each course name is a blue button with a white eye icon.

Cursos disponíveis
Microsserviços: Padrões de projeto
Mais sobre microsserviços
Integração contínua
Entrega contínua
12-Factor Apps
Docker
Kubernetes
Nginx 1
Nginx 2

Trazendo todos os cursos aos quais eu tenho acesso.

A arquitetura da aplicação é exibida abaixo:

Então, temos 3 microsserviços:



Temos o front-end feito com angular, que chama diretamente um API Gateway (Nginx).

Esse API Gateway tem a configuração para direcionar as requests para 3 APIs:

- Marketing: primeira API que o sistema consome, que é aquela primeira tela de dados pessoais, porque criamos um lead no marketing (situações em que o aluno não completa sua matrícula - teremos o nome e e-mail para ficar mandando campanhas, promoções, etc.). A API e o consumidor de filas estão no mesmo projeto, feito com Node e Typscript e o servidor web é o Express. Além disso, temos um BD, o MongoDB.
- Financeiro: segunda API, onde cadastramos na segunda tela as informações de financeiro. A API e a tarefa de Background estão no mesmo projeto também, já feito em PHP e o servidor web é o Swoole, que dá algumas facilidades para tarefas executadas em plano de fundo. O BD do financeiro é um banco relacional, um SQLite. O que acontece é que recebemos os dados financeiros pela API, e esses dados são mandados para uma tarefa de plano de fundo para que seja feito o processamento do pagamento (aqui verificamos se os dados do cartão são do titular, se está tudo certo, detalhes de fraudes, enviar o pedido para o Gateway de pagamentos (serviço externo)). Depois disso, o serviço financeiro manda uma mensagem (através de mensageria, utilizando o RabbitMQ

- um meio de campo, que fica no meio de várias aplicações recebendo mensagens, que são consumidas por outros serviços. Ele é um carteiro.).
- Acadêmico: o microsserviço acadêmico fica ouvindo as mensagens do RabbitMQ, esperando essa mensagem (essa carta) do microsserviço financeiro. Quando essa mensagem chega, ele pega o nome e e-mail que vieram e vai criar um aluno no nosso banco de dados PostgreSQL e gerar uma senha para ele e a partir do momento que essa senha é gerada, ele manda um e-mail para o aluno informando que ele já está pronto para acessar os cursos. Ao mesmo tempo que isso acontece, o serviço de Marketing também recebe essa carta, e muda o status do lead de interessado pelo curso para aluno do curso. E isso também é feito através das mensagens enviadas ao RabbitMQ.
A API vai conter informações de login do aluno e dos cursos, ou seja, se ele marca um curso como assistido, vai ser recebido via API e ela vai alterar o status no Banco de Dados. O Consumidor de filas vai criar o login do aluno.
Daria também para mandar para o RabbitMQ para o marketing saber quantos cursos o aluno fez, ter um outro serviço para gameficação, etc.

Se repararmos, hora nenhuma a comunicação entre os serviços foi feita de forma direta. Toda a comunicação está sendo feita de forma assíncrona. A única comunicação síncrona é entre o frontend e os microsserviços, pois o frontend espera o retorno das APIs. Mas as APIs e tarefas de plano de fundo não se importam com as respostas dos outros, eles só executam e mandam mensagens.

Conhecimento: todos os componentes de um mesmo serviço devem usar a mesma linguagem de programação?

The screenshot shows a digital learning interface. At the top, there's a navigation bar with a menu icon, the text "04 Componentes de linguagens", and a blue button labeled "PRÓXIMA ATIVIDADE". Below the navigation bar, there's some introductory text: "Vimos neste vídeo que um microsserviço pode ser composto por mais de um componente. Uma API e um leitor de mensagens da fila, por exemplo." Then, the main question is asked: "Todos os componentes de um mesmo serviço precisam usar a mesma linguagem de programação?"

The question has three options:

- A** Não, porém é aconselhável. ✖
Alternativa correta! Embora cada componente possa usar uma linguagem diferente, normalmente uma equipe é responsável por todo o serviço, então utilizar uma única linguagem pode facilitar a manutenção da equipe e contratação de novos colaboradores.
- B** Não. Não faz diferença. ✖
Alternativa errada! Realmente não é obrigatório, mas faz sim diferença essa escolha.
- C** Sim, é obrigatório. ✖
Alternativa errada! Não é obrigatório o uso de uma única linguagem por serviço.

O projeto está aqui:

The screenshot shows the GitHub repository page for `CViniciusSDias/alura-ms`. The repository has 4 stars, 100 forks, and 112 issues. The commit history lists 35 commits from various contributors, mostly related to updating submodules and managing SSH/HTTPS. The repository is described as part of a course on microservices.

Commit	Author	Message	Date
CViniciusSDias Alterando de SSH para HTTPS	CViniciusSDias	Alterando de SSH para HTTPS	7512cfa - last month
academico-php @ 53c0cc7	academico-php	Atualizando submódulo	2 years ago
academico-php-web @ 3d04486	academico-php-web	Atualizando submódulos	2 years ago
financeiro-php @ 6998f77	financeiro-php	Travando versão do swoole em financeiro	last month
front-end @ 78dcad7	front-end	Atualizando submódulo de front-end	3 years ago
mkt-node @ a2769d2	mkt-node	Atualizando submódulos	3 years ago
servicos-nginx		Adicionando o componente web do microserviço acadêmico	3 years ago
.gitignore		Atualizando submódulos	3 years ago
.gitmodules		Alterando de SSH para HTTPS	last month
academico-php-web.sh		Criando os entrypoints de cada serviço	3 years ago
academico-php.sh		Criando os entrypoints de cada serviço	3 years ago
docker-compose.yml		Travando node na versão 16	2 years ago
financeiro-php.sh		Criando os entrypoints de cada serviço	3 years ago
front-end.sh		Criando os entrypoints de cada serviço	3 years ago
mkt-node.sh		Criando os entrypoints de cada serviço	3 years ago

Link: <https://github.com/CViniciusSDias/alura-ms>

Agora vamos instalar na máquina.

Ele utilizou nesse repositório o conceito de submódulos (explicado logo a frente).

Portanto, para o clone, vamos ter que executar o seguinte comando:

```
jooao in Feitos/Alura/Microserviços
→ git clone --recurse-submodules --remote-submodules git@github.com:CViniciusSDias/alura-ms.git
```

Isso faz o clone do submodule central e também faz o clone de cada submodule.

Conhecimento: vamos usar o docker.

Como funciona e como usar? :

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-20-04-pt>

<https://docs.docker.com/compose/>

Introdução

O Docker simplifica o fluxo de gerenciamento de processos de aplicações em contêineres. Embora os contêineres sejam semelhantes às máquinas virtuais em certos aspectos, eles são mais leves e fáceis de usar. Isso permite que os desenvolvedores dividam um ambiente de aplicação em vários serviços isolados.

Para aplicações que dependem de vários serviços, orquestrar todos os contêineres para iniciar, comunicar e fechar juntos pode tornar-se algo rapidamente incontrolável. O [Docker Compose](#) é uma ferramenta que lhe permite executar ambientes de aplicações com vários contêineres com base nas definições contidas em um arquivo YAML. Ele usa as definições de serviço para compilar ambientes totalmente personalizados com contêineres múltiplos que podem compartilhar redes e volumes de dados.

Neste guia, vamos demonstrar como instalar o Docker Compose em um servidor Ubuntu 20.04 e como começar a usar esta ferramenta.

Instalando o docker:

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04>

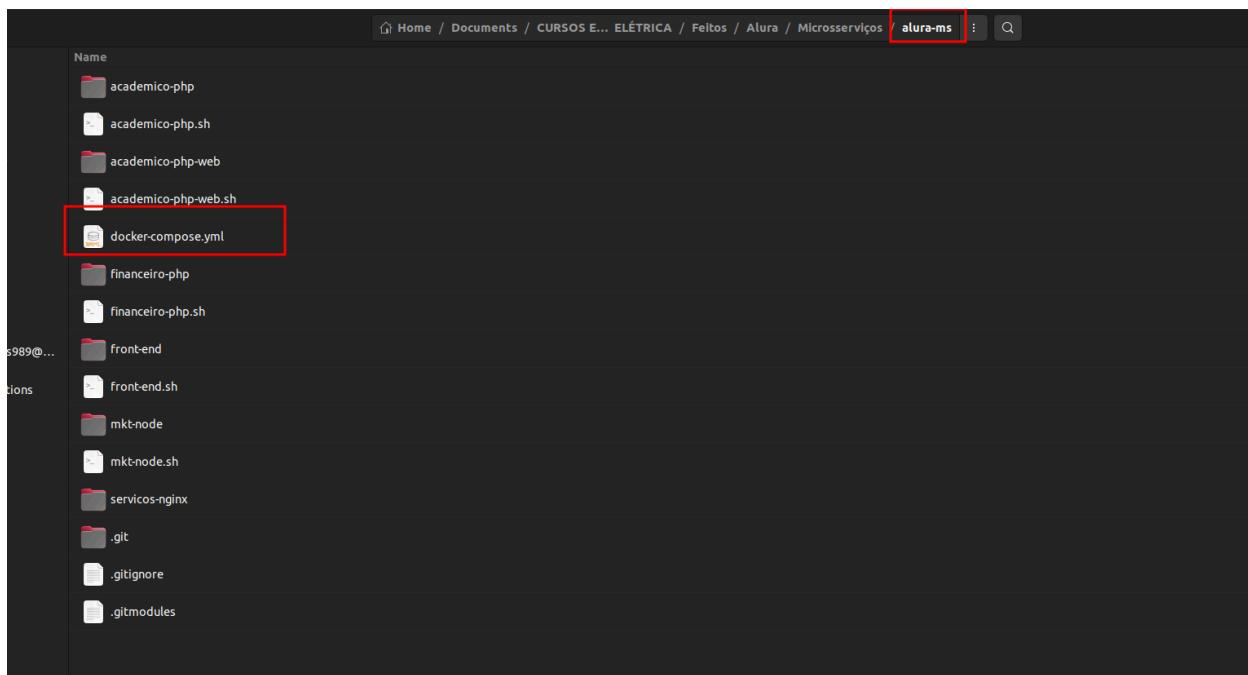
Obs: para usar o docker-compose, tive que usar como sudo.

Após instalação do docker e docker-compose, vamos iniciar a aplicação. Basta entrar no diretório da aplicação e rodar um “sudo docker-compose up --build”, ou seja, vou instalar todas as dependências no docker, para que meu projeto consiga rodar.

Essa etapa demora bastante.

```
joao in alura-ms on ✘ main
$ sudo docker-compose up --build
Creating network "alura-ms_default" with the default driver
Pulling front (node:16)...
16: Pulling from library/node
311d0ac465ea: Pull complete
7e9bf114588c: Pull complete
ffd9397e94b7: Pull complete
513d77925604: Pull complete
ae2b00f35300: Pull complete
0e42fc0ff4f4: Pull complete
ca266fd61921: Pull complete
e97d78be1eb9: Pull complete
Digest: sha256:777a1aeaf2da8d83a45ec990f45df50f1a286c5fe8bbfb8c0e4246c6389705c0b
Status: Downloaded newer image for node:16
Pulling mongo-mkt (mongo:)...
latest: Pulling from library/mongo
3713021b0277: Pull complete
39bdcacc9d97: Pull complete
d6b691142508: Pull complete
bcc1924dee0d: Pull complete
991a7990873d: Pull complete
77e5254f6a88: Pull complete
403f3af5f507cc: Pull complete
80003b3e087c: Pull complete
Digest: sha256:d672a079266a46faee269e0b5c0b1c7b9d9de3ddd8a1d5097a0881e15576bbb4
Status: Downloaded newer image for mongo:latest
Pulling rabbitmq (rabbitmq:)...
latest: Pulling from library/rabbitmq
3713021b0277: Already exists
1ecd0a70azfd5: Pull complete
508f9ebed870: Pull complete
775963a3c4364: Pull complete
5efeedb2eb07: Pull complete
a6122e2e132a0: Pull complete
bba0313c45e6: Pull complete
51e983ab78a8: Pull complete
13ff05a2922a0: Pull complete
310dccc310dcc83e3c9
Status: Downloaded newer image for rabbitmq:latest
Building web-financiero
Step 1/6 : FROM php:8.0.14
8.0.14: Pulling from library/php
a2abf6c4d29d: Pull complete
c5608244554d: Pull complete
2d07066487a0: Pull complete
1b0dfaf1958c: Pull complete
355a07f4a308: Pull complete
b540e144b47f: Pull complete
0deed0b3b380: Pull complete
3fbde89d2939: Pull complete
0e8e2ed22239: Pull complete
Digest: sha256:a00554e28cae156a0c9ab25db53155052973ac4d7459970a81a793348532691
Status: Downloaded newer image for php:8.0.14
--> 3f1426d77701
Step 2/6 : RUN apt-get update && apt-get install -y libzip-dev libssqlite3-dev && docker-php-ext-install zip
--> Running in e67893c72de0
```

Dependências vão ser baixadas, banco de dados vão ser criados, leva um tempinho.



OBS: vale ressaltar que precisamos rodar o docker-compose up dentro da pasta onde está o docker-compose.yml.

```
sudo docker-compose up --build
[Front_1] | Compiling @angular/cdk/a11y : es2015 as esm2015
[Front_1] | Compiling @angular/forms : es2015 as esm2015
[Front_1] | Compiling @angular/platform-browser/animations : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/collections : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/portal : es2015 as esm2015
[Front_1] | Compiling @angular/common/http : es2015 as esm2015
[Front_1] | Compiling @angular/material/core : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/overlay : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/layout : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/stepper : es2015 as esm2015
[Front_1] | Compiling @angular/cdk/text-field : es2015 as esm2015
[Front_1] | Compiling @angular/material/button : es2015 as esm2015
[Front_1] | Compiling @angular/material/card : es2015 as esm2015
[Front_1] | Compiling @angular/material/form-field : es2015 as esm2015
[Front_1] | Compiling @angular/material/divider : es2015 as esm2015
[Front_1] | Compiling @angular/platform-browser-dynamic : es2015 as esm2015
[Front_1] | Compiling @angular/router : es2015 as esm2015
[Front_1] | Compiling @angular/material/input : es2015 as esm2015
[Front_1] | Compiling @angular/material/list : es2015 as esm2015
[Front_1] | Compiling @angular/material/icon : es2015 as esm2015
[Front_1] | Compiling @angular/material/snack-bar : es2015 as esm2015
[Front_1] | Compiling @angular/material/toolbar : es2015 as esm2015
[Front_1] | Compiling @angular/material/tooltip : es2015 as esm2015
[Front_1] ✓ Browser application bundle generation complete.
[Front_1]
[Front_1] | Initial Chunk Files
[Front_1] vendor.js
[Front_1] polyfills.js
[Front_1] styles.css
[Front_1] main.js
[Front_1] runtime.js
[Front_1] | Initial Total | 2.87 MB
[Front_1]
[Front_1] | Lazy Chunk Files
[Front_1] default-node_modules_angular_cdk__ivy_ngcc__fesm2015_collections_js_node_modules_angular_cd-3180de.js | Names | Size
[Front_1] default-node_modules_angular_cdk__ivy_ngcc__fesm2015_layout_js_node_modules_angular_cdk__l-e0tab3.js | - | 847.98 kB
[Front_1] src_app_compra_compra.module.ts.js | - | 289.44 kB
[Front_1] default-node_modules_angular_material__ivy_ngcc__fesm2015_lcon_js-node_modules_angular_nate-0a30bd.js | - | 200.63 kB
[Front_1] default-node_modules_angular_material__ivy_ngcc__fesm2015_input_js.js | - | 178.40 kB
[Front_1] src_app_area-logada.area-logada_module_ts.js | - | 163.52 kB
[Front_1] src_app_login_login_module_ts.js | - | 87.84 kB
[Front_1] | - | 63.38 kB
[Front_1]
[Front_1] Build at: 2024-08-14T11:37:36.056Z - Hash: 72c89d3ac068d0c9abdc - Time: 23960ms
[Front_1]
[Front_1] ** Angular Live Development Server is listening on 0.0.0.0:4200, open your browser on http://localhost:4200/ **
[Front_1]
[Front_1] ✓ Compiled successfully. →
[Front_1] ✓ Browser application bundle generation complete.
[Front_1]
[Front_1] 12 unchanged chunks
```

Após demorar um tanto, ele foi compilado.

Subiu tudo, nos logs tem bastante coisa informativa.

Ele baixa todas as dependências, de todos os serviços e enquanto isso o RabbitMQ vai subindo, que ele demora um tempo para subir.

Temos umas migrations que são aplicadas no banco de dados e até alguns carregamentos de informações nele.

NOTA: no docker-compose, quando eu rodar, ele vai subir os serviços e liberar várias portas, é importante que elas não estejam sendo usadas por outras aplicações.

Se eu der um docker ps:

```
sudo su - joao
joao@alura-ms ~ % docker ps
[sudo] password for joao:
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS                               NAMES
13d36d0a08c8        nginx              "docker-entrypoint..."   3 hours ago       Up 3 hours          0.0.0.0:80->80/tcp, :::80->80/tcp
e5f91be19e3        alura-ms_consumer-academico   "docker-php-entrypoint..." 3 hours ago       Up 3 hours          0.0.0.0:9501->9501/tcp, :::9501->9501/tcp
emtico_1           alura-ms_web-financeiro    "docker-php-entrypoint..." 3 hours ago       Up 3 hours          0.0.0.0:9501->9501/tcp, :::9501->9501/tcp
93549ac5c54f        alura-ms_web-financeiro    "docker-php-entrypoint..." 3 hours ago       Up 3 hours          0.0.0.0:9501->9501/tcp, :::9501->9501/tcp
o_1                node:16              "docker-entrypoint.s..." 3 hours ago       Up 3 hours          0.0.0.0:3000->3000/tcp, :::3000->3000/tcp
d33aa70b9a0        alura-ms_web-academico    "docker-php-entrypoint..." 3 hours ago       Up 3 hours          0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
alura-ms_web-mkt_1
alura-ms_web-academico
alura-ms_web-financeiro
alura-ms_postgre-acade
alura-ms_rabbitmq_1
alura-ms_mongo-mkt_1
alura-ms_front_1
joao@alura-ms ~ % took 2,0s
```

Tenho vários containers referentes à minha aplicação.

Se eu acessar agora a porta 4200, tenho acesso ao front:

localhost:4200/compra

Sistemas Unes... Symbolab Mat... Curso em Vide... Lucidchart 16 Aulas Free... 16 Aulas Finan... Minicurso de... Aprenda Dash... Win-Win | Dem... Oportunidade... SciELO.org All Bookmark

alura

1 Dados pessoais **2 Detalhes de pagamento** **3 Fim**

Nome Completo

E-mail

Next

Informações da sua compra

Promo mega power

Valor total do plano **R\$ 123,45**

1234 cursos

- Estude por 1 ano
- Certificado de participação
- Apps para Android e iOS

Vamos cadastrar um usuário e ir acompanhando pelo terminal:

Preenchi essa primeira tela e:

O API Gateway já foi chamado e já temos informações no marketing. Se eu acessar a página do marketing, consigo visualizar:

```
Java e Microservicos - D x | A FrontEnd x localhost/mkt/leads +  
localhost/mkt/leads  
Sistemas Unes... Symbolab Mat... Curso em Vide... Lucidchart 16 Aulas Free... 16 Aulas Finan... Minicurso de... Aprenda Dash...  
Pretty-print  
{"email":"joao_victorbarros@hotmail.com","leadStatus":0}]
```

Já tenho o usuário que cadastrei aqui.

Agora vou preencher e enviar as infos de pagamento:

localhost:4200/compra

Sistemas Unes... Symbolab Mat... Curso em Vídeo... Lucidchart 16 Aulas Free... 16 Aulas Finan... Minicurso de... Aprenda Dash... Win-Win | Dem... Oportunidade... SciELO.org

alura

Dados pessoais

CPF (Agenzia numerico)
15478965498

Nome do titular do cartão
João Victor B

Número da conta
1234569874569874

Data de expiração
February 2025

Código de segurança
123

Detalhes de pagamento

Fim

Informações da sua compra

Plano mega power

Valor total do plano R\$ 123,45

1234 cursos

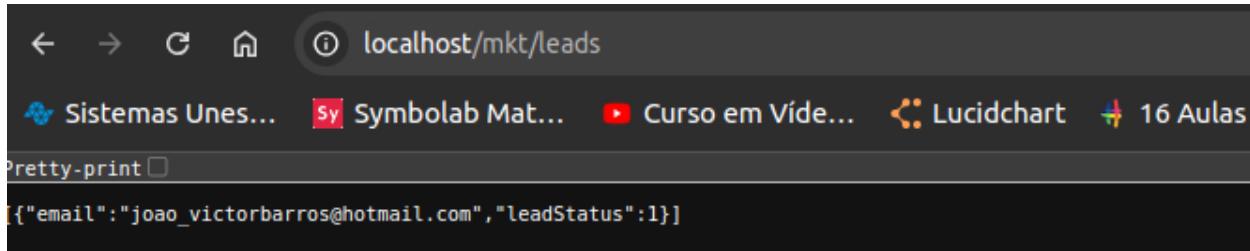
- Estude por 1 ano
- Certificado de participação
- Apps para Android e iOS

Next

```
api-gateway_1  | 172.18.0.1 - - [14/Aug/2024:14:26:59 +0000] "OPTIONS /financeiro/clients HTTP/1.1" 204 0 "http://localhost:4200/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36" "-"
home-financiero_1 | Processando pagamento de João Victor B
api-gateway_1  | 172.18.0.1 - - [14/Aug/2024:14:26:59 +0000] "POST /financeiro/clients HTTP/1.1" 201 0 "http://localhost:4200/" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36" "-"
rabbitmq_1   | 2024-08-14 14:27:20.719+00:00 [info] <0.1280.0> accepting AMQP connection <0.1280.0-> {172.18.0.9:42624 -> 172.18.0.5:5672}
rabbitmq_1   | 2024-08-14 14:27:20.720295+00:00 [info] <0.1280.0> {172.18.0.9:42624 -> 172.18.0.5:5672} user 'guest' authenticated and granted access to vhost '/'
```

Então o ms de pagamento começa a processar o pagamento, depois ele manda mensagem para o RabbitMq.

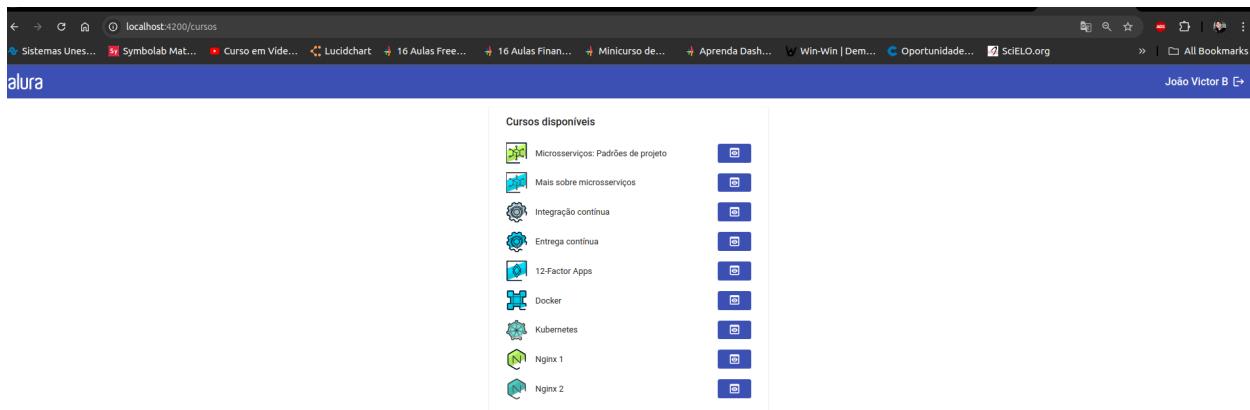
Então, o consumer-academico vai consumir essa mensagem e enviar um e-mail para o usuário informado e também o serviço de marketing consome essa mensagem e ocorre a mudança no marketing, de status do lead de 0 para 1, o que significa que o lead foi convertido:



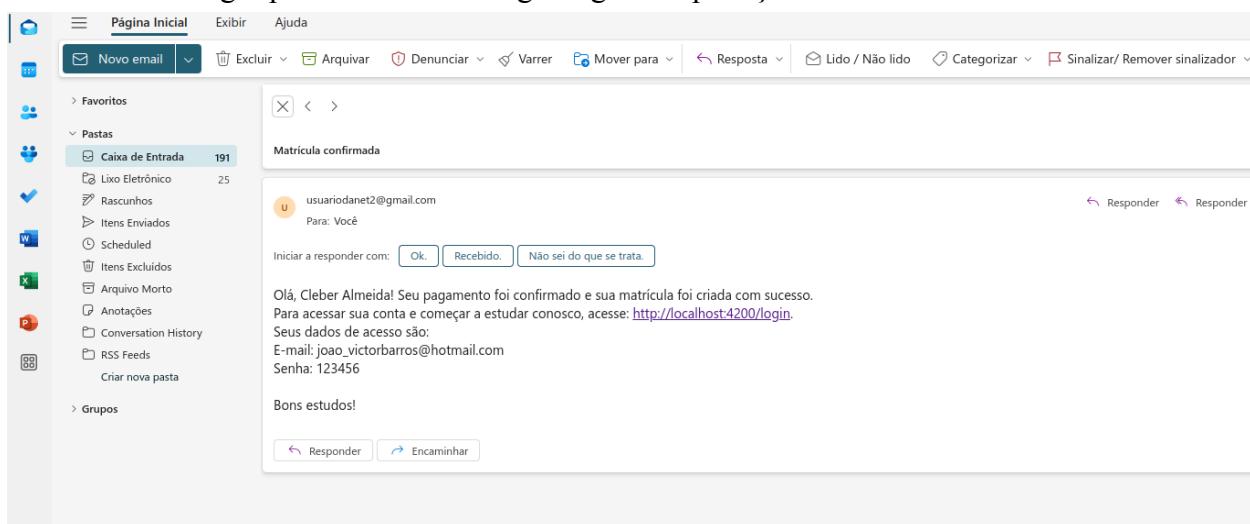
```
{"email": "joao_victorbarros@hotmail.com", "leadStatus": 1}
```

Então, todos os processos aconteceram.

E entrando na aplicação (login):



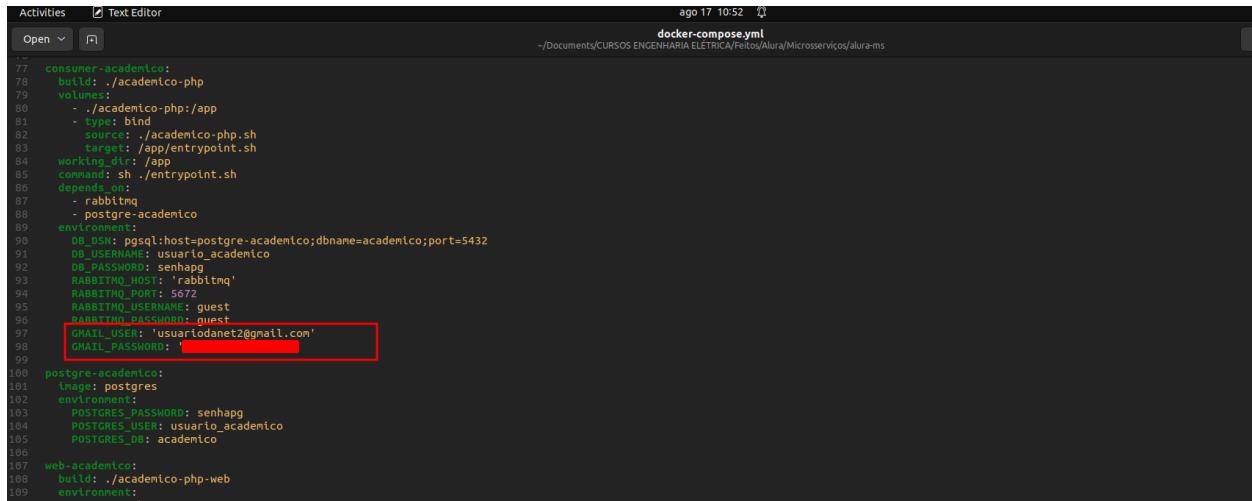
Então recebi o login pelo e-mail e consegui logar na aplicação e acessar meus cursos.



Olá, Cleber Almeida! Seu pagamento foi confirmado e sua matrícula foi criada com sucesso.
Para acessar sua conta e começar a estudar conosco, acesse: <http://localhost:4200/login>.
Seus dados de acesso são:
E-mail: joao.victorbarros@hotmail.com
Senha: 123456

Bons estudos!

Para receber o e-mail, tive que configurar no docker-compose.yml meu e-mail e senha e configurar no e-mail uma senha de aplicativos (para conseguir mandar um e-mail a partir do meu):

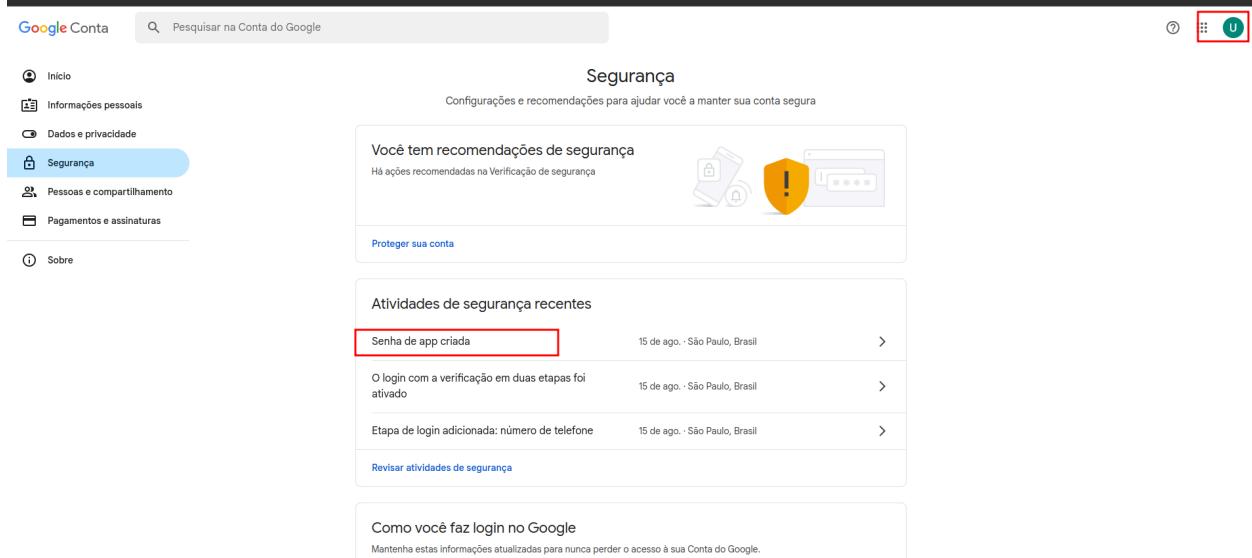


```

Activities Text Editor
Open ↗
docker-compose.yml
-/Documents/CURSOS ENGENHARIA ELÉTRICA/Felitoz/Alura/Microserviços/alura-ms
...
77 consumer-academico:
78   build: ./academico-php
79   volumes:
80     - ./academico-php:/app
81     - type: bind
82       source: ./academico-php.sh
83       target: /app/entrypoint.sh
84   working_dir: /app
85   command: sh ./entrypoint.sh
86   depends_on:
87     - rabbitmq
88     - postgres-academico
89   environment:
90     DB_HOST: pgsql:host=postgres-academico;dbname=academico;port=5432
91     DB_USERNAME: usuario_academico
92     DB_PASSWORD: senhapg
93     RABBITMQ_HOST: 'rabbitmq'
94     RABBITMQ_PORT: 5672
95     RABBITMQ_USERNAME: guest
96     RABBITMQ_PASSWORD: guest
97     GMAIL_USER: 'usuariodanet2@gmail.com'
98     GMAIL_PASSWORD: [REDACTED]
99
100 postgres-academico:
101   image: postgres
102   environment:
103     POSTGRES_PASSWORD: senhapg
104     POSTGRES_USER: usuario_academico
105     POSTGRES_DB: academico
106
107 web-academico:
108   build: ./academico-php-web
109   environment:

```

No password, coloquei uma senha de aplicativos, porque se eu colocar a senha normal do e-mail, o google não deixa usar para mandar e-mails.



Google Conta Pesquisar na Conta do Google

Segurança

Configurações e recomendações para ajudar você a manter sua conta segura

Você tem recomendações de segurança

Há ações recomendadas na Verificação de segurança

Proteger sua conta

Atividades de segurança recentes

Senha de app criada	15 de ago. - São Paulo, Brasil
O login com a verificação em duas etapas foi ativado	15 de ago. - São Paulo, Brasil
Etapa de login adicionada: número de telefone	15 de ago. - São Paulo, Brasil

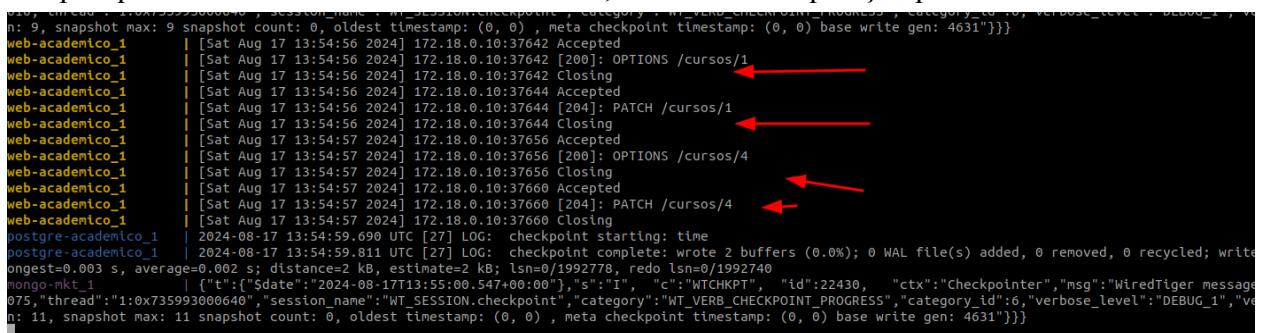
Revisar atividades de segurança

Como você faz login no Google

Mantenha estas informações atualizadas para nunca perder o acesso à sua Conta do Google.

Retornando à aplicação:

Sempre que eu marco um curso como assistido, ele faz uma requisição para atualizar:



```

n: 9, snapshot max: 9 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0, 0) base write gen: 4631"}}
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37642 Accepted
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37642 [200]: OPTIONS /cursos/1
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37642 Closing ←
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37642 Accepted
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37644 [204]: PATCH /cursos/1
web-academico_1 | [Sat Aug 17 13:54:56 2024] 172.18.0.10:37644 Closing ←
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37656 Accepted
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37656 [200]: OPTIONS /cursos/4
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37656 Closing ←
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37660 Accepted
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37660 [204]: PATCH /cursos/4 ←
web-academico_1 | [Sat Aug 17 13:54:57 2024] 172.18.0.10:37660 Closing
postgre-academico_1 | 2024-08-17 13:54:59.690 UTC [27] LOG: checkpoint starting: time
postgre-academico_1 | 2024-08-17 13:54:59.811 UTC [27] LOG: checkpoint complete: wrote 2 buffers (0.0%); 0 WAL file(s) added, 0 removed, 0 recycled; write
ongest=0.003 s, average=0.002 s; distance=2 kB, estimate=2 kB; lsn=0/1992778, redo lsn=0/1992740
mongo-mkt_1 | {"t": {"$date": "2024-08-17T13:55:00.547+00:00"}, "s": "I", "c": "WTCHKPT", "id": 22430, "ctx": "Checkpointer", "msg": "WiredTiger message 075, \"thread\": \"1:0x735993000640\", \"session_name\": \"WT_SESSION.checkpoint\", \"category\": \"WT_VERB_CHECKPOINT_PROGRESS\", \"category_id\": 6, \"verbose_level\": \"DEBUG_1\", \"version\": 11, snapshot max: 11 snapshot count: 0, oldest timestamp: (0, 0) , meta checkpoint timestamp: (0, 0) base write gen: 4631"}}

```

Então dá um PATCH para atualizar o curso como feito.

A screenshot of a web browser window titled 'localhost:4200/cursos'. The page has a blue header bar with the word 'alura' in white. Below the header, there is a sidebar with the title 'Cursos disponíveis' and a list of course icons and names. Each course entry consists of a small icon followed by the course name and a small blue square button to its right. The courses listed are:

- Mais sobre microserviços
- Integração contínua
- 12-Factor Apps
- Docker
- Kubernetes
- Nginx 1
- Nginx 2
- Microserviços: Padrões de projeto
- Entrega contínua

E aí os cursos que já foram feitos, já vão para baixo.

Conhecimento: submódulos no github

06 **Para saber mais: Submódulos**

Há um comando mais simples que permite o `clone` do projeto usando os submódulos. Basta adicionar a opção `--recursive`. Sendo assim, o comando completo fica:

```
git clone --recursive https://github.com/CViniciusSDias/alura-ms.git
```

[COPIAR CÓDIGO](#)

[DISCUTIR NO FÓRUM](#) [PRÓXIMA ATIVIDADE](#)

E sobre docker:

07 **Só um comando**

Para ter o nosso projeto de pé, precisamos apenas de um comando e todos os serviços foram inicializados. Parece mágica, mas é só tecnologia.

Quais tecnologias foram envolvidas no processo de organização dos serviços?

A Git (submódulos).

Alternativa correta! Nós separamos cada projeto em um submódulo do Git.

B Kubernetes.

C Docker (docker-compose).

Alternativa correta! Nós usamos o docker-compose para definir os serviços.

[DISCUTIR NO FÓRUM](#) [PRÓXIMA ATIVIDADE](#)

Conhecimento: envio de e-mails pelo gmail

08 **Para saber mais: E-mail**

No projeto configurado em meu ambiente eu defini meu e-mail e senha para enviar o e-mail de matrícula do usuário através do microsserviço acadêmico.

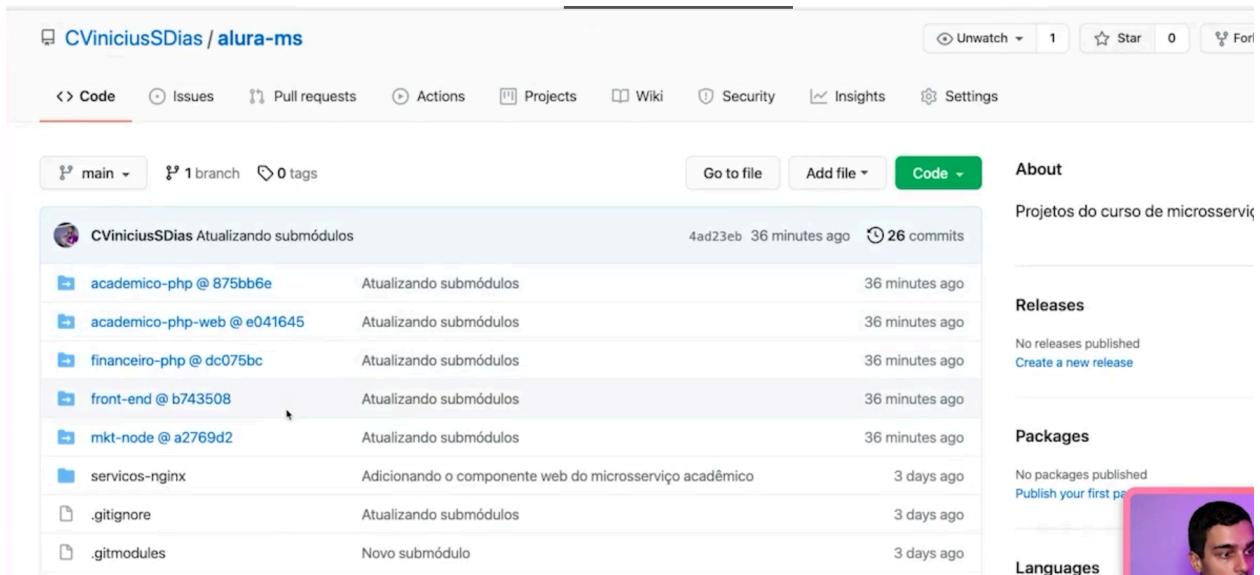
Você vai precisar configurar uma conta do gmail para realizar esse envio. Para isso, defina no docker-compose.yml as variáveis `GMAIL_USER` e `GMAIL_PASSWORD` com os valores corretos.

Para que seu e-mail e senha funcionem, você deve OU desativar autenticação em 2 fatores OU utilizar uma senha de app, conforme é explicado [nesse tópico do Google](#).

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Link: <https://support.google.com/accounts/answer/185833?hl=pt-BR>

A separação do projeto por submódulos foi feita da seguinte forma:

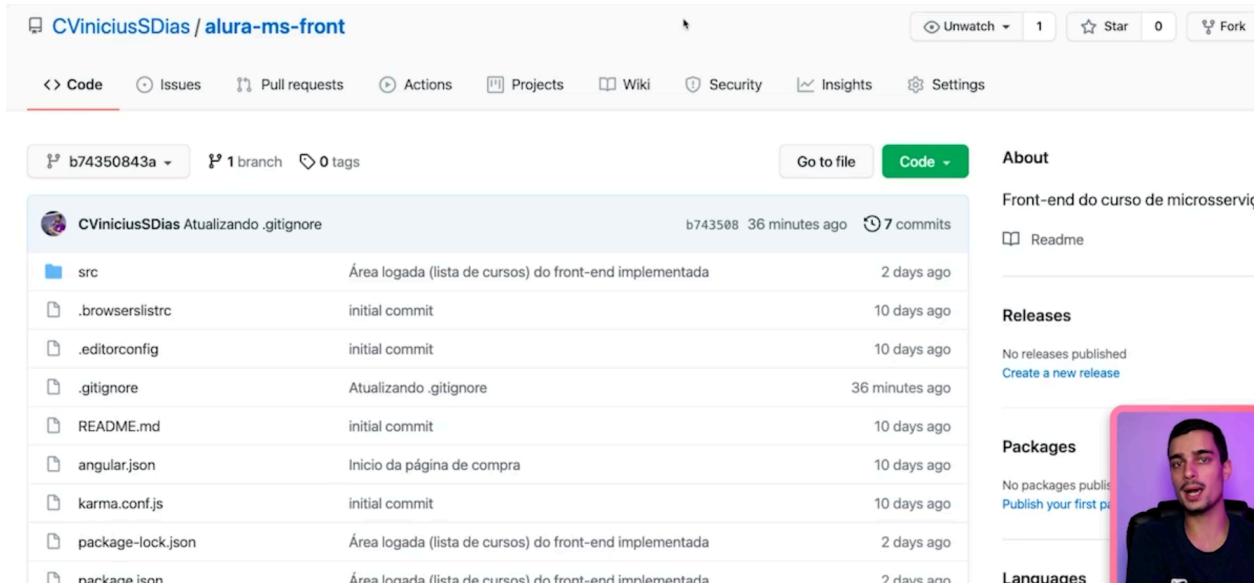


The screenshot shows the GitHub repository page for `CViniciusSDias / alura-ms`. The main repository page displays a list of 26 commits from the user `CViniciusSDias`. The commits are categorized under submodules such as `academico-php`, `academico-php-web`, `financeiro-php`, `front-end`, `mkt-node`, `servicos-nginx`, `.gitignore`, and `.gitmodules`. The commits are dated between 36 minutes ago and 3 days ago. To the right of the commit list, there are sections for **About**, **Releases**, **Packages**, and **Languages**, along with a profile picture of the user.

Ele tem um projeto central chamado “alura-ms”.

Em cada subpasta do alura-ms, ele tem um projeto diferente do git.

Por exemplo, ao clicar em front-end:



The screenshot shows the GitHub repository page for `CViniciusSDias / alura-ms-front`. The repository page displays a list of 7 commits from the user `CViniciusSDias`. The commits are dated between 36 minutes ago and 10 days ago. The commits are associated with files like `src`, `.browserslistrc`, `.editorconfig`, `.gitignore`, `README.md`, `angular.json`, `karma.conf.js`, `package-lock.json`, and `package.json`. To the right of the commit list, there are sections for **About**, **Releases**, **Packages**, and **Languages**, along with a profile picture of the user.

Ele tomou então a decisão de ter um projeto separado para cada microsserviço e um projeto separado também para o front-end.

Cada um dos microservices é um repositório git diferente, é tudo independente.

Isso é chamado de multi-repos ou multi repositórios, onde a arquitetura de microsserviços é organizada em repositórios diferentes.

A alternativa para isso seria o mono repo, onde se teriam todos os projetos no mesmo github, e não apenas uma referência para outro. Então, para ter um projeto, teria que ter todos.

O multi repo parece ser mais eficaz. Mas, por exemplo no Google e Facebook, multi techs, eles tem milhares de microservices. Então, manter esses repositórios diferentes fica bastante complicado e bastante caro também. Então, empresas grandes, que tem milhares de microserviços, eles mantém a estratégia de monorepos.

Para nós, que vamos desenvolver algo pequeno, compensa adotar o multi repo, porque dessa forma, não teremos um repositório bagunçado, com tudo dentro dele.

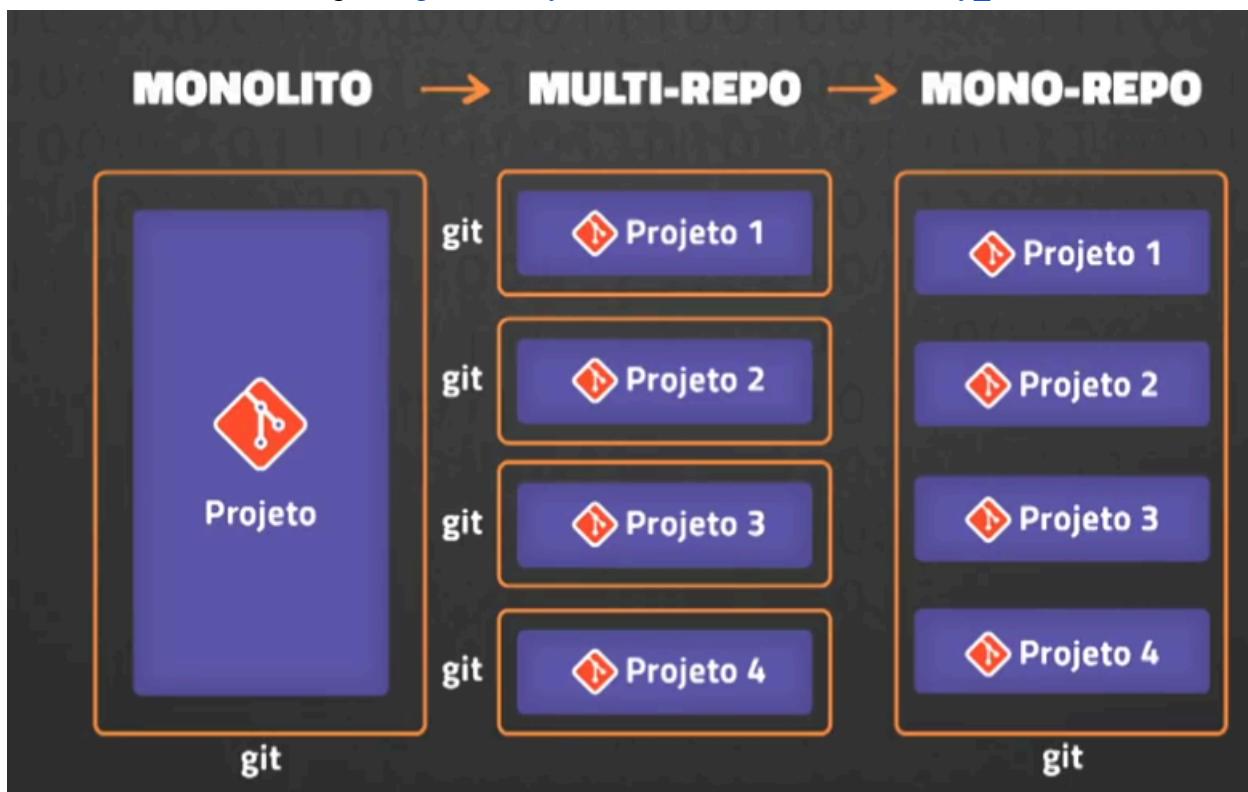
02 **Para saber mais: Monorepo** PRÓXIMA ATIVIDADE

Neste vídeo nós falamos sobre a abordagem de multi vs monorepo. No nosso projeto nós utilizamos multi repo com submódulos. O fato de não termos comunicação síncrona entre os serviços ajuda BASTANTE nessa abordagem.

Se quiser se aprofundar um pouco mais sobre esse conceito e as vantagens e desvantagens de monorepo, você pode conferir aqui:

- [Monorepo \(Por Que Essa Estratégia Funciona em Grandes Empresas?\)](#)

Saber mais sobre mono repo: https://www.youtube.com/watch?v=BbNIuUy_F0w



Vantagens do monorepo:

VANTAGENS

**Ambiente de
Desenvolvimento**



Reutilização de Código



**Gerenciamento de
Dependências**



Gerenciamento de Versões



Desafios:

DESVANTAGENS

CI/CD



Segurança



Equívocos



CI/CD no sentido de ser difícil identificar qual aplicação em si foi mudada para buildar e deployar.

Segurança porque os devs tem acesso a tudo, e não somente ao projeto específico. No Google, utilizam o piper, que é um controle de acessos ao repositório onde a maioria das pessoas tem acesso a tudo, mas algumas pastas específicas, com importantes arquivos de configuração ou informações secretas são de posse e uso somente de algumas pessoas em específico.

Equívoco no sentido de estar na pasta errada e instalar uma dependência de forma errada.

Conhecimento: desvantagem a mais do multi repo

The screenshot shows a mobile application interface for a poll titled "Mais desvantagem" (More disadvantage). The poll has four options:

- A**: Cada equipe precisa apenas do seu repositório para trabalhar. (Each team needs only its own repository to work.)
- B**: Controle de dependência entre projetos. (Control of dependencies between projects.)

Alternativa correta! Se nós tivéssemos comunicação síncrona e dependência entre serviços, teríamos um trabalho a mais para manter isso sob controle. Talvez até um trabalho manual.
- C**: Criação de um novo serviço sem padronização. (Creation of a new service without standardization.)

Alternativa errada! Embora essa seja sim uma desvantagem, ela foi citada no vídeo. :-D

At the bottom, there are buttons for "DISCUTIR NO FÓRUM" (Discuss in the forum) and "PRÓXIMA ATIVIDADE" (Next activity).

Ele optou por uma abordagem mais flexível usando os submódulos, pois ele pode ter vários serviços com linguagens e padronizações diferentes uns dos outros.

Em um projeto real, talvez seria mais interessante ter um monorepo para evitar essa flexibilização e ter uma certa padronização, pois dessa forma, poderíamos ter um bootstrap, ou seja, uma automação para criar um novo serviço, que quando eu fosse criar um novo serviço, ele já criaria um boilerplate para mim, com as dependências em comum que todos precisam, com a estrutura que todos usam, etc. No jeito que ele fez, não dá para fazer isso, pois cada serviço está usando uma linguagem e uma abordagem diferente.

Vamos entender melhor sobre submódulos do git.

Ele usou o submódulo aqui principalmente para ter um repo centralizado, com apenas referências para os outros projetos, para que pudesse subir todos os serviços de uma única vez.

Como funciona?

Quando eu faço um clone de um projeto que tem submódulos, os submódulos em si (as pastinhas dos submódulos) não tem o projeto do submódulo ali... ele tem apenas um ponteiro para algum commit em específico que foi feito por último (geralmente) naquele submódulo. Então, se eu fizer alguma modificação no submódulo e dar um commit, eu tenho que dar uma atualizada no meu repositório central (que tem os links dos submódulos) para apontar para esse novo commit e ficar tudo atualizado.

The screenshot shows a terminal window with the following content:

```
joo@joo-mbp:~/Documentos/WORKSHOP/ALURA/Alura-Ms$ cd alura-ms
joo in alura-ms on ✘ main [!]
→ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   docker-compose.yml

no changes added to commit (use "git add" and/or "git commit -a")
joo@joo-mbp:~/Documentos/WORKSHOP/ALURA/Alura-Ms$ ls -la
total 68
drwxrwxr-x  9 joao  joao 4096 ago 15 08:05 .
drwxrwxr-x  3 joao  joao 4096 ago 15 08:04 ..
drwxrwxr-x  4 joao  joao 4096 ago 15 08:05 academico-php
-rw-rw-r--  1 joao  joao   85 ago 15 08:04 academico-php.sh
drwxrwxr-x 11 joao  joao 4096 ago 15 08:05 academico-php-web
-rw-rw-r--  1 joao  joao 153 ago 15 08:04 academico-php-web.sh
-rw-rw-r--  1 joao  joao 3043 ago 15 08:05 docker-compose.yml
drwxrwxr-x  4 joao  joao 4096 ago 15 08:05 financeiro-php
-rw-rw-r--  1 joao  joao   83 ago 15 08:04 financeiro-php.sh
drwxrwxr-x  4 joao  joao 4096 ago 15 08:05 front-end
-rw-rw-r--  1 joao  joao 110 ago 15 08:04 front-end.sh
drwxrwxr-x  9 joao  joao 4096 ago 17 11:47 .git
-rw-rw-r--  1 joao  joao    6 ago 15 08:04 .gitignore
-rw-rw-r--  1 joao  joao 574 ago 15 08:04 .gitmodules
drwxrwxr-x  4 joao  joao 4096 ago 15 08:05 mkt-node
-rw-rw-r--  1 joao  joao   80 ago 15 08:04 mkt-node.sh
drwxrwxr-x  2 joao  joao 4096 ago 15 08:04 servicos-nginx
joo@joo-mbp:~/Documentos/WORKSHOP/ALURA/Alura-Ms$ cd academico-php
joo in academico-php on ✘ main
→ git status
HEAD detached at 53c0cc7 ←
nothing to commit, working tree clean
joo in academico-php on ✘ main
→
```

A red box highlights the command `cd academico-php` and its output. A red arrow points to the line `HEAD detached at 53c0cc7`.

É só observar a imagem acima, quando estou no repo central (alura-ms), eu estou na branch main e foi alterado o docker-compose.yml.

Quando estou em um submódulo, o academico-php, por exemplo, eu dou um git status e tenho um apontamento para um commit em específico (último commit daquele submódulo).

Exemplo:

```
vinicius.dias@A5000070 alura-ms % cd front-end
vinicius.dias@A5000070 front-end % git status
HEAD detached at b743508
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   package-lock.json ←

no changes added to commit (use "git add" and/or "git commit -a")
vinicius.dias@A5000070 front-end %
```

Ele entrou no de front-end que tem uma alteração no package-lock.json, provavelmente porque atualizou alguma dependência (npm install).

```
vinicius.dias@A5000070 front-end % git switch main
```

Sempre que entro no repositório do submódulo, eu dou um switch ou checkout para a main, para entrar realmente no código do repositório, porque até então estava só apontando para o último commit.

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   package-lock.json

no changes added to commit (use "git add" and/or "git commit -a")
vinicius.dias@A5000070 front-end % git commit -am "Atualizando package-lock.json"
[main 78dcad7] Atualizando package-lock.json
 1 file changed, 1 insertion(+)
vinicius.dias@A5000070 front-end % git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 385 bytes | 385.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:CViniciusSDias/alura-ms-front.git
  b743508..78dcad7 main -> main
vinicius.dias@A5000070 front-end %
```



Então ele deu um commit e um push nesse submódulo do front-end. Logo, no GitHub, o projeto do front-end foi atualizado e tem uma nova versão de commit mais recente.

Porém, o repositório principal, o repo central (alura-ms), ainda não tem essa atualização, porque esse repo central estava apontando para outro commit, um commit anterior.

```

commit 78dcad72721b807a6256647585782fe4cc0d1a05 (HEAD -> main, origin/main, origin/H
Author: Vinicius Dias <carlosv775@gmail.com>
Date: Sat Aug 14 16:43:17 2021 -0300

    Atualizando package-lock.json
    ↓
commit b74350843ab59ee95756aa58ad6c1abbe61a3589
Author: Vinicius Dias <carlosv775@gmail.com>
Date: Sat Aug 14 15:54:22 2021 -0300

    Atualizando .gitignore

commit 482e1fd2cf81e75c24fb8db44a2b6aae36bbca14
Author: Vinicius Dias <vinicius.dias@achievers.com>
Date: Thu Aug 12 23:01:34 2021 -0300

```

No repo central, o front está apontando para o b74, mas agora o meu front está no 78d.

Logo, temos que fazer essa atualização.

Quando volto para o repo central e dou um git status:

```

vinicius.dias@A5000070 alura-ms % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   front-end (new commits)

no changes added to commit (use "git add" and/or "git commit -a")
vinicius.dias@A5000070 alura-ms %

```

Ele mostra que tem atualizações no submódulo de front-end.

Então, tenho que dar um add no submódulo, um commit e um push:

```

vinicius.dias@A5000070 alura-ms % git add front-end ←
vinicius.dias@A5000070 alura-ms % git commit -m "Atualizando submódulo de front-end" ←
[main d9b2641] Atualizando submódulo de front-end
  1 file changed, 1 insertion(+), 1 deletion(-)
vinicius.dias@A5000070 alura-ms % git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 278 bytes | 278.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:CViniciusSDias/alura-ms.git
  4ad23eb..d9b2641  main -> main
vinicius.dias@A5000070 alura-ms %

```



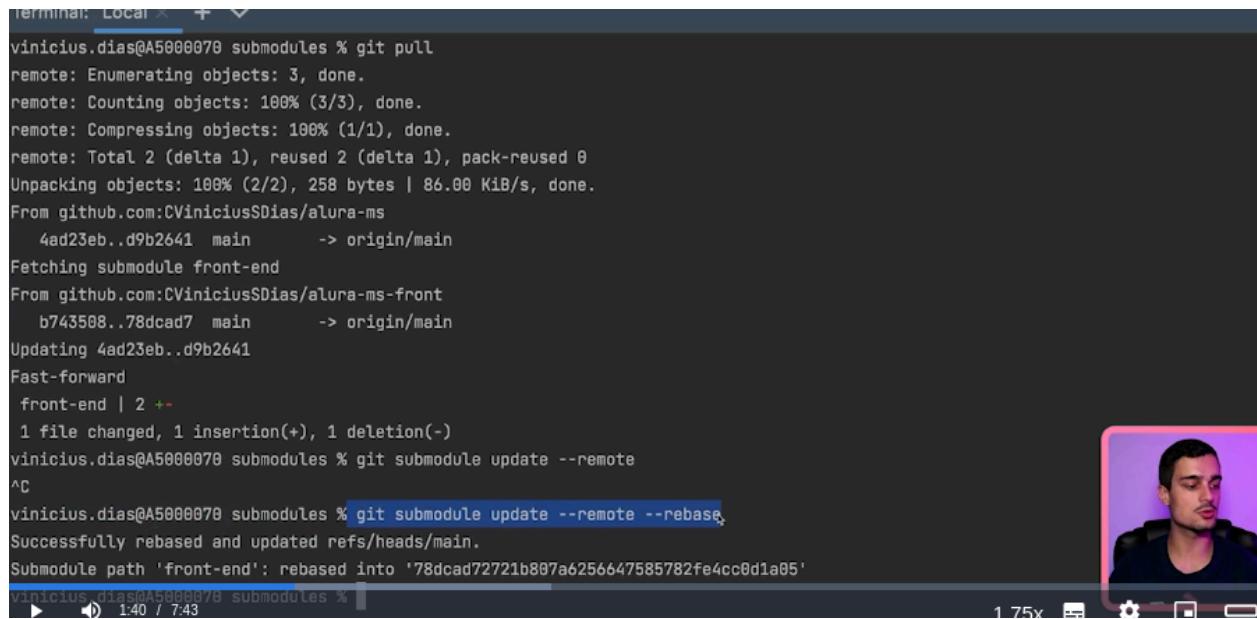
Dessa forma, o meu repo central, vai apontar corretamente para o último commit do meu submódulo de front-end, para a última versão dele.

OBS: eu sempre faço o switch/checkout para a main ao entrar em um submódulo porque quando eu clonei um repositório que tem submódulos, os submódulos vem apenas com uma referência para o último commit e se eu quiser fazer novos commits, tenho que fazer isso dentro de uma branch. Para isso, tenho que fazer o git switch/checkout para a branch.

Mas quando o time de dev vai trabalhar em cima de algum projeto, de algum serviço em específico, é ideal que trabalhem na própria pasta dele, e não na pasta central, no repo central (alura-ms). Até podem trabalhar no alura-ms, mas não é ideal, tem que ficar lembrando se sempre atualizar o submódulo como feito acima (git add submódulo e git commit e git push).

Mais sobre submódulos: <https://www.atlassian.com/git/tutorials/git-submodule>

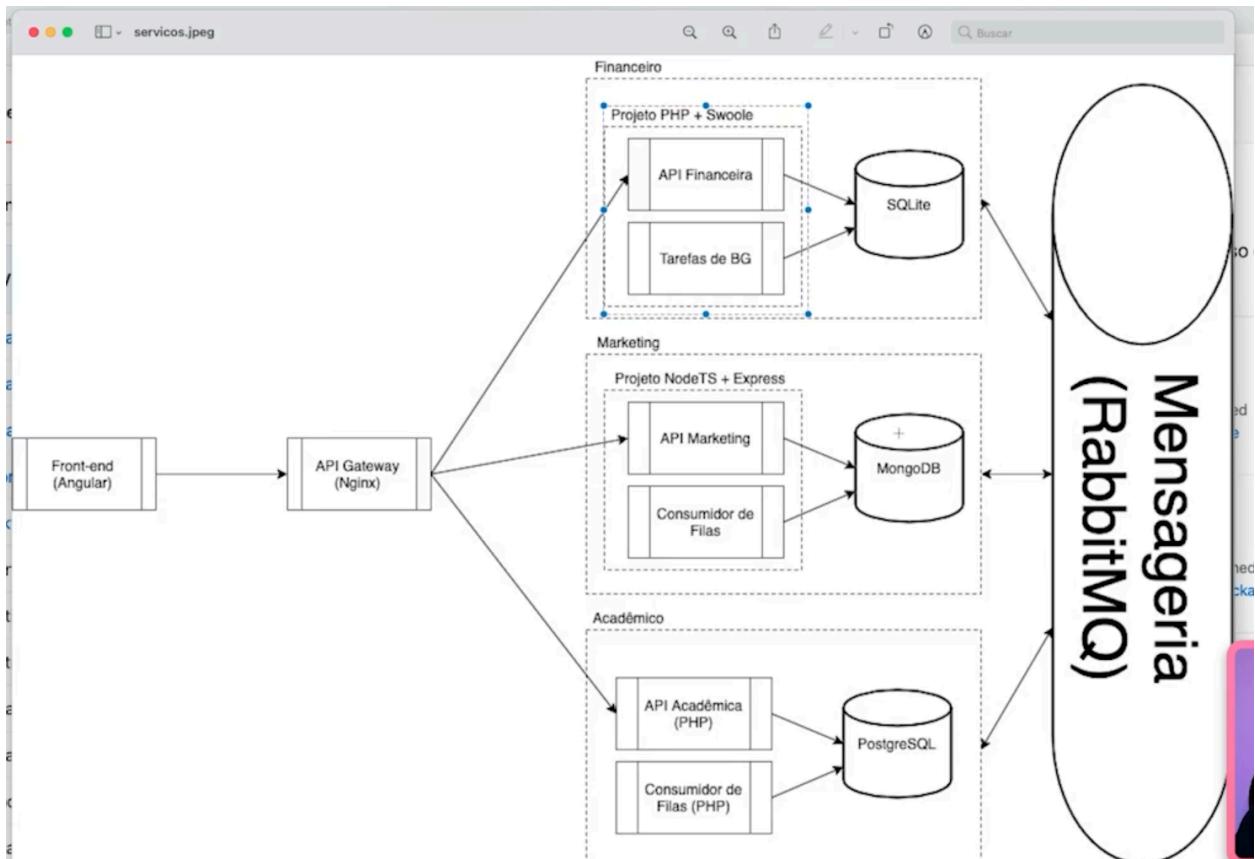
Por fim, para inserir essas modificações dentro dos submódulos nas minhas pastas dentro do meu computador (localmente), ou seja, estou com o IntelliJ aberto e quero atualizar tudo... eu abro o projeto principal (alura-ms), dou um git pull, e ele vai me mostrar que mudanças no front-end:



```
Terminal: Local
vinicius.dias@A5000070 submodules % git pull
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 2 (delta 1), reused 2 (delta 1), pack-reused 0
Unpacking objects: 100% (2/2), 258 bytes | 86.00 KiB/s, done.
From github.com:CViniciusSDias/alura-ms
  4ad23eb..d9b2641  main      -> origin/main
Fetching submodule front-end
From github.com:CViniciusSDias/alura-ms-front
  b743508..78dcad7  main      -> origin/main
Updating 4ad23eb..d9b2641
Fast-forward
 front-end | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
vinicius.dias@A5000070 submodules % git submodule update --remote
^C
vinicius.dias@A5000070 submodules % git submodule update --remote --rebase
Successfully rebased and updated refs/heads/main.
Submodule path 'front-end': rebased into '78dcad72721b807a6256647585782fe4cc0d1a05'
vinicius.dias@A5000070 submodules %
```

Então, para atualizar tudo, e não precisar ficar entrando em submódulo por submódulo, eu dou um git submodule update --remote --rebase, e dessa forma eu consigo atualizar tudo, todos os submódulos que tiveram alterações mapeadas.

Vamos rever o desenho para falar sobre a Infraestrutura:



Aqui tem um ponto importante: não podemos ter microsserviços compartilhando o mesmo banco de dados. Se algum microsserviço compartilha banco de dados com outro, isso é um péssimo sinal.

E outra coisa, daria para melhorar essa infra ai... disponibilidade -> criar réplicas, outras instâncias dos bancos de dados, para o caso de quando eles caírem, criar réplicas das APIs também e inserir um load balancer na frente, para se alguma cair ter outra, etc.

A infraestrutura de cada microsserviço precisa ser independente. Todo microsserviço precisa ter sua infraestrutura separada de outro microsserviço. Cada um terá sua particularidade.

Isso é importante porque no nosso caso, temos 3 microsserviços, com infras totalmente diferentes, com linguagens diferentes, bancos de dados diferentes e para subir todos eles de uma só vez, temos um docker-compose.yml, que retrata todas as infras de cada microsserviço e sobe tudo isso junto.

Por isso a containerização é ideal no cenário de microsserviços. Porque eu poderia virtualizá-los também, ou seja, criar uma máquina virtual para cada um, mas ficaria um pouco mais caro e mais complexo ou poderia criar uma IaC (infra como código) e mandar instalar tudo na minha máquina de cada microsserviço (a infra que cada um precisa), mas isso também ficaria muito caro e exigiria bastante da máquina.

No container, eu consigo ter vários mundos, várias infras instaladas e conviver os microsserviços juntos, cada um com sua particularidade.

Quando eu falo da infra de cada microsserviço eu digo em respeito ao banco de dados, aplicativos que precisam instalados, dependências, libs, etc.

The screenshot shows a digital course interface with a dark theme. At the top, there's a navigation bar with three dots on the left, the text "02 Escalando serviços" in the center, and a blue button on the right labeled "PRÓXIMA ATIVIDADE". Below the navigation bar, there's a text area containing the following content:

Neste vídeo nós revisamos nossa infraestrutura e um desafio ficou: Replicar e escalar serviços.

Qual técnica nós poderíamos utilizar para escalar nossos serviços?

Three options are listed in a grid:

- A** Orquestrador de container.
Alternativa correta! Docker Swarm ou Kubernetes são ótimas ferramentas para esse trabalho.
- B** Ambiente em Cloud.
- C** Docker.

At the bottom of the screen are two buttons: "DISCUTIR NO FÓRUM" on the left and "PRÓXIMA ATIVIDADE" on the right.

Falando sobre a independência de microsserviços, por exemplo o mkt-node - é um microsserviço que precisa do node instalado, ou seja, dentro dele eu vou ter um container com todos os requisitos que ele precisa:

```

version: '3.7'
services:
  web-mkt:
    image: node
    volumes:
      - ./app:/app
    working_dir: /app
    command: npm start
    environment:
      SERVER_PORT: 3000
      MONGO_URL: "mongodb://mkt-usario:mkt-senha@mongo-mkt:27017/mkt?authSource=admin"
      RABBITMQ_HOST: 'rabbitmq'
      RABBITMQ_PORT: 5672
      RABBITMQ_USERNAME: guest
      RABBITMQ_PASSWORD: guest
    ports:
      - 3000:3000
    depends_on:
      - rabbitmq
      - mongo-mkt

  mongo-mkt:
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: mkt-usario
      MONGO_INITDB_ROOT_PASSWORD: mkt-senha
      MONGO_INITDB_DATABASE: mkt

  rabbitmq:
    image: rabbitmq
    ports:
      - 5672:5672

```

Então, se uma equipe for trabalhar somente com o microsserviço de marketing, já teremos ali toda a infraestrutura necessária: tenho o serviço web (minha API), tenho o serviço de banco de dados (MongoDB) e tenho o serviço de mensageria (rabbitmq).

Já tenho toda a infraestrutura definida no próprio serviço.

No caso do microsserviço financeiro, além do docker-compose.yml, ele tem um docker file para informar as infos de PHP que ele precisa, o swoole, etc. Ou seja, ele já tem tudo o que precisa para ser trabalhado:

```

version: '3.7'
services:
  rabbitmq:
    container_name: rabbitmq
    image: rabbitmq
    ports:
      - 5672:5672
    php:
      build: .
      command: php index.php
      ports:
        - 9001:9001
      volumes:
        - ./financeiro-php/app:/app
      working_dir: /app
      depends_on:
        - rabbitmq

```

```

FROM php:8.0.14
RUN apt-get update && apt-get install -y libzip-dev libsasl2-dev && docker-php-ext-install zip
RUN docker-php-ext-install sockets
RUN echo 'yes\nno\nno\nno\nno\nno\nno\nyes\n' | pecl install swoole-5.1.3
RUN docker-php-ext-enable swoole
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin --filename=composer

```

Ele já tem tudo, tanto as infos de infra quanto as infos de instalação do PHP no arquivo DockerFile.

Ou seja, todos os meus microsserviços, se eu quiser trabalhar separadamente com cada um deles eu consigo porque eu já tenho um docker-compose.yml em cada um e uns DockerFiles em

alguns com tudo o que eles precisam e consigo subir eles tranquilamente, sem depender de outros.

The screenshot shows a digital learning interface. At the top, there's a navigation bar with three dots on the left, the text "04 Por que isolar?", and a blue button on the right labeled "PRÓXIMA ATIVIDADE". Below this is a dark grey header area with three vertical dots on the right side. The main content area has a white background. It starts with a text block: "Nós vimos neste vídeo que cada serviço é independente e contém todas as informações necessárias para funcionar de forma isolada." followed by a question: "Se a aplicação precisa de todos os serviços, por que eu precisaria conseguir executar um serviço de forma isolada?". Three options are listed below, each with a colored square (A is green, B is blue, C is light blue) and a letter: A) "Dessa forma cada equipe pode trabalhar em um serviço de forma independente." B) "Dessa forma podemos ter uma melhor performance em cada serviço." C) "Dessa forma podemos ter métricas de cada serviço." Option A is highlighted with a green checkmark icon and a callout box stating "Alternativa correta! Se nós não isolássemos os serviços seria muito difícil cada equipe trabalhar apenas em seu respectivo serviço." At the bottom, there are two blue navigation buttons: "Anterior" and "Próximo".

Nós vimos neste vídeo que cada serviço é independente e contém todas as informações necessárias para funcionar de forma isolada.

Se a aplicação precisa de todos os serviços, por que eu precisaria conseguir executar um serviço de forma isolada?

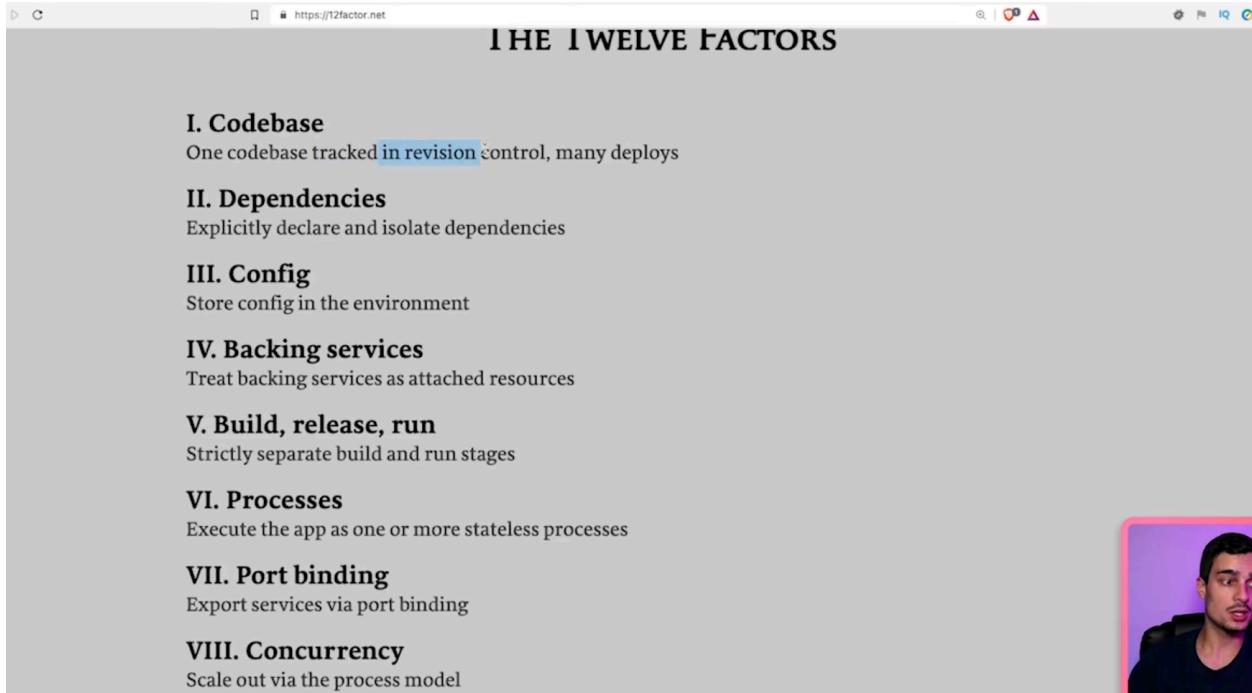
A Dessa forma cada equipe pode trabalhar em um serviço de forma independente.

B Dessa forma podemos ter uma melhor performance em cada serviço.

C Dessa forma podemos ter métricas de cada serviço.

Alternativa correta! Se nós não isolássemos os serviços seria muito difícil cada equipe trabalhar apenas em seu respectivo serviço.

OBS: temos esse 12 fatores de código que é bom seguir sempre:



The screenshot shows a web browser displaying the 'THE TWELVE FACTORS' page from <https://12factor.net>. The page lists twelve factors, each with a bold title and a descriptive subtitle. The factors are: I. Codebase, II. Dependencies, III. Config, IV. Backing services, V. Build, release, run, VI. Processes, VII. Port binding, and VIII. Concurrency. To the right of the list, there is a small portrait photo of a man with dark hair and a beard, wearing a black shirt, set against a purple background.

- I. Codebase**
One codebase tracked in revision control, many deploys
- II. Dependencies**
Explicitly declare and isolate dependencies
- III. Config**
Store config in the environment
- IV. Backing services**
Treat backing services as attached resources
- V. Build, release, run**
Strictly separate build and run stages
- VI. Processes**
Execute the app as one or more stateless processes
- VII. Port binding**
Export services via port binding
- VIII. Concurrency**
Scale out via the process model

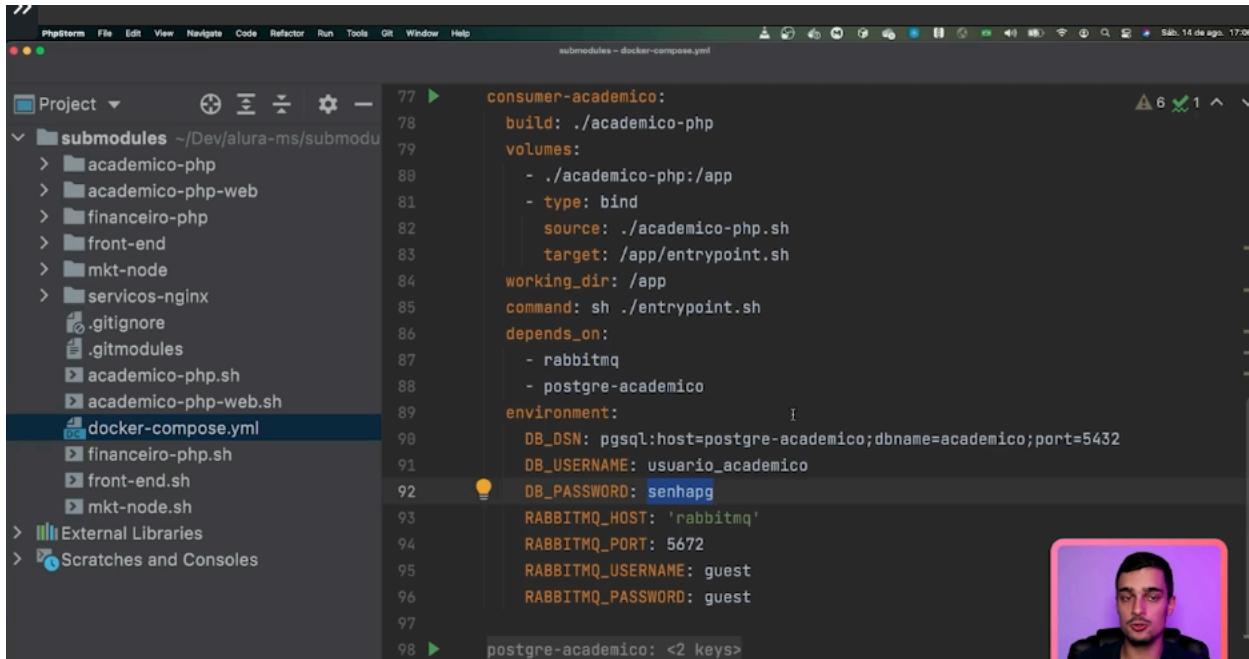
Site: <https://12factor.net/>

Vamos ver quais estamos seguindo.

1 - sim, pois estamos trabalhando com git.

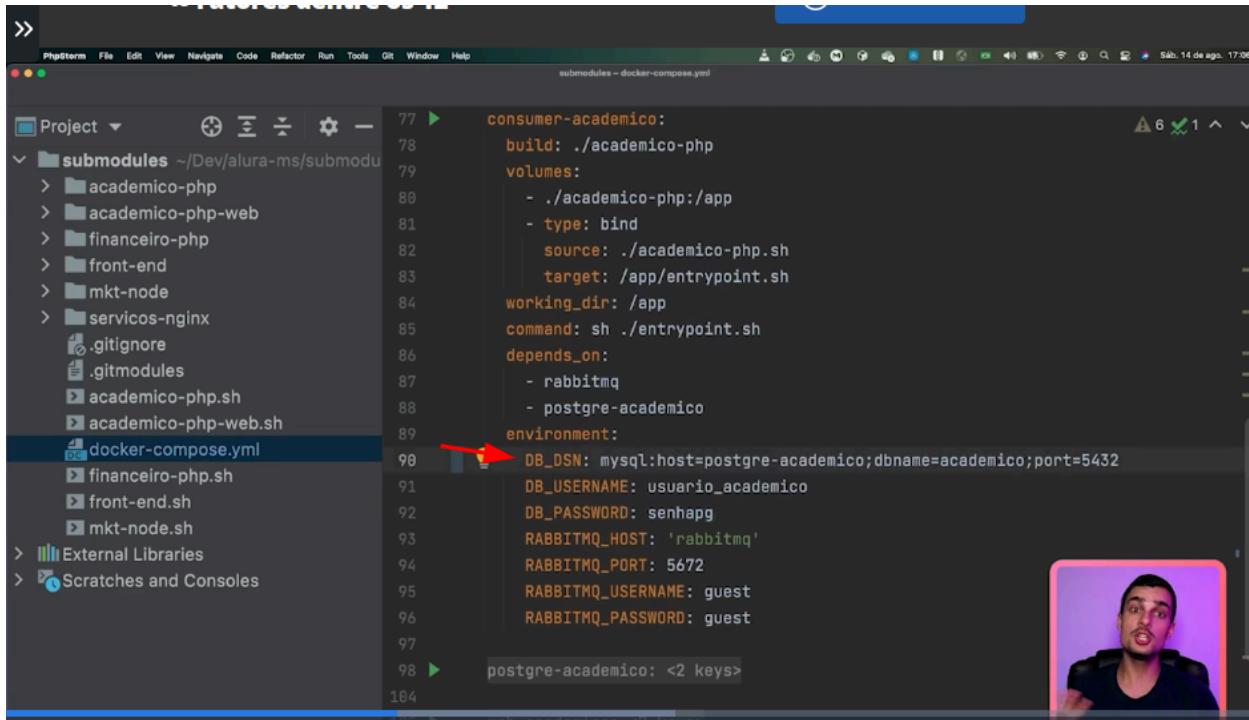
2 - sim, estamos trabalhando com gerenciadores de dependências - em node trabalhamos com npm, em php estão utilizando composer, etc.

3 - sim, temos as configs no ambiente... se eu quiser mudar a senha do banco por exemplo, é só ir no docker-compose.yml e alterar (e posso fazer isso em todos os ambientes) - o meu código não precisa ser alterado, basta eu definir a configuração no arquivo e no ambiente:



```
consumer-academico:
  build: ./academico-php
  volumes:
    - ./academico-php:/app
    - type: bind
      source: ./academico-php.sh
      target: /app/entrypoint.sh
  working_dir: /app
  command: sh ./entrypoint.sh
  depends_on:
    - rabbitmq
    - postgre-academico
  environment:
    DB_DSN: pgsql:host=postgresql;dbname=academico;port=5432
    DB_USERNAME: usuario_academico
    DB_PASSWORD: senhapg
    RABBITMQ_HOST: 'rabbitmq'
    RABBITMQ_PORT: 5672
    RABBITMQ_USERNAME: guest
    RABBITMQ_PASSWORD: guest
```

4 - tratar os serviços de apoio como recursos anexados - sim também, pois podemos por exemplo trocar o banco de postgres para MySql sem modificar o código, basta modificar no arquivo docker-compose.yml:



```
consumer-academico:
  build: ./academico-php
  volumes:
    - ./academico-php:/app
    - type: bind
      source: ./academico-php.sh
      target: /app/entrypoint.sh
  working_dir: /app
  command: sh ./entrypoint.sh
  depends_on:
    - rabbitmq
    - postgre-academico
  environment:
    DB_DSN: mysql:host=postgresql;dbname=academico;port=5432
    DB_USERNAME: usuario_academico
    DB_PASSWORD: senhapg
    RABBITMQ_HOST: 'rabbitmq'
    RABBITMQ_PORT: 5672
    RABBITMQ_USERNAME: guest
    RABBITMQ_PASSWORD: guest
```

5 - ainda não estamos fazendo.

6 - processos - sim, temos! Se eu trocar meu microsserviço de marketing por um de java, eu posso fazer isso tranquilamente, meus outros serviços nem vão saber.

7 - port binding - já estamos expondo cada um dos nossos serviços de forma isolada, através de uma porta e quem se conecta no nosso serviço não precisa saber o que está sendo feito.

8 - concorrência - não estamos fazendo, pois cada serviço é único, não tem muito como distribuir a carga entre eles.

9 - os nossos serviços estão prontos para serem jogados fora e subir novamente! Então, estamos atendendo (só dar um docker-compose down ou docker-compose down serviço para derrubar todos ou um específico). Mas no projeto que ele nos deu, o código não é resiliente para isso, se o RabbitMQ estiver fora por exemplo, nenhum serviço funciona.

10 - paridade do ambiente de PRD - estamos usando conteinerização, então estamos prontos para aderir a isso.

11 - logs - não estamos lidando com logs, é um ponto de melhoria.

12 - mais ou menos - pois temos um processo que roda no plano de fundo para processar pagamentos, por exemplo.

Quanto mais fatores aderirmos, mais robusto nosso serviço é.

Vamos conversar sobre build da nossa aplicação.

Podemos fazer algumas etapas no processo de build:

1 - verificar se não tem erros de sintaxe;

2 - verificar se está dentro do estilo de código necessário para a minha linguagem (ex: java - legal abrir as chaves dos métodos na mesma linha - verifica isso).

3 - análise de código, análise de software estática - para garantir que não tem bugs, sem precisar rodar a aplicação - testes unitários.

No nosso, vamos partir do pressuposto que ele vai analisar apenas o estilo de código, ou seja, algo bem simples, mas que vai precisar passar por uma etapa antes de fazer o build de fato.

Temos algumas opções para termos uma pipeline, um processo de build:

1 - Jenkins - conseguimos até ter um servidor Jenkins on-premises.

2 - Travis CI - aqui temos servidor de build na cloud, e não on-premises. Por exemplo, o GitHub tem uma integração com ele que pode notificá-lo quando alguma alteração é realizada e ele roda ali o CI/CD.

3 - GitHub Actions - feature recente do GitHub - automatizar processos, pipeline, flows.

Vamos usar o GitHub Actions.

Essas ferramentas são a mesma coisa do Azure DevOps.

Vimos neste vídeo 3 ferramentas: Jenkins, Travis CI e GitHub Actions. Acredito que as vantagens do Jenkins (ser usado em infra on-premise) e do GitHub Actions (grátis e integrado com GitHub) ficaram claras.

Qual a vantagem do Travis CI?

A

Poder ser instalado em qualquer lugar.



Poder usar em outro servidor (como Bitbucket ou GitLab) sem custo.

Alternativa correta! Para projetos open source fora do GitHub podemos usar o Travis CI sem nenhum custo nem preocupação com infra on-premise.

C

Uso gratuito de determinadas funções.

Alternativa errada! O GitHub actions possui praticamente as mesmas funcionalidades grátis.

No GitHub Actions, ele tem alguns templates já prontos de coisas que posso usar para fazer essas verificações na etapa de build e até um marketplace com várias ações já pré configuradas e prontas para cada tipo de linguagem (java, php, etc.).

Por exemplo, ele fez uma do php:

The screenshot shows the GitHub Actions setup interface. At the top, there's a navigation bar with links like 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below that, the repository name 'CViniciusSDias / alura-ms-academico-php' is displayed. The 'Actions' tab is selected and highlighted with a red box. Underneath, a 'PHP' workflow template by GitHub Actions is shown, also highlighted with a red box. To the right of the workflow, there's a video thumbnail of a person speaking. The overall heading is 'Get started with GitHub Actions'.

O GitHub Actions já até traz umas actions pré definidas ai.

Quanto eu apertei ali em set up this workflow:

The screenshot shows the GitHub Actions workflow editor for the file '.github/workflows/php.yml'. The code editor displays the following YAML configuration:

```
1 name: PHP Composer
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10   build:
11     runs-on: ubuntu-latest
12
13     steps:
14       - uses: actions/checkout@v2
15
16       - name: Validate composer.json and composer.lock
17         run: composer validate --strict
```

To the right of the code editor, there's a 'Marketplace' sidebar with a search bar and several recommended actions:

- Setup Node.js environment (1.3k)
- Setup Go environment
- Setup Java JDK

A video thumbnail of the speaker is visible on the right side of the editor.

Ele já criou o workflow e já trouxe algumas coisas pré configuradas.

Todas as ações do GitHub Actions devem ficar dentro da pasta .github/workflows. E aqui dentro eu sempre uso arquivos yml.

The screenshot shows the GitHub Actions workflow editor for a repository named 'alura-ms-academico-php/.github/workflows/'. The file is named 'php.yml' and is set to run in the 'main' branch. The code defines a job named 'PHP Code Sniffer' that triggers on pushes or pull requests to the main branch, and runs on an Ubuntu 20.04 LTS machine. It includes steps for checking out the code and validating composer.json and composer.lock files.

```

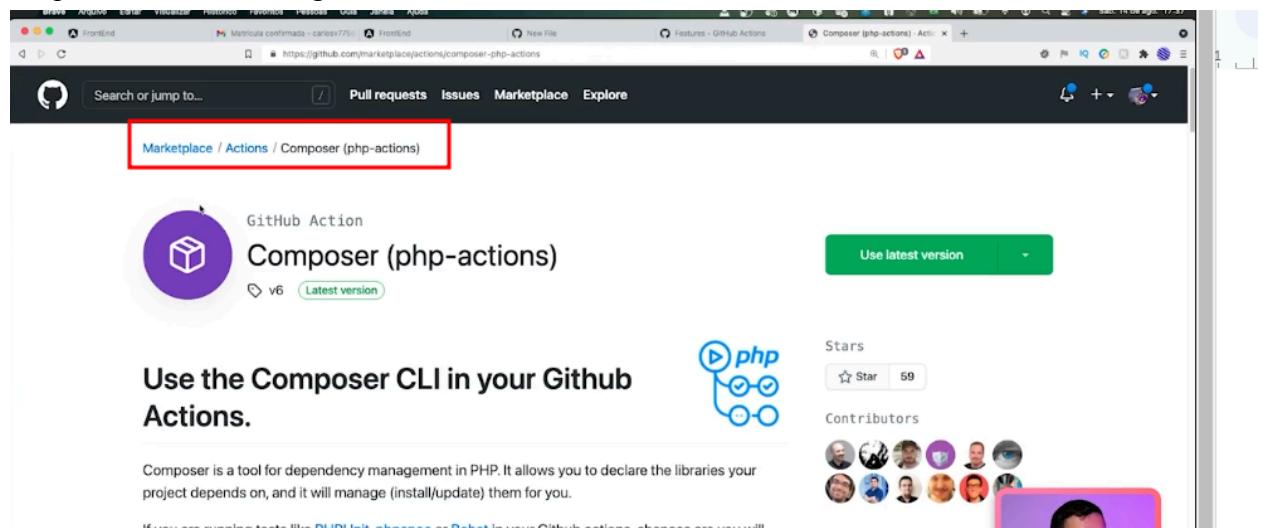
1 name: PHP Code Sniffer
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10   build:
11
12     runs-on: ubuntu-latest
13
14     steps:
15       - uses: actions/checkout@v2
16
17       - name: Validate composer.json and composer.lock
18         run: composer validate --strict

```

Até agora ele renomeou para PHP Code Sniffer, manteve os triggers ali para quando tiver um push ou PR para a branch main.

Vai rodar um job apenas e ele vai rodar no linux... posso ter vários jobs em que um roda no linux, outro no windows, etc.

A primeira etapa é dar o checkout, ou seja, ele vai pegar o nosso código e mandar para a máquina onde esse código irá rodar.



Aqui ele tem um marketplace de actions do GitHub Actions... então, posso pegar várias coisas já prontas e implementar na minha rotina.

CviniciusSDias / alura-ms-academico-php

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

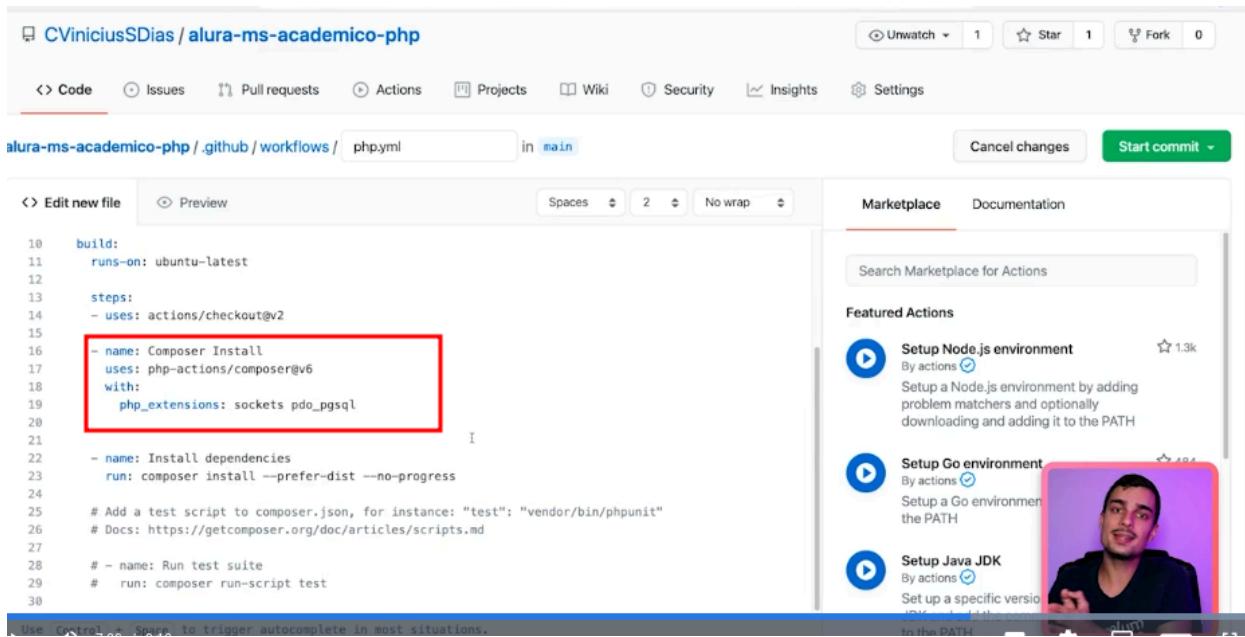
alura-ms-academico-php/.github/workflows/php.yml in main

Cancel changes Start commit

<> Edit new file Preview Spaces 2 No wrap Marketplace Documentation

```
10 build:
11   runs-on: ubuntu-latest
12
13 steps:
14   - uses: actions/checkout@v2
15
16   - name: Composer Install
17     uses: php-actions/composer@v6
18     with:
19       php_extensions: sockets pdo_pgsql
20
21   - name: Install dependencies
22     run: composer install --prefer-dist --no-progress
23
24 # Add a test script to composer.json, for instance: "test": "vendor/bin/phpunit"
25 # Docs: https://getcomposer.org/doc/articles/scripts.md
26
27 # - name: Run test suite
28 #   run: composer run-script test
29
30
```

Use Control + Space to trigger autocomplete in most situations.



Pegou isso do marketplace e até deu uma incrementada... isso é específico do PHP, mas ele informou nesse trecho que quando for dar o composer, precisa daquelas extensões instaladas (sockets e postgreSql).

CviniciusSDias / alura-ms-academico-php

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

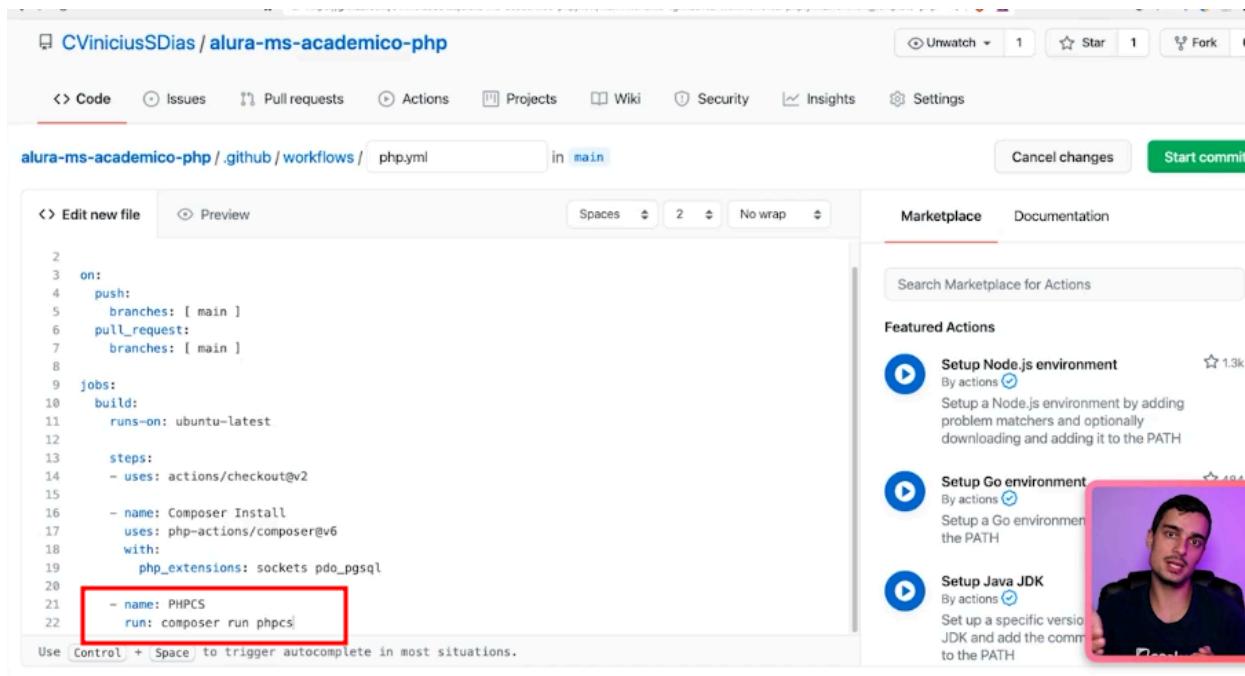
alura-ms-academico-php/.github/workflows/php.yml in main

Cancel changes Start commit

<> Edit new file Preview Spaces 2 No wrap Marketplace Documentation

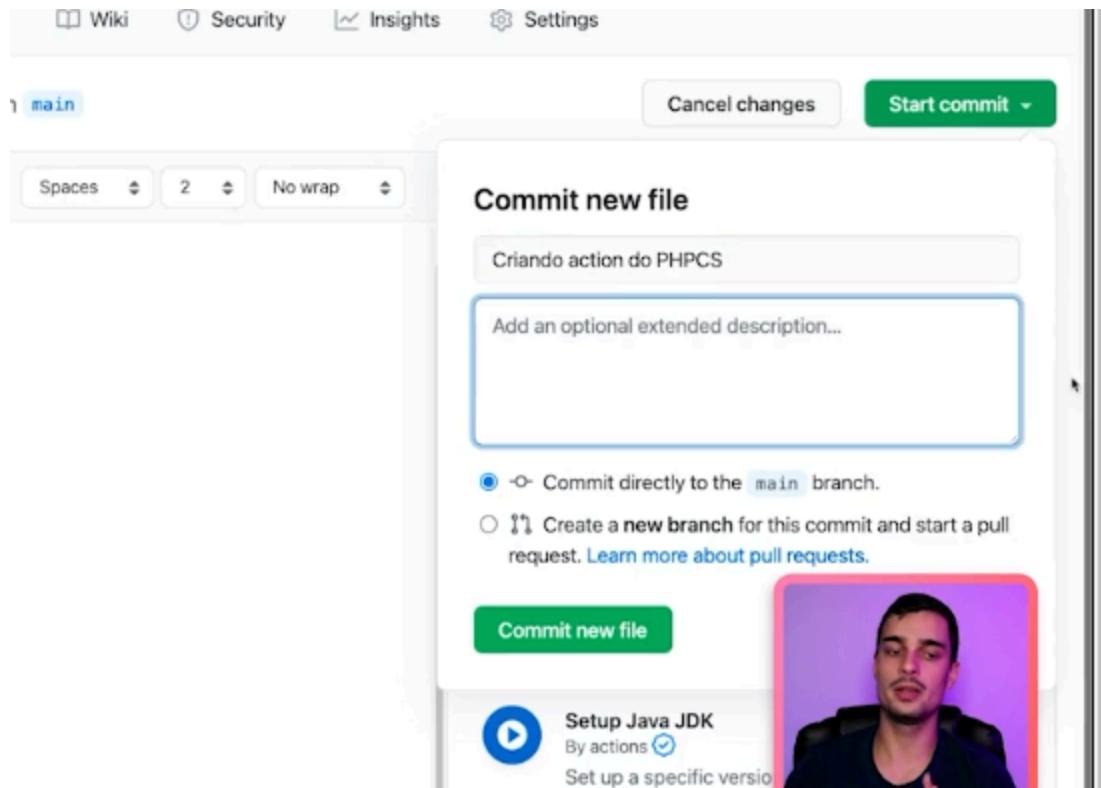
```
2
3 on:
4   push:
5     branches: [ main ]
6   pull_request:
7     branches: [ main ]
8
9 jobs:
10   build:
11     runs-on: ubuntu-latest
12
13   steps:
14     - uses: actions/checkout@v2
15
16     - name: Composer Install
17       uses: php-actions/composer@v6
18       with:
19         php_extensions: sockets pdo_pgsql
20
21     - name: PHPCS
22       run: composer run phpcs
23
24
```

Use Control + Space to trigger autocomplete in most situations.



Aqui ele vai verificar o código, se está tudo certo (formatação, etc.). Esse comando pode ser rodado no terminal, ele só inseriu aqui.

Agora vai dar o commit da action:



Quando eu dei o commit, ele já disparou a action:

The screenshot shows the GitHub Actions page for the repository 'CVinicioSDias / alura-ms-academico-php'. The 'Actions' tab is highlighted with a red box. Below it, the 'All workflows' section is shown, with a blue box around the 'All workflows' button. A workflow named 'PHP Code Sniffer' is listed. The 'All workflow runs' section shows one run, which is highlighted with a red box. The run details show a green checkmark next to 'Criando action do PHPCS', indicating it was successful. The run was triggered by a 'PHP Code Sniffer #1: Commit 7e5aeecc pushed by CVinicioSDias' on the 'main' branch, 1 minute ago. A small video thumbnail of the same person is visible on the right.

Isso porque eu falei que quando tivesse um push na main, para disparar.
Só tem um job (que é o de build):

The screenshot shows a GitHub Actions build log for a repository named 'CVinicioSDias / alura-ms-academico-php'. The build status is green, indicating success. The build name is 'build' and it succeeded 1 minute ago in 45s. The log details the execution of several actions:

- Set up job
- Run actions/checkout@v2
- Composer Install
- PHPCS
- Post Run actions/checkout@v2
- Complete job

A small video player window in the bottom right corner shows a man speaking, likely the developer or instructor.

E dentro do job, tem alguns stages (set up - levantar a máquina com os requisitos necessários, checkout - clonar nosso projeto, composer install - onde já instalou as dependências que eu especifiquei:

The screenshot shows a GitHub Actions build log for the same repository. The build name is 'build' and it succeeded 1 minute ago in 45s. The log details the execution of the 'Composer Install' stage:

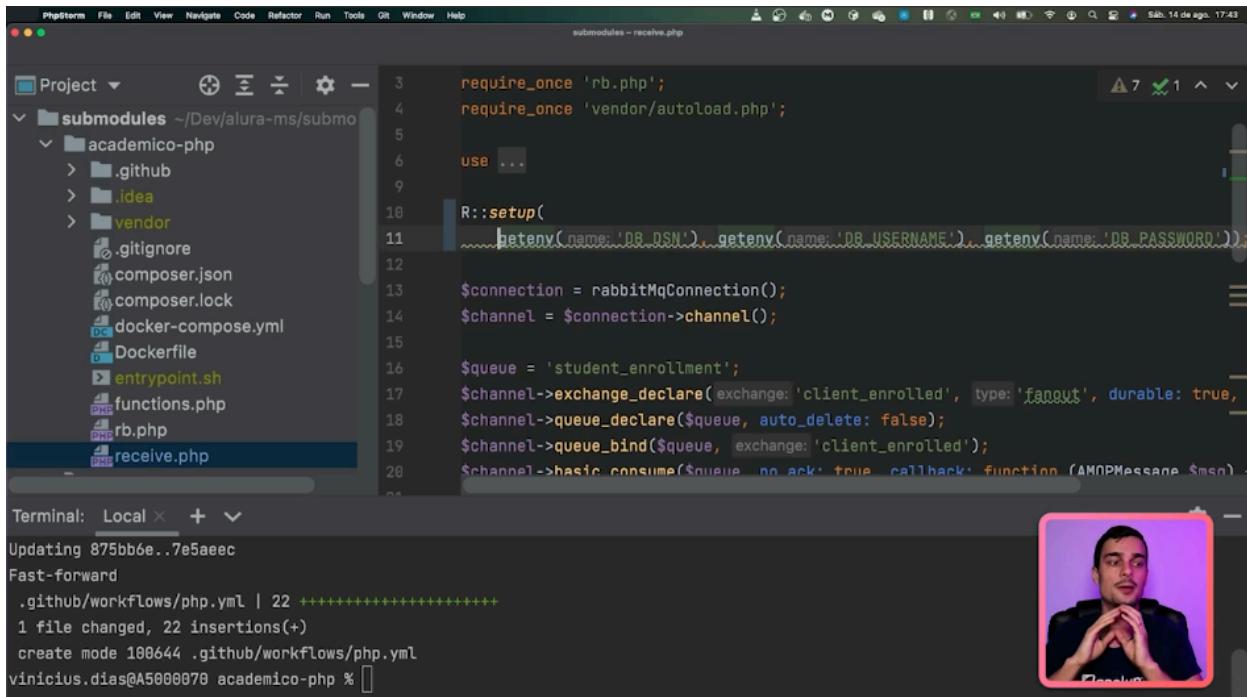
```
Run actions/checkout@v2
  ▶ Determining the checkout info
  71 ▶ Checking out the ref
  75   /usr/bin/git log -1 --format='%H'
  76   '7e5aeecc04aa8c47246897cc97673fec808bd829b'

Run composer install
  ▶ Run php-actions/composer@v6
  1 Building PHP latest with extensions: sockets pdo_pgsql ...
  1 INSTALLING dependencies FROM lock file (including require-dev)
  13 Verifying lock file contents can be installed on current platform.
  14 Package operations: 22 installs, 0 updates, 0 removals
  15   - Downloading doctrine/lexer (1.2.1)
  16   - Downloading paragonie/random_compat (v9.99.100)
  17   - Downloading paragonie/constant_time_encoding (v2.4.0)
  18   - Downloading phpscilib/phpseclib (3.0.9)
  19   - Downloading php-amqplib/php-amqplib (v3.0.0)
  20   - Downloading squizlabs/php_codesniffer (3.6.0)
  21   - Downloading psr/event-dispatcher (1.0.0)
  22   - Downloading symfony/event-dispatcher-contracts (v2.4.0)
  23   - Downloading psr/container (1.1.1)
  24   - Downloading symfony/service-contracts (v2.4.0)
  25   - Downloading symfony/polyfill-php80 (v1.23.1)
  26   - Downloading symfony/polyfill-mbstring (v1.23.1)
  27   - Downloading symfony/polyfill-ctype (v1.23.0)
```

A small video player window in the bottom right corner shows a man speaking.

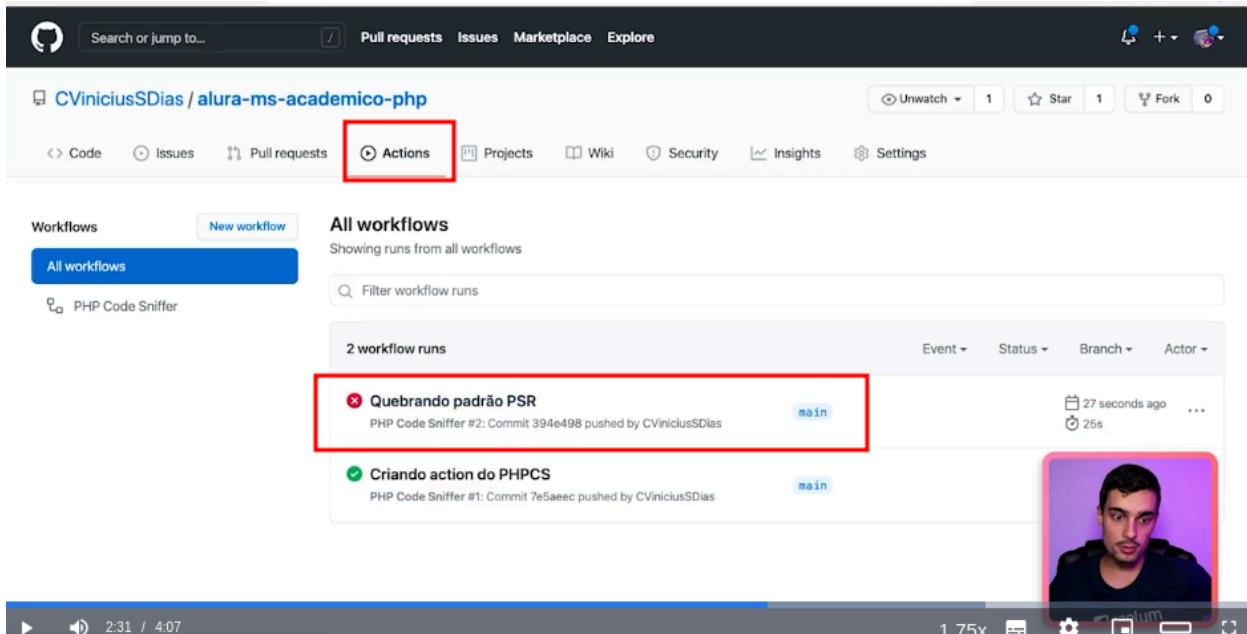
e depois já baixou as dependências/pacotes e por fim já rodou o PHPCS).

Para testar um cenário diferente:



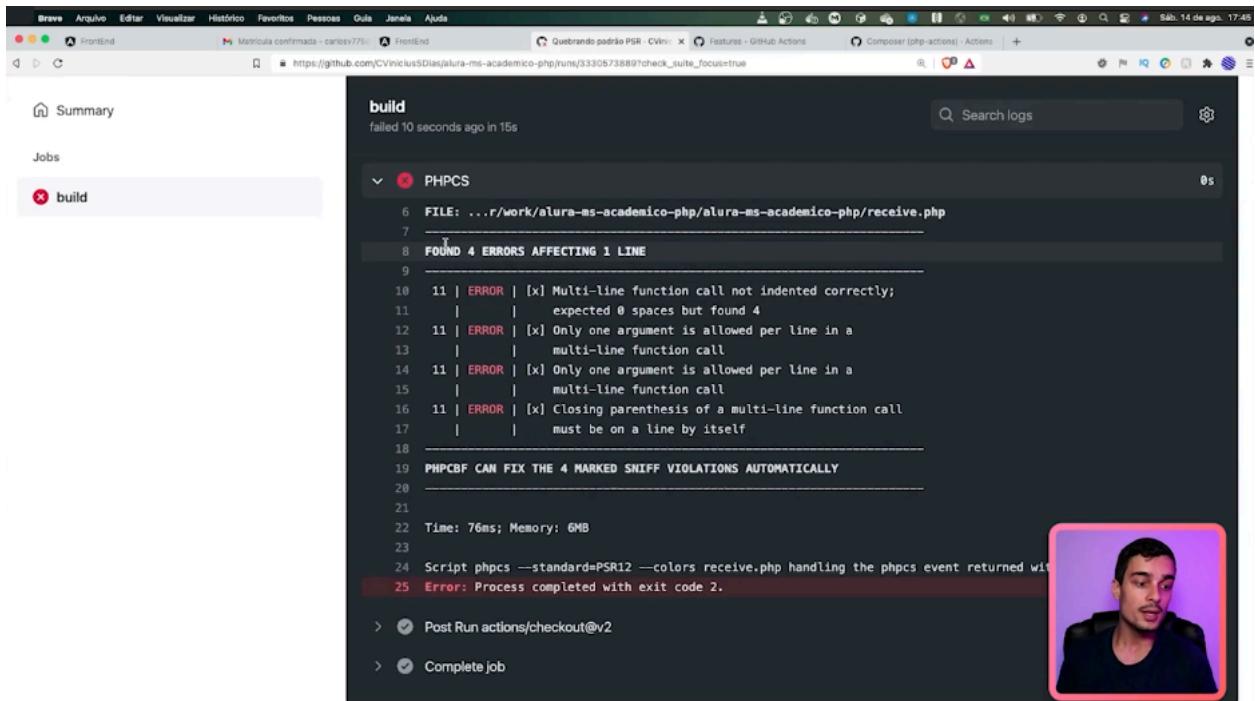
A screenshot of the PhPStorm IDE. The left sidebar shows a project structure with a 'submodules' folder containing 'academico-php'. Inside 'academico-php' are files like .github, .idea, vendor, .gitignore, composer.json, composer.lock, docker-compose.yml, Dockerfile, entrypoint.sh, functions.php, rb.php, and receive.php. The right pane displays the contents of 'receive.php'. The code includes require_once statements for 'rb.php' and 'vendor/autoload.php', and a use statement for 'R'. It then defines a setup function that uses getenv to get DB_DSN, DB_USERNAME, and DB_PASSWORD, creates a rabbitMqConnection, a channel, and a queue named 'student_enrollment'. The channel is configured to be a fanout exchange, and a queue bind is established. A basic consumer is set up to handle messages. Below the code editor is a terminal window showing a git commit message: 'Updating 875bb6e..7e5aeeec Fast-forward .github/workflows/php.yml | 22 +++++++----- 1 file changed, 22 insertions(+)' followed by 'create mode 100644 .github/workflows/php.yml'. On the far right, there is a video player window showing a man speaking.

Ele quebrou a linha e saiu do padrão do PHP, fez o push e a action foi disparada novamente, mas agora vai dar erro:



A screenshot of the GitHub Actions interface. At the top, there are tabs for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The 'Actions' tab is highlighted with a red box. Below it, under 'Workflows', there is a 'New workflow' button and a 'All workflows' button, which is also highlighted with a red box. The main area shows 'All workflows' with a heading 'Showing runs from all workflows'. There is a search bar labeled 'Filter workflow runs'. Below the search bar, there is a table titled '2 workflow runs' with columns for Event, Status, Branch, and Actor. The first run, 'Quebrando padrão PSR', is highlighted with a red box and has a status of 'main'. It was triggered by 'PHP Code Sniffer #2: Commit 394e498 pushed by CVinicioSDias' and completed 27 seconds ago. The second run, 'Criando action do PHPCS', has a status of 'main' and was triggered by 'PHP Code Sniffer #1: Commit 7e5aeeec pushed by CVinicioSDias' and completed 25s ago. On the far right, there is a video player window showing a man speaking.

Deu ruim!



```
build
failed 10 seconds ago in 15s

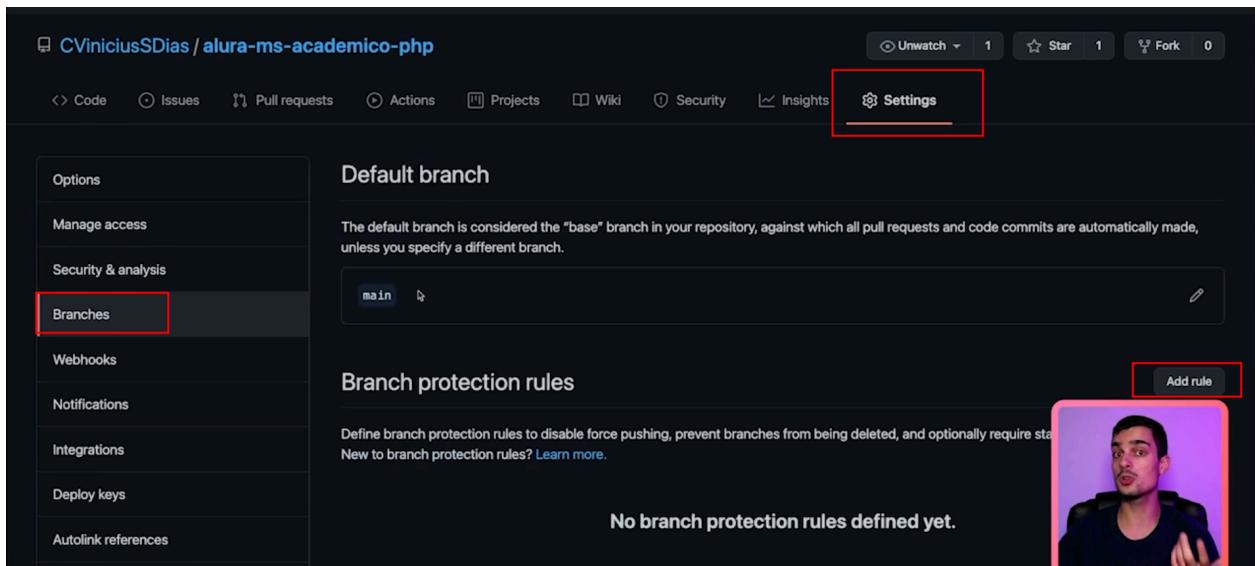
PHPCS
FILE: ...r/work/alura-ms-academico-php/alura-ms-academico-php/receive.php
FOUND 4 ERRORS AFFECTING 1 LINE
11 | ERROR | [x] Multi-line function call not indented correctly;
|           expected 0 spaces but found 4
12 | ERROR | [x] Only one argument is allowed per line in a
|           multi-line function call
13 | ERROR | [x] Only one argument is allowed per line in a
|           multi-line function call
14 | ERROR | [x] Closing parenthesis of a multi-line function call
|           must be on a line by itself
PHPCBF CAN FIX THE 4 MARKED SNIFF VIOLATIONS AUTOMATICALLY
Time: 76ms; Memory: 6MB
Script phpcs --standard=PSR12 --colors receive.php handling the phpcs event returned with
Error: Process completed with exit code 2.

Post Run actions/checkout@v2
Complete job
```

Já mostrou os erros.

Ainda não está muito legal a parte de poder fazer um push direto na main - justamente por conta dessas situações, por exemplo, temos um código que não vai passar, que está quebrado e joguei na main.

Vamos trabalhar com Pull Requests:



CViniciusSDias / alura-ms-academico-php

Unwatch 1 Star 1 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Default branch

The default branch is considered the "base" branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

main

Branch protection rules

Add rule

No branch protection rules defined yet.

Branch name pattern

main

Protect matching branches

Require pull request reviews before merging

When enabled, all commits must be made to a non-protected branch and submitted via a pull request with the required number of reviews and no changes requested before it can be merged into a branch that matches this rule.

Required approving reviews: 1 ▾

Dismiss stale pull request approvals when new commits are pushed

New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

Require review from Code Owners

Require an approved review in pull requests including files with a designated code owner.

Require status checks to pass before merging

Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled,

Require status checks to pass before merging

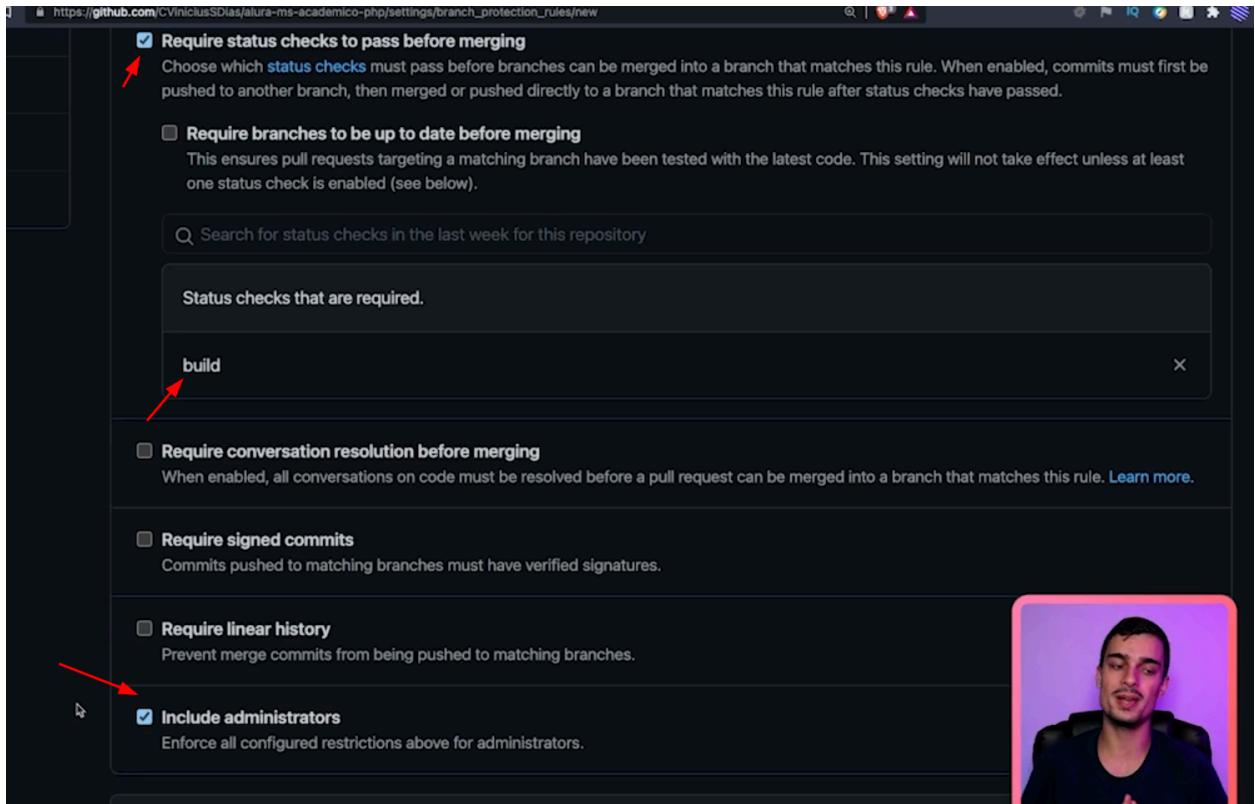
Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

Require branches to be up to date before merging

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Search for status checks in the last week for this repository

Status checks that are required.



The screenshot shows the GitHub settings page for branch protection rules. A red arrow points to the first section, 'Require status checks to pass before merging', which is checked. Another red arrow points to the 'build' entry in the 'Status checks that are required' list. A third red arrow points to the 'Include administrators' checkbox, which is also checked. On the right side of the screen, there is a small video player window showing a man speaking.

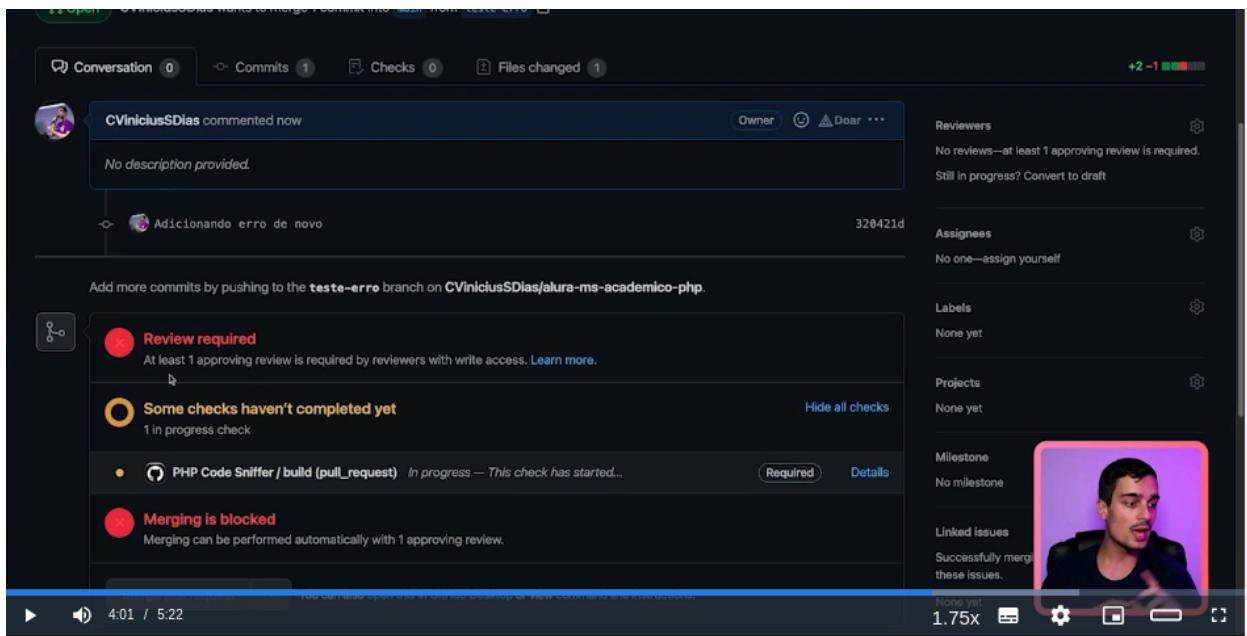
Aqui eu setei que quero verificar o status antes de dar o merging e do job de build... então antes de fazer o merge, vai ter que rodar o build e ver se está tudo certo.

E também inseri para incluir administradores, para que até eu que sou o dono do projeto, precise passar por esse fluxo.

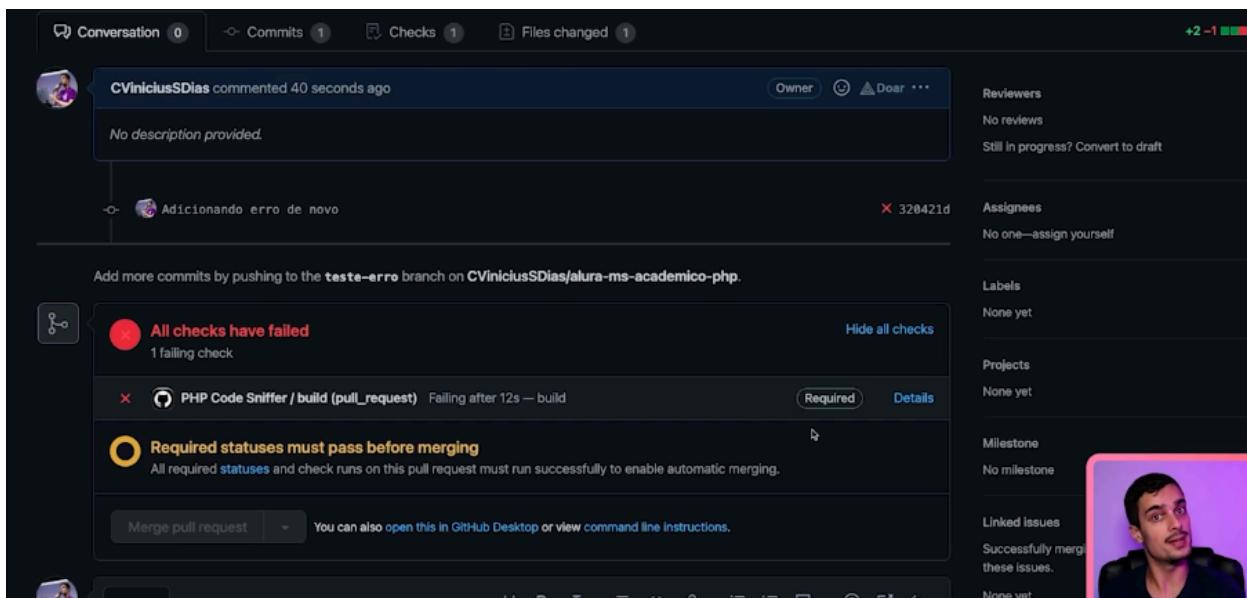
Tentou dar o push na main e deu erro:

```
remote: error: GH006: Protected branch update failed for refs/heads/main.
remote: error: Required status check "build" is expected. At least 1 approving review is required by reviewers wit
To github.com:CViniciusSDias/alura-ms-academico-php.git
! [remote rejected] main -> main (protected branch hook declined)    ↵
error: failed to push some refs to 'github.com:CViniciusSDias/alura-ms-academico-php.git'
vinicius.dias@A5000070 academico-php % git reset --hard origin/main
```

Agora, vai ter que criar outra branch, a partir da main, fazer os commits e push e fazer um PR para a main.



Ele subiu, fez um PR e agora precisa de um reviewer e o status check está rodando ali.. o job do build.



Aqui, deu erro porque ele colocou denovo aquele trecho de código quebrado, então o PR não permite que eu faça o push para a main. Isso ajuda a nos proteger.

Vimos neste vídeo como garantir que nossas actions estejam passando antes de permitir um `merge` de algum `pull request`.

Que vantagem isso nos traz?

A

Garantia de que o código sempre vai estar escrito da melhor forma.



Segurança de que o branch principal sempre vai estar válido.



Alternativa correta! Se algum erro ocorrer no processo de build, o merge deste pull request não é permitido, logo, não pode ser unido ao branch principal. Desta forma ele sempre vai estar válido seguindo nossa rotina de build.

C

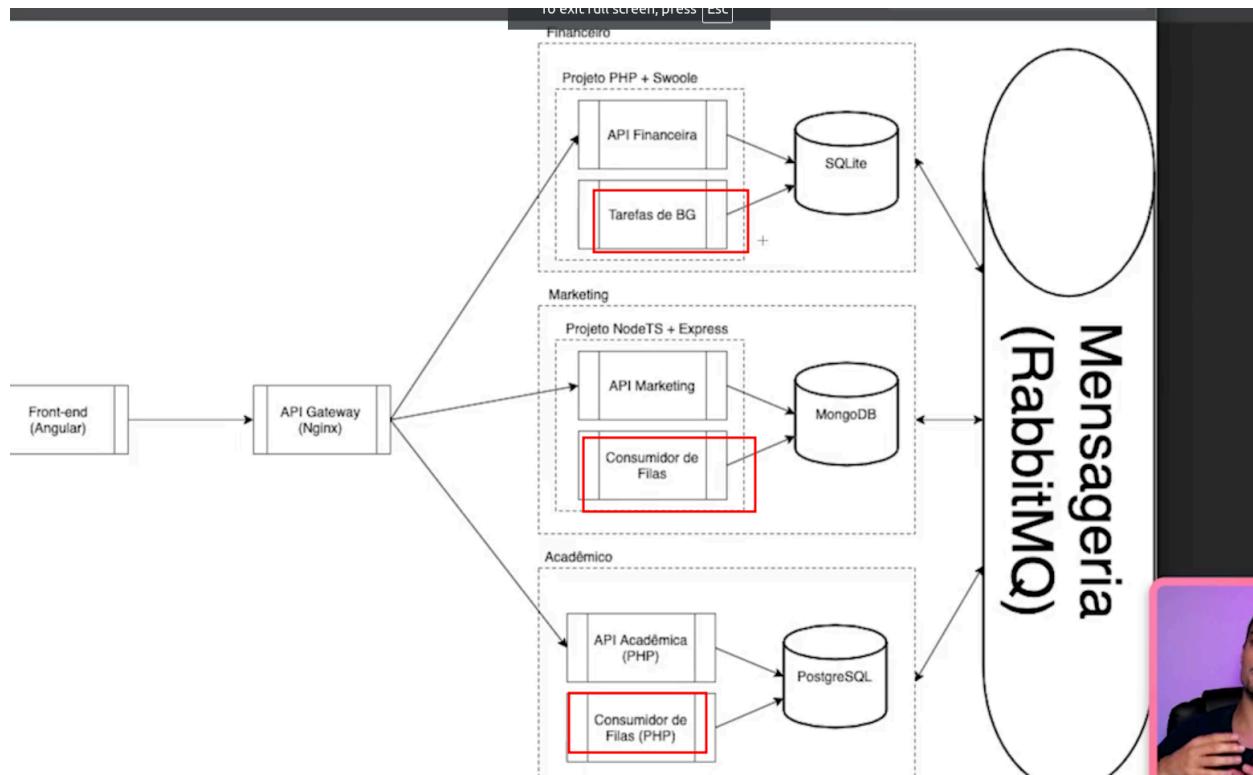
Segurança de que o processo de build é robusto.



Uma conclusão: o código é apenas uma pequena parte da arquitetura de microsserviços.
O código pode ser diferente de um ms para outro, e posso usar diferentes arquiteturas e designs de código de um para o outro.

Em serviços diferentes, posso ter linguagens diferentes e também abordagens totalmente diferentes.

Falando mais especificamente das tarefas de plano de fundo:



Tem várias formas de configurar essas tarefas de plano de fundo:

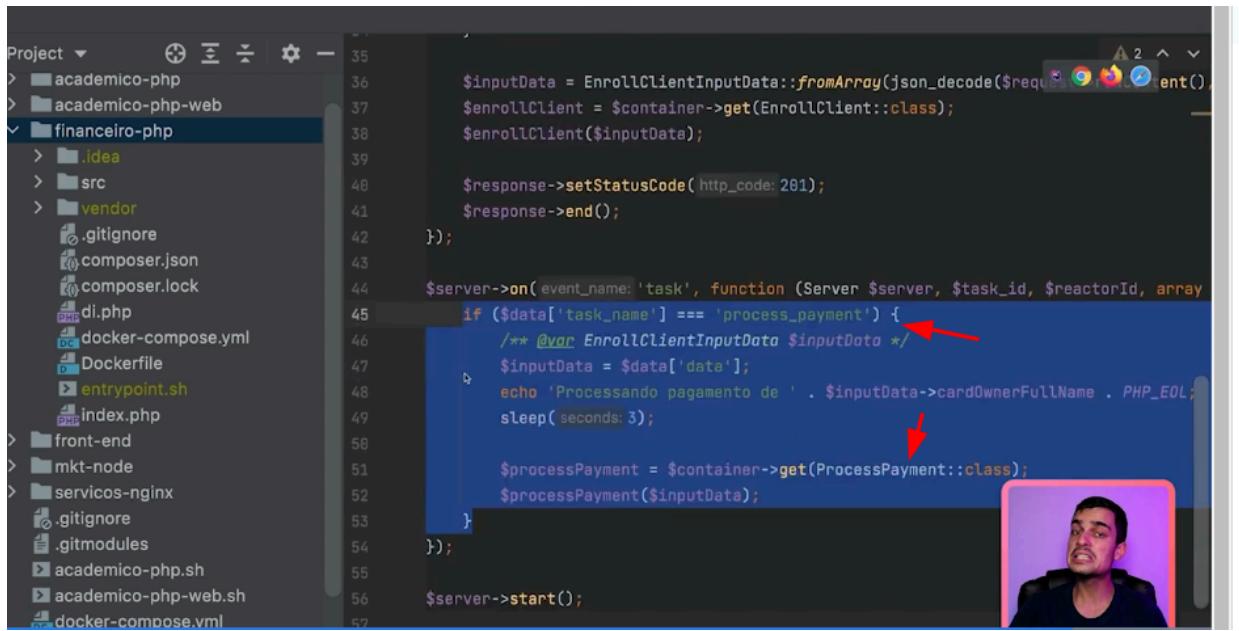
Posso fazer um CRON (no Linux é CRON), ou seja, programar o código para rodar de minuto em minuto, por exemplo, ou de hora em hora.

Posso criar o código para rodar infinitamente, ou seja, crio um código que nunca para de rodar.

No caso do curso, ele escolheu algo que é mais do PHP, por já estar usando o PHP.

Por exemplo: a tarefa de pagamento é algo demorado.. então vou jogá-la para ser executada em uma tarefa de plano de fundo. Poderia por exemplo jogar em um BD e rodar um CRON e minuto em minuto para verificar se tem algum pagamento a ser feito.

No caso dele:



```
Project ▾ 2 ^ v
> academico-php
> academico-php-web
> financeiro-php
> .idea
> src
> vendor
  .gitignore
  composer.json
  composer.lock
  di.php
  docker-compose.yml
  Dockerfile
  entrypoint.sh
  index.php
> front-end
> mkt-node
> servicios-nginx
  .gitignore
  .gitmodules
  academico-php.sh
  academico-php-web.sh
  docker-compose.yml

35
36     $inputData = EnrollClientInputData::fromArray(json_decode($request->getContent()));
37     $enrollClient = $container->get(EnrollClient::class);
38     $enrollClient($inputData);
39
40     $response->setStatusCode( http_code: 201 );
41     $response->end();
42   );
43
44   $server->on( event_name: 'task', function ( Server $server, $task_id, $reactorId, array
45     if ($data['task_name'] === 'process_payment') {
46       /* @var EnrollClientInputData $inputData */
47       $inputData = $data['data'];
48       echo 'Processando pagamento de ' . $inputData->cardOwnerFullName . PHP_EOL;
49       sleep( seconds: 3 );
50
51       $processPayment = $container->get(ProcessPayment::class);
52       $processPayment($inputData);
53     }
54   );
55
56   $server->start();
57
```

Sempre que chega algo com a task name process_payment na API, ele chama o processamento de pagamentos para rodar em um plano de fundo.

Aqui ele usou mensageria, manda isso para uma fila de mensagens.

Conhecimento: CRON

Neste vídeo a ferramenta *cron* foi citada como uma das alternativas para executar tarefas de plano de fundo.

O que é *cron*?

A

É um software específico para executar tarefas de plano de fundo de microsserviços.



É o software agendador de tarefas do Linux.



Alternativa correta! Através do *cron* nós podemos agendar tarefas para rodar diariamente, semanalmente e até de minuto em minuto.

C

É o software agendador de tarefas de qualquer sistema operacional.



DISCUTIR NO FÓRUM

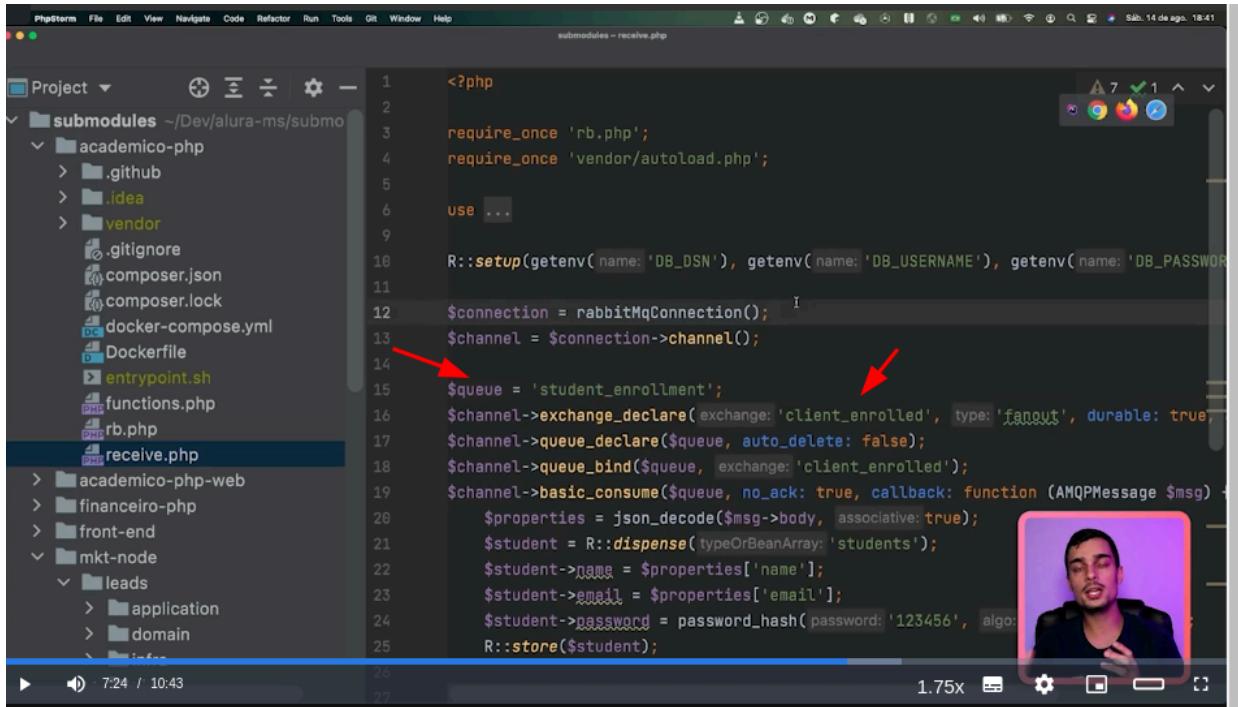


PRÓXIMA ATIVIDADE

Vamos conversar sobre comunicação assíncrona na prática.

A comunicação síncrona é quando, por exemplo, eu faço uma requisição HTTP e espero o resultado.

A assíncrona é quando, por exemplo, eu uso um sistema de mensageria, como o RabbitMq.



```
<?php
require_once 'rb.php';
require_once 'vendor/autoload.php';

use ...

R::setup(getenv('DB_DSN'), getenv('DB_USERNAME'), getenv('DB_PASSWORD'));

$connection = rabbitMqConnection();
$channel = $connection->channel();

$queue = 'student_enrollment';
$channel->exchange_declare(exchange: 'client_enrolled', type: 'fanout', durable: true);
$channel->queue_declare($queue, auto_delete: false);
$channel->queue_bind($queue, exchange: 'client_enrolled');

$channel->basic_consume($queue, no_ack: true, callback: function (AMQPMessage $msg) {
    $properties = json_decode($msg->body, associative: true);
    $student = R::dispense(typeOrBeanArray: 'students');
    $student->name = $properties['name'];
    $student->email = $properties['email'];
    $student->password = password_hash(password: '123456', algo: 'sha256');
    R::store($student);
});
```

Eu escuto uma exchange chamada “client_enrolled” e crio uma fila chamada student_enrollment específica para o meu microsserviço.

Aqui criei uma fila específica para conversão de aluno.

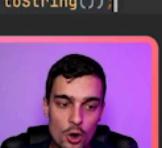
No de mkt, eu criei uma fila específica para conversão de leads, para o microsserviço olhar para ela.

E ai eu dou um bind, ou seja, sempre que chega uma mensagem na exchange, eu mando a mensagem para a minha fila em questão que foi criada.

Esse é o conceito no RabbitMq de fanout, ou seja, sempre que uma mensagem chega em uma exchange, essa exchange manda para várias filas, várias filas podem consumir essa mensagem.

É a mesma lógica para as mais diferentes linguagens.

Para o caso de marketing:



The screenshot shows a developer's interface with a code editor and a video overlay. The code editor displays a TypeScript file named `rabbitmq-consumer.ts`. The developer has highlighted several lines of code with red arrows:

```
    amqplib.connect(url: {
      hostname: config.rabbitmqHost,
      port: <number> config.rabbitmqPort,
      username: config.rabbitmqUser,
      password: config.rabbitmqPassword,
    }) Bluebird<Connection>
      .then(onFulfill: conn => conn.createChannel()) Bluebird<Channels>
      .then(onFulfill: async channel => {
        await channel.assertExchange(exchange: 'client_enrolled', type: 'fanout',
        await channel.assertQueue(queue: 'lead_conversion', options: { autoDelete: true },
        await channel.bindQueue(queue: 'lead_conversion', source: 'client_enrolled')

        return channel;
      }) Bluebird<Channel>
      .then(onFulfill: channel => channel.consume(queue: 'lead_conversion', onMessage: m
        const properties = JSON.parse(<string> msg?.content.toString());
        const useCase = new ConvertLead(repository);
        useCase.execute(properties.email);
      }));
    }
}
```

The developer is likely explaining the logic for connecting to RabbitMQ, creating a channel, and setting up a consumer to handle messages from the 'lead_conversion' queue.

Mesma lógica.

Conceito de Fanout: <https://www.rabbitmq.com/tutorials/tutorial-three-php>

Vamos falar agora sobre front-end - o front é em angular.

Até agora, temos apenas um front-end... todas as páginas são da mesma aplicação!

Mas, poderia ter várias aplicações de front-end.. por exemplo uma para fazer o cadastro, etc. e outra para ser a área logada.

Na Alura tem isso.. uma para o cadastro e outra para a área logada.

Temos duas abordagens de front-end para lidar com microsserviços no back-end... temos que estar preparados em nosso back-end, mas também no nosso front-end, para os casos onde algum microsserviço pode estar fora, etc.

Temos a abordagem Optimistic:

The screenshot shows a web application interface on the left and its corresponding Network tab in the browser developer tools on the right. The UI displays a list of courses under 'Cursos disponíveis'. The 'Entrega contínua' course card is highlighted with a red box and a cursor is hovering over it. In the developer tools, two requests are visible: a preflight request (Type: Preflight...) and a main request (Type: xhr). Both requests show a status of 200 and a size of 0 B. An arrow points from the UI's 'Entrega contínua' card to the main request in the Network tab.

Um exemplo é: quando eu clico para marcar que já fiz um curso, ele já atualiza no front-end primeiramente, sem saber a resposta do meus microserviços. No primeiro curso, eu marquei como assistido, ele já muda na hora no front-end e espera o retorno da API. Deu sucesso! No segundo curso, eu derrubei o serviço web através do docker stop e marquei o curso como assistido.. o front-end muda na hora e espera o retorno da API.
No nosso caso, deu timeout e deu erro:

The screenshot shows the same web application and developer tools as the previous one. However, the 'Entrega contínua' course card is now highlighted with a red box and a cursor is hovering over it. In the developer tools, the main request (Type: xhr) has failed with a status of 502 and a reason code of CO... (Connection reset by peer). The preflight request (Type: Preflight...) is still successful with a status of 200.

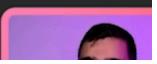
Então, ele reverteu a alteração no front-end, ou seja, o curso voltou a ser não assistido. Isso se chama Optimistic UI Update / Optimistic renderization. Eu estou tomando uma abordagem otimista para renderizar a aplicação, eu parto do princípio que vai dar tudo certo e se der errado, ele desfaz o que foi feito.

```
@Injectable()
export class CursosService {

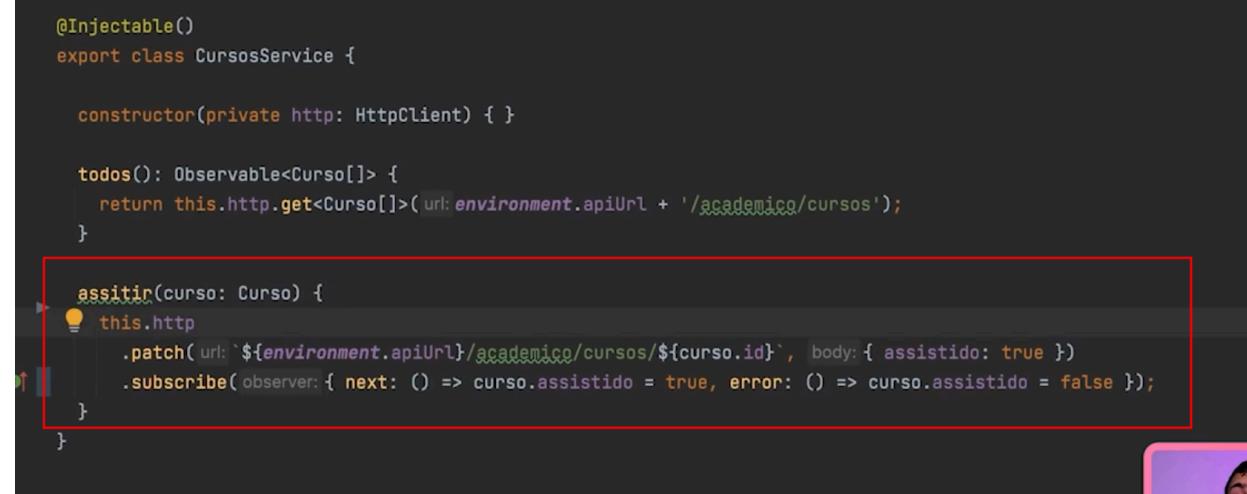
  constructor(private http: HttpClient) { }

  todos(): Observable<Curso[]> {
    return this.http.get<Curso[]>(`url: ${environment.apiUrl} /academico/cursos`);
  }

  assitir(curso: Curso) {
    curso assistido = true;
    this.http
      .patch(`url: ${environment.apiUrl} /academico/cursos/${curso.id}`, { assistido: true })
      .subscribe(observer: { error: () => curso assistido = false });
  }
}
```



A outra abordagem é a Pessimistic, ou seja, eu assumo que vai dar erro e não atualizo de cara a interface, e caso dê certo, ai sim eu atualizo.



```
@Injectable()
export class CursosService {

  constructor(private http: HttpClient) { }

  todos(): Observable<Curso[]> {
    return this.http.get<Curso[]>(url: environment.apiUrl + '/academico/cursos');
  }

  assitir(curso: Curso) {
    this.http
      .patch(` ${environment.apiUrl}/academico/cursos/${curso.id}`, { assistido: true })
      .subscribe(observer: { next: () => curso.assistido = true, error: () => curso.assistido = false });
  }
}
```

Somente em caso de sucesso eu atualizo, se não, nem faço nada.

Só marca como assistido, quando a API retornar como sucesso!

Otimistic x Pessimistic

03 Qual abordagem?

PRÓXIMA ATIVIDADE

Vimos neste vídeo 2 abordagens para lidar com renderização de atualizações na interface: Otimista e Pessimista.

Qual das 2 abordagens é a correta quando se trata de microsserviços?

A Otimista

B Pessimista

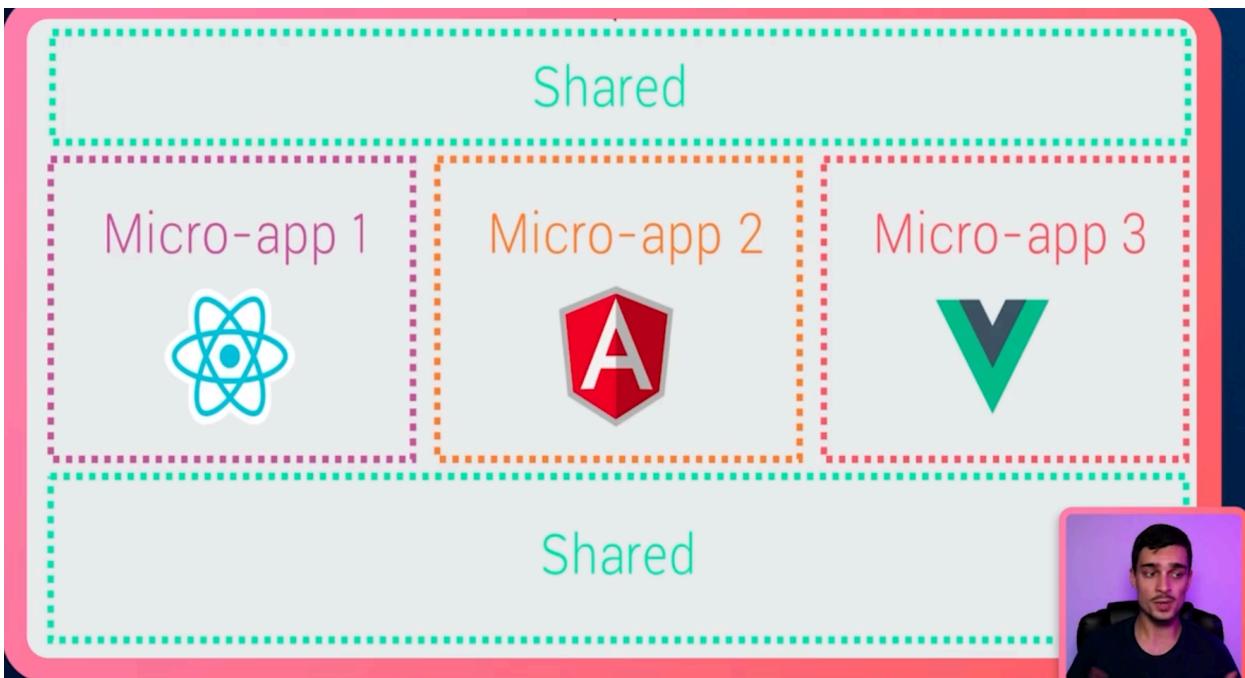
C Depende

Alternativa correta! Ah, a palavrinha mágica: "Depende". Se seu *uptime* é altíssimo e sua taxa de falhas é mínima, a abordagem otimista traz uma sensação de mais fluidez no uso da aplicação. Se a taxa de falhas da aplicação é considerável, a abordagem pessimista frustra menos o usuário, visto que não há uma interação sendo desfeita. Cada caso é um caso, então vale avaliar.

DISCUTIR NO FÓRUM

PRÓXIMA ATIVIDADE

Bônus: Microfrontends



Para o nosso exemplo:

A screenshot of a web browser showing a multi-step form for a purchase. The form consists of three steps: 1. Dados pessoais (Personal data), 2. Detalhes de pagamento (Payment details), and 3. Fim (End). The first step shows fields for Nome Completo (Full Name) and E-mail. The third step is labeled "Fim". To the right, a sidebar displays "Informações da sua compra" (Purchase information) including a plan named "Plano mega power", total value "R\$ 123,45", 1234 cursos (courses), and a list of benefits: "Estude por 1 ano", "Certificado de participação", and "Apps para Android e iOS". A video feed of a person's face is visible in the bottom right corner.

A tela de dados pessoais poderia ser uma aplicação React, a etapa 2 de Detalhes do Pagamento poderia ser uma aplicação em Angular e a tela 3 poderia ser uma aplicação em View, por exemplo.

Isso é o conceito de micro front-ends.

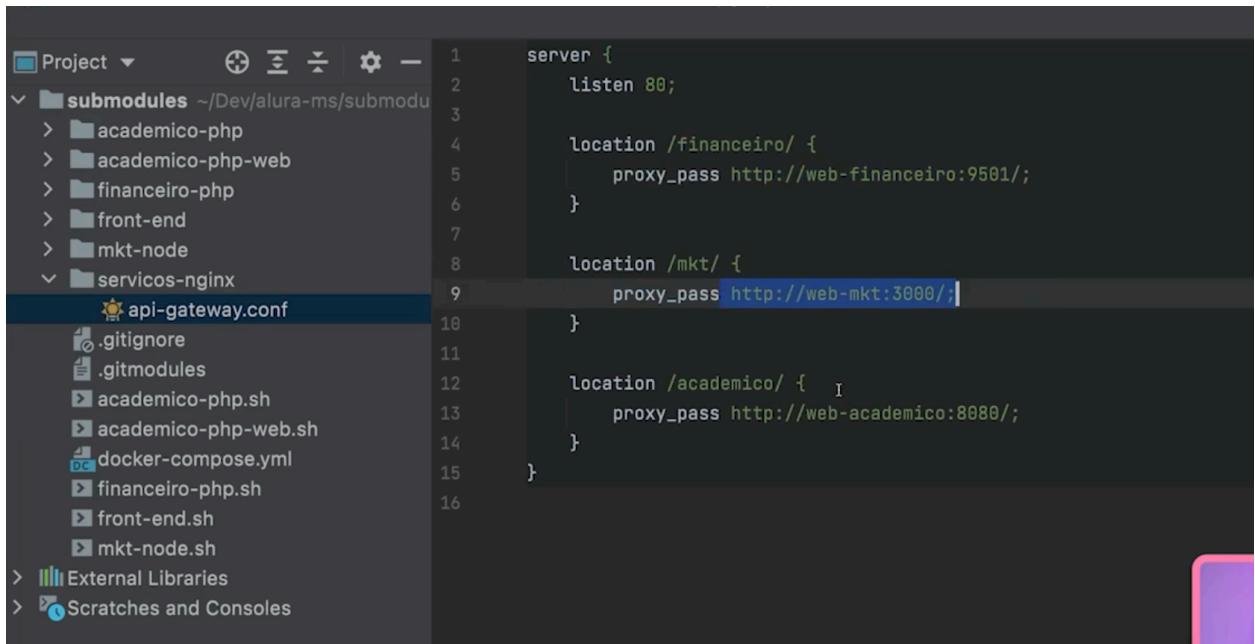
Geralmente a abordagem é usada quando temos uma aplicação bem complexa, com uma grande riqueza de detalhes e complexidade, e para não precisar, ao subir uma alteração em uma tela, atualizar todas as outras, eu separo em microfrontends.

Vários front-ends é: eu tenho várias telas e cada uma das telas é uma aplicação.

Micro front-ends: na mesma tela, eu tenho várias aplicações diferentes.

Mesmo conceito de microsserviços.

Extra: nginx



The screenshot shows a code editor with an nginx configuration file named `api-gateway.conf`. The file contains the following code:

```
1 server {
2     listen 80;
3
4     location /financeiro/ {
5         proxy_pass http://web-financeiro:9501/;
6     }
7
8     location /mkt/ {
9         proxy_pass http://web-mkt:3000/; // Line 9
10    }
11
12     location /academico/ {
13         proxy_pass http://web-academico:8080/; // Line 13
14     }
15 }
16
```

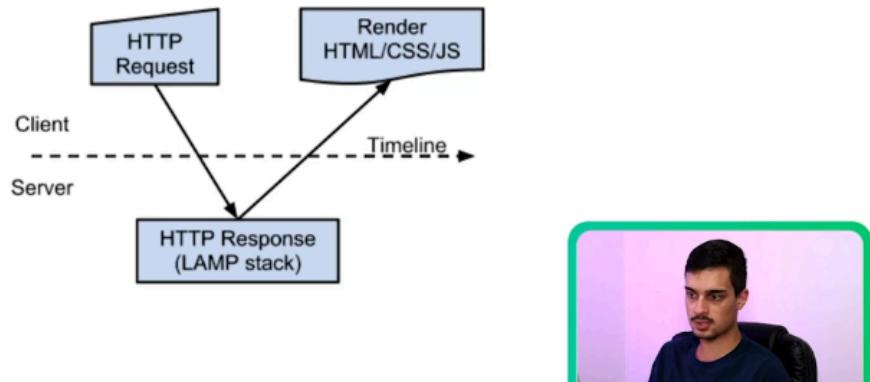
Se eu entro em /financeiro, ele me redireciona para o serviço financeiro, se eu entro em /mkt, ele me redireciona para o serviço de marketing e assim por diante.

O nginx pode ser usado para gerar algumas métricas, por exemplo... horário de pico, quantidade de acessos, etc.

Segundo curso - Microsserviços: padrões de projeto

Vamos entender na prática o que são microsserviços e para isso, como funciona a web: Internet funciona através de um protocolo chamado HTTP:

HTTP

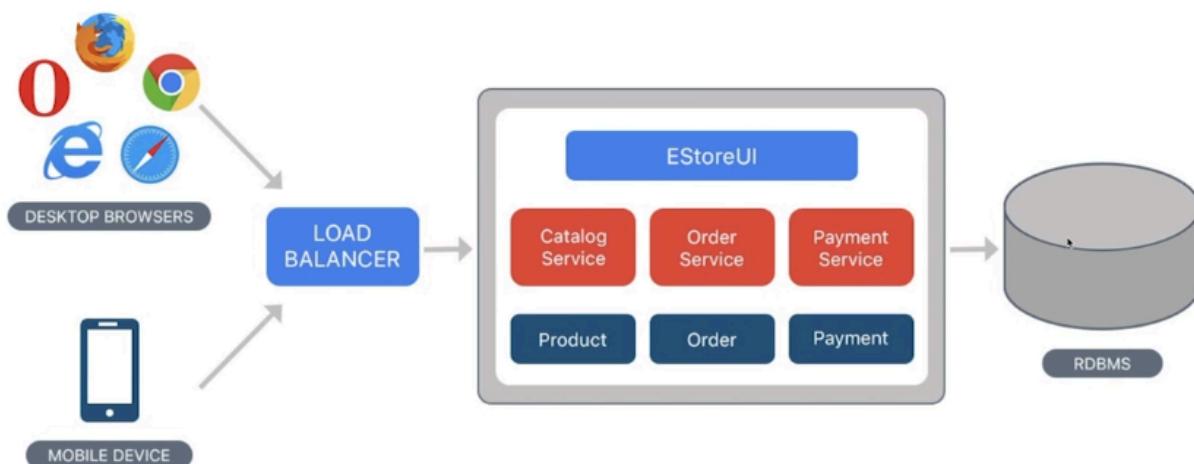


Através de algum cliente, um celular, computador, qualquer coisa que acesse a web, eu acesso algum lugar.

Quando digito “www.alura.com.br”, o meu navegador (cliente) envia um pedido para o servidor. No caso, o servidor da Alura. O servidor da Alura vai então processar o pedido (requisição) e vai verificar uma série de coisas (se é aluno, quais cursos já fez, etc.). Depois que fizer tudo, monta uma resposta, geralmente em HTML, e o navegador que sabe conversar com essa resposta, vai interpretar e montar/renderizar a página na web.

Vamos agora entender os conceitos de aplicações monolíticas:

Aplicações monolíticas



Temos os clientes (navegadores, mobiles, etc.) e esse clientes fazem suas requisições para o servidor. Essas requisições passam por um load balancer, para fazer o balanceamento de carga, como o próprio nome já diz e faz com que sejam direcionados para máquinas menos sobrecarregadas.

No caso da aplicação monolítica, uma única aplicação vai ser várias responsabilidades, como por exemplo consultar o pedido, consultar o cliente que está logado, verificar cursos, montar resposta, fazer o pagamento, etc.

E tudo isso é armazenado em um banco de dados.

Problemas dessa abordagem:

Alguns problemas

- Demoras no deploy
- Falhas podem derrubar o sistema todo
- 1 projeto = 1 tecnologia



Exemplo: Facebook -> se eu mexer apenas na parte de notificações, teria que fazer o redeploy da aplicação inteira, em partes que nem sequer foram alteradas. Além disso, se eu inserir uma falha sem querer em uma parte pequena, vai quebrar toda a minha aplicação, inclusive partes da aplicação que não tem nada a ver com o que foi alterado. Além disso, estamos presos em uma só tecnologia.

A abordagem de aplicações monolíticas, assim como qualquer abordagem no desenvolvimento de software, possui vantagens e desvantagens.

Qual importante desvantagem de aplicações monolíticas?



A Deploy possivelmente mais perigosos



Alternativa correta! Uma aplicação monolítica tem um único deploy, então um erro em uma parte da aplicação pode quebrar uma outra que não tem nenhuma relação.

B

Maior simplicidade da infra



C

Base de código única



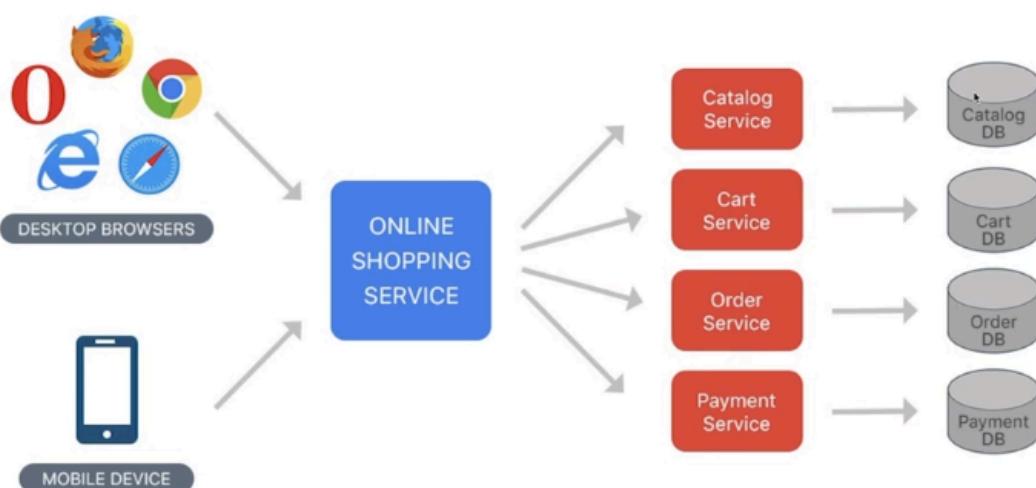
DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora, vamos falar sobre a arquitetura de microsserviços:

Arquitetura de microsserviços



Então todas aqueles “serviços/funcionalidades” que tínhamos na aplicação monolítica, agora vão estar em serviços separados, ou seja, tenho vários serviços e cada serviço tem seu banco de dados em específico. Logo, restringimos e separamos as responsabilidades em pequenos serviços, chamados de microsserviços.

Se por exemplo o serviço de pedidos sair do ar, ainda consigo ver na aplicação o catálogo de produtos, e todos os outros serviços que não caíram, ou seja, não cai a aplicação inteira.

Temos mais flexibilidade.

Definição formal:

Definição formal

Microsserviços são uma abordagem arquitetônica e organizacional do desenvolvimento de software na qual o software consiste em pequenos serviços independentes que se comunicam usando APIs bem definidas. Esses serviços pertencem a pequenas equipes autossuficientes.

E vale ressaltar que os microsserviços, para comporem uma aplicação total, eles precisam se comunicar, precisam se conhecer, pois como vou fazer um pedido se não sei o catálogo de produtos que tenho?!

Ele não precisa conhecer o código, a estrutura do banco de dados, etc. Mas tem que saber um endpoint específico onde eu consigo visualizar todas as características daquele produto em si.

Quando usar?

Entendemos neste vídeo o que são microsserviços.

Antes mesmo de conhecer suas vantagens e desvantagens, em que cenário você adotaria essa abordagem?

A Quando manter uma aplicação monolítica se tornar um problema

Alternativa correta! Se uma aplicação monolítica tem deploys muito demorados e frágeis, usa tecnologias que não parecem ser coerentes com a tarefa e precisaria ser quebrada em equipes individuais, microsserviços podem ser a solução.

B Sempre que for desenvolver um novo sistema

C Nunca adotaria essa abordagem

 DISCUTIR NO FÓRUM  PRÓXIMA ATIVIDADE

Vantagens x desvantagens:

Algumas vantagens

- Projetos independentes = tecnologias independentes
- Falha em 1 serviço é isolada
- Deployes menores e mais rápidos



Podemos ter equipes diferentes, trabalhando com tecnologias diferentes, stacks diferentes, para os serviços da aplicação e todo esse trabalho compõe a aplicação.

Podemos ter também escalas diferentes para os serviços, por exemplo, o serviço de assistir os cursos é muito mais usado do que o serviço de pagamento, logo, ele precisa ser escalado de uma forma maior do que o de pagamento. Consigo ter esse dinamismo.

Algumas desvantagens

- Maior complexidade de desenvolvimento e infra
- Debug mais complexo
- Comunicação entre os serviços deve ser bem pensada
- Diversas tecnologias pode ser um problema
- Monitoramento é crucial e mais complexo

Por aumentar muito a complexidade, tanto de desenvolvimento de código, quanto de infra, por ter mais servidores, diferentes bancos de dados, etc., eu preciso pensar se realmente preciso implementar o microserviço. Não é quando eu quero implementar, e sim quando o monolito está causando uma toxidade para o sistema, demorando muito para fazer deploy, etc.

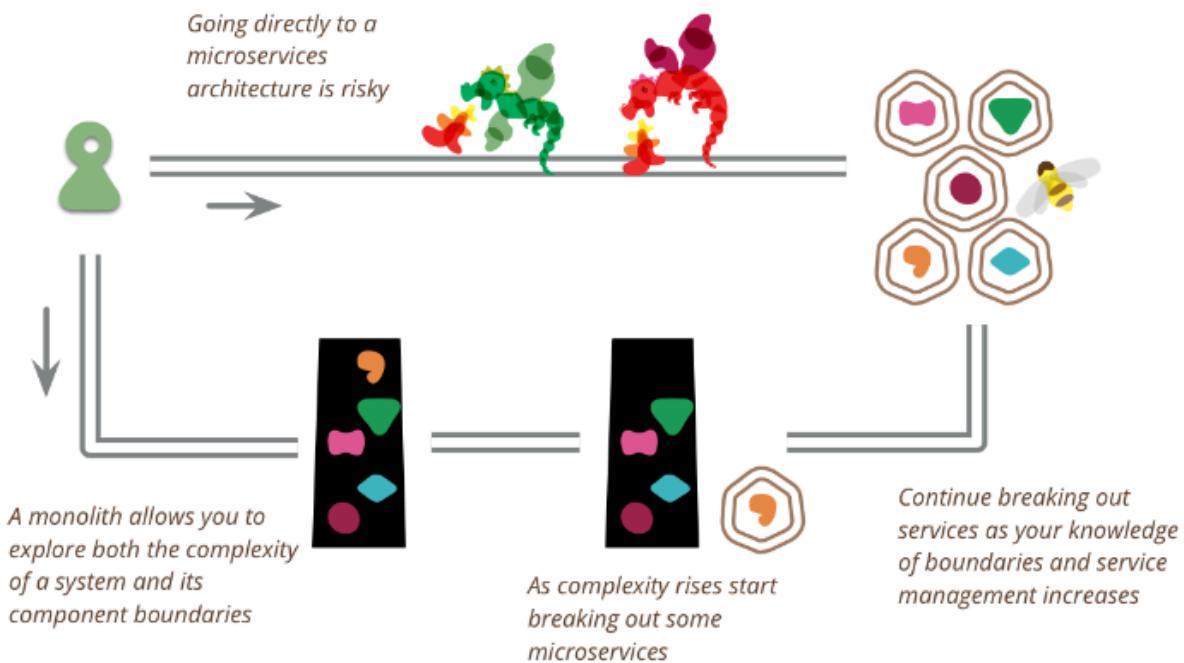
Quando utilizar?

Referência interessante

<https://martinfowler.com/bliki/MonolithFirst.html>

<https://martinfowler.com/bliki/MonolithFirst.html>

Monolito por padrão



Ele usa a abordagem do monolito first, ou seja, sempre começar com um monolito para evitar complexidades no início do desenvolvimento.

Assim que for percebendo a necessidade de se ter uma abordagem de microsserviços, ai sim começa.

Quais os tipos de microsserviços?

Tipos de microsserviços

- Data service
- Business service
- Translation service
- Edge service

- **Data Service:** tipo de serviço que vai simplesmente expor dados. Uma fina camada antes do banco de dados, que recebe os dados, faz algum processamento antes de incluir no banco de dados, para manter os dados consistentes.
- **Business service:** além de consumir dados, ou consumindo um data service, ou acessando diretamente o banco de dados, ele ainda fornece operações mais complexas, agregando dados e tomando decisões.
- **Translation service:** uma forma de acessar algum recurso externo, mantendo um certo controle. Por exemplo: quero consumir um serviço de log externo, posso ter um serviço que consome um serviço de log externo na minha aplicação, porque se o serviço externo mudar, eu já sei onde mudo na minha aplicação.
- **Edge service:** serviço de ponta, entregue diretamente para o cliente, podendo ter necessidades específicas. Por exemplo: na alura, temos um edge service para mostrar os cursos para os clientes web e um para os clientes mobile, porque no aplicativo não vamos mostrar a mesma coisa que na web.

Como podemos separar serviços?

Como determinada parte da aplicação monolítica vira determinado tipo de serviço, etc.?

Vamos fazer isso usando o conceito de Serviços de Domínio, ou seja, não vamos pegar o contexto inteiro da aplicação e sim somente o contexto específico de algum domínio.

Serviços de domínio

- Domain-driven Design
- Comece modelando seu domínio, não pensando na persistência
- Avalie as ações que serão disponibilizadas
- Construa o serviço, pensando primeiro no contrato
- REST e RPC podem andar juntos



Importante conhecer bastante as técnicas de DDD (Domain-driven Design).

DDD:

<https://medium.com/beelabacademy/domain-driven-design-vs-arquitetura-em-camadas-d01455698ec5>

Na prática, vamos começar modelando o domínio, não vamos pensar na persistência ainda.

Exemplo: Alura, quero ter um data service que vai ser do domínio de aluno - sempre que um aluno se matricular ou alterar algum dado seu, vamos ter que alterar um serviço que lida com os alunos.

E ai depois disso, vou avaliar o que será disponibilizado no serviço... vou poder alterar todas as infos do aluno? Vou poder usar esse serviço para cadastrar o aluno? Vou usar para outra coisa?

Vamos inserir um aluno ou matricular um aluno? Qual faz mais sentido para o domínio em análise?

Na hora de alterar, vai ser PUT ou PATCH?

Pensar muito bem no contrato, no que vou receber e enviar, ter tudo isso muito bem definido.

Então, o que foi visto são os Domain Services, um tipo de Data Service que tem como propósito fornecer acesso a determinado domínio e suas regras (somente àquele domínio).

Aprendemos neste vídeo sobre *Domain Services*, que são um tipo possível de *Data Services*.

Qual o propósito deste tipo de serviço?

A

Fornecer funcionalidades específicas para cada cliente



Fornecer acesso a determinado domínio e suas regras



Alternativa correta! Um domain service fornece acesso a um único domínio da aplicação e lá suas regras estão contidas.

C

Fornecer acesso a um processo de negócio



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Vimos então como separar um data service usando o conceito padrão de serviços de domínio. Pegamos um pequeno domínio da nossa aplicação e separa em um serviço, que vai fornecer o acesso aos dados desse domínio.

As vezes, possuímos regras que precisam ser implementadas, mas que dependem de mais de um domínio.

Para isso, temos os serviços de negócio (business service) - é basicamente a junção de vários data services.

Serviços de negócio

Em determinados momentos as operações precisam de mais de um modelo do nosso domínio para serem corretamente representadas em um serviço.

Por exemplo: matrícula de alunos - não depende apenas de um registro no domínio de alunos, talvez temos que ter um registro no serviço financeiro também, na gameficação, etc. Temos detalhes muito mais complexos do que lidar somente com um domínio.

Precisamos pensar bem sobre o processo do domínio do negócio.

Processo do domínio do negócio

- Proveem uma funcionalidade do negócio de mais alto nível
- Permite encapsular domínios relacionados

Existe um processo que vai ser executado, quando pensamos com uma visão de negócios (nem vamos pensar em código por enquanto).

Ex: matricular aluno

Esse processo é representado através desse serviço específico, provendo uma funcionalidade mais alto nível - matricular aluno e não granulares - inserir um aluno, inserir uma matrícula financeira, inserir uma matrícula em gameficação, etc. Então, vai ser um processo de mais alto nível - matricular - e através dele, as operações de mais baixo nível (essas todas operações que devem ser feitas), vão ser orquestradas.

Dessa forma, conseguimos encapsular domínio relacionados, ou seja, o cadastro de um aluno e o cadastro da matrícula financeira tem um relacionamento e conseguimos encapsular isso, sem ter a necessidade de quando se vai cadastrar um aluno, o cliente precisar chamar vários endpoints (inserir aluno, inserir matrícula financeira, inserir na gameficação) - conseguimos encapsular isso tudo e simplificar o processo, tornando ele mais próximo do que falamos no domínio.

Para criar um serviço de negócio:

- Identifique o processo que você pretende expor
- Identifique os domínios que serão necessários nesse serviço
- Defina a API que será utilizada, focando no domínio e não nos dados
- Consuma serviços de domínio para executar os processos

Agora, vamos ver alguns padrões:

- Começar com o monolito.

Já falamos desse, ou seja, começar as aplicações como monolitos e somente se tivermos a necessidade, construiremos um microsserviço a partir dele.

O primeiro padrão que vamos ver é o de Estrangulamento (Strangler pattern):

Strangler pattern

- Quebrar um monolito, tirando as funcionalidades dele
- Podemos começar isolando os dados
- Ou podemos começar isolando o domínio

Posso começar separando os dados, separando os bancos de dados ou até mesmo os domínios em serviços.

Outro padrão é o Sidecar pattern.

Supondo que precisamos compartilhar um processo comum: exemplo - logs. Poderia ter um serviço que faz só log e todos os outros serviços conversam com ele. Mas, tem vezes que não faz sentido, então vou deixar em um serviço, mas é um módulo compartilhável, ou seja, vários serviços podem usar.

Podemos ter um pedaço de código que ajuda outros microsserviços, mas vai estar dentro de um em específico.

Sidecar pattern

- Determine o processo comum
- Construa um módulo compartilhável
- Aplique esse *sidecar* nos serviços que precisam dele



O processo comum por exemplo realizar logs - é comum entre todos os serviços.

Vamos criar um pacote que pode ser compartilhado entre os vários serviços de forma independente, mas o serviço, o código em si, está em um lugar só, de forma que se eu atualizar esse código, essa atualização será refletida em todos os serviços que utilizam.

Por exemplo, no java, criaria um novo pacote para ser usado no maven e gradle, no node seria no npm, etc.

Esse padrão é sobre manter códigos compartilhados.

Conhecimento: quebrando o monolito

The screenshot shows a mobile application interface for a learning activity. At the top, there is a navigation bar with a back arrow, the title 'Quebrando o monolito' in bold white text, and a blue button labeled 'PRÓXIMA ATIVIDADE' with a right-pointing arrow icon. Below the title, there is a large video player area with a play button icon and a progress bar indicating '05'. The main content area contains a question and three answer options. The question is: 'Por que é interessante quebrar a aplicação em serviços depois que ela já está desenvolvida e funcionando?'. Option A is highlighted with a green checkmark icon and the text: 'Porque neste momento conhecemos melhor o negócio'. A callout box next to it says: 'Alternativa correta! Com uma aplicação já funcional, é muito mais fácil identificar os domínios que precisam ser separados e o que não é tão crítico, além de termos mais confiança nas implementações das regras.' Options B and C are shown below: 'Porque os gurus do desenvolvimento recomendam essa abordagem' and 'Porque assim deixamos a parte difícil para o final'. At the bottom, there are two buttons: 'DISCUTIR NO FÓRUM' with a speech bubble icon and 'PRÓXIMA ATIVIDADE' with a right-pointing arrow icon.

Neste vídeo aprendemos 2 padrões muito comuns no desenvolvimento de microsserviços. Mas vamos falar do padrão *Strangler*.

Por que é interessante quebrar a aplicação em serviços depois que ela já está desenvolvida e funcionando?

A Porque neste momento conhecemos melhor o negócio

B Porque os gurus do desenvolvimento recomendam essa abordagem

C Porque assim deixamos a parte difícil para o final

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Agora vamos falar um pouco sobre integração entre serviços.

Um ponto importante disso é ter um ponto único de entrada, ou seja, quando o meu cliente precisa se conectar com meus serviços, é importante demais que se tenha um ponto único de entrada para a minha aplicação.

Ex: algum serviço pode esperar em xml, outro em json, outro em binário. Mas o cliente da minha API precisa de uma abstração de tudo isso, precisa apenas chamar uma URL, passando o que deseja fazer e o ponto único de entrada se encarrega de direcionar para o serviço responsável e realizar a normalização de dados.

Para ter esse ponto único de entrada, temos um padrão, o mais famoso quando falamos de microsserviços, chamado API Gateway:

API Gateway

- Problema: Clientes acessando livremente os serviços geram caos
- Gateway fornece um proxy, uma fachada, para as necessidades reais
- **Desvantagem: Esse portão de entrada pode se tornar um ponto central de falha**

Esse caos é sem o uso do gateway, porque imagina ter que acessar vários serviços diferentes... se eu acessar em ordem diferente, vai dar um bo, além de gerar uma certa má experiência para o cliente que está consumindo minha API. Para isso, podemos usar o gateway.

O padrão Gateway é como se fosse um portão, ou seja, um ponto de entrada único que vai saber como direcionar as requisições corretamente na minha aplicação para conseguir realizar o que precisa ser realizado.

Pode ser um ponto central de falha porque se o API Gateway cai, minha aplicação toda cai também.

Comportamento:

Comportamentos do Gateway

- Simplesmente autorizar e redirecionar os requests
- Uso de *Decorator* para adicionar informações necessárias aos requests
- Limitar o acesso ou conteúdo trafegado

Pode simplesmente autorizar e redirecionar os requests, ou seja, o aluno se autenticou e autorizou, então ele pode acessar todas as urls do microsserviço.

Pode usar de um *Decorator* para adicionar informações ao requests (inclusão de cabeçalhos, exclusão de infos desnecessárias, etc.).

Também pode limitar acesso ao conteúdo trafegado, ou seja, antes de mandar a resposta, retira algumas coisas da web quando algum cliente mobile solicitar (não é muito o papel do Gateway fazer isso, mas as vezes ele pode fazer).

Conhecimento: Facade

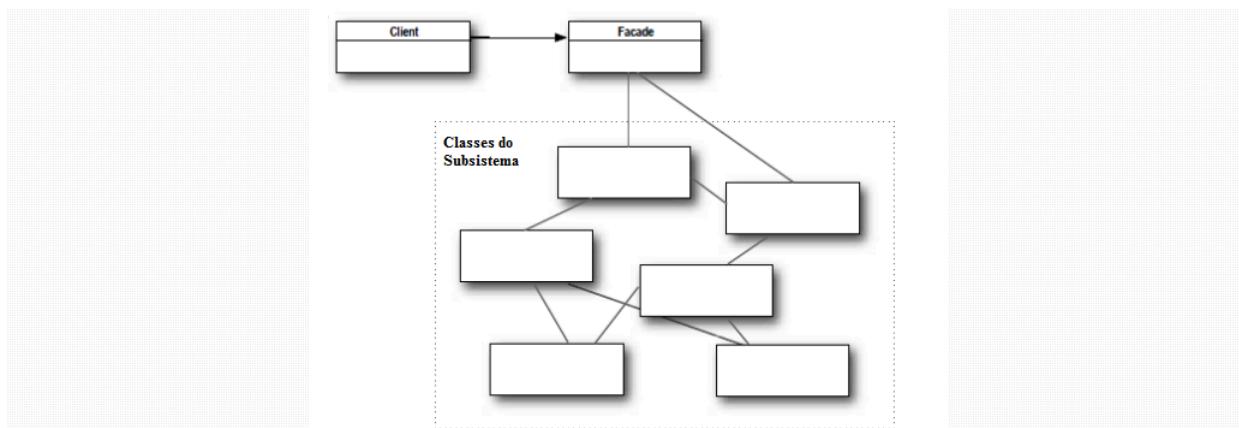


Figura 1: Diagrama de classe do Padrão Facade

No diagrama de classe acima tem-se o Client que é quem acessa a Facade que, por sua vez, é uma interface simplificada do subsistema, sendo esta unificada e fácil de ser utilizada pelo cliente. Abaixo da Facade tem-se as classes do sistema que podem ser chamados diretamente, mas estão sendo agrupados na Facade.

O gateway se parece com um padrão de código (design) que chama facade (fachada), ou seja, eu simplifico o acesso à uma API, a um sistema complexo, por exemplo, concentrando as requisições em um ponto único, em uma fachada.

Conhecimento: Gateway.

Vimos neste vídeo sobre um dos padrões mais famosos na hora de implementar microsserviços:
API Gateway.

Qual a principal vantagem deste padrão?

A

Garantir a integridade de todos os demais serviços



Ter um ponto único de acesso para nossa aplicação



Alternativa correta! Dessa forma podemos ter controles de acesso unificados, autenticação em ponto único, etc. Mas essa também é a principal desvantagem: o ponto único de falha

C

Facilitar a escrita do código de cada endpoint



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora vamos falar sobre Agregar processos.

Temos serviços de domínio e serviços de negócio.

Um serviço de negócio, agrupa vários serviços de domínio, para que tenhamos um processo completo.

E ainda podemos ter um agregador de processos (o que vamos ver aqui), que agrupa vários processos - é menos comum, mas pode acontecer em processos muito complexos, ou seja, processos que dependem de muitos outros processos.

Um agregador de processos agrupa serviços de negócio (e os serviços de negócio agrupam serviços de domínio).

Process aggregator pattern

- Serviços de negócio agrupam serviços de domínio
- *Process aggregators* agrupam serviços de negócio
- Agregadores fazem as chamadas para os serviços necessários e montam a resposta correta
- Pode (e deve) ter lógica de processamento



Temos esse último ponto - pode e deve ter a lógica de processamento - pelo fato de que ao agrupar vários serviços de negócio, cada serviço de negócio te dá uma resposta e não posso simplesmente mandar uma junção de todas essas respostas para o cliente, e sim fazer um processamento e montar uma resposta final.

Construindo um agregador

- Defina um novo modelo para representar os dados agrupados
- A partir deste modelo, pense na API que fornecerá as operações

Vamos acabar criando um novo domínio, um novo modelo para representar essa agregação.

Por exemplo, um domínio novo de renovação de matrícula - que chama os processos de matrícula e de analytics, para matricular a pessoa novamente e a pessoa saber o quanto estudou no ano anterior, quais cursos fez, etc.

Agora vamos falar sobre Gateways Específicos - podemos ter a necessidade de ter gateways específicos para cada um dos clientes e o nome desses Gateways é o Edge Pattern:

Edge pattern

- Gateway específico para determinado(s) cliente(s)
 - Foco nas necessidades reais de determinados clientes
-

Temos um tipo de serviço mais próximo de clientes... para esses casos, podemos ter gateways específicos para cada cliente, e isso é chamado de Edge Pattern.

Exemplo: posso ter uma empresa que fornece formulários para clientes. Para um cliente, preciso de um formulário igual, mas com uma pequena diferença. Eu posso ter um serviço na frente, um edge, que vai modificar esse formulário e devolver de acordo com a necessidade do cliente. Por isso é um Gateway Específico. Ao invés de mudar toda a lógica de negócios, eu posso mudar apenas o gateway do cliente e devolver as necessidades dele.

Exemplo: posso ter dois edges para a web alura e para o mobile alura e cada um devolve o front de acordo com a realidade de cada um.

Tem devs que trabalham apenas com edge services, ou seja, não tem um gateway único, um ponto central, temos gateways diferentes para determinados clientes.

É um BFF.

Para construir:

Construindo uma ponta

- Identifique o cliente e suas necessidades
- Construa contratos específicos para o cliente
- Modifique os dados que são transferidos para garantir a otimização do processo
- Existe a possibilidade de ter apenas Edges, e não Gateway



Edge services:

Aprendemos sobre o tipo de serviço conhecido como *Edge Service*, ou serviço de ponta.

Quando esse tipo de serviço é interessante?

A

Sempre que vou expor serviços a qualquer cliente



Quando clientes diferentes possuem necessidades diferentes



Alternativa correta! Se seu aplicativo móvel precisa de menos dados que a aplicação web, um serviço de ponta pode garantir que os dados extra não sejam enviados de forma desnecessária, por exemplo.

C

Somente quando quero realizar logs específicos de cada endpoint



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora vamos ver como lidar com dados de microsserviços:
Vamos ter um bd único para cada um? Vamos compartilhar?

Single service database

- Problema: Escalabilidade do serviço e do banco são fortemente relacionados
- Solução: Cada serviço (que precisar) terá seu próprio banco de dados



Vamos seguir o Single service database - um banco de dados é apenas para um microsserviço. Isso é porque a escalabilidade do serviço e do banco são fortemente relacionados. Ou seja, tenho 2 microsserviços: um para matrícula e outro para exibir os vídeos - o de exibir os vídeos vai receber muito mais requisições do que o da matrícula. Por isso, a máquina dele tem que ser bem maior. Consequentemente, o seu banco de dados também terá que ser bem maior. Para resolver essa correlação, usamos esse conceito de que cada serviço (que precisar) vai ter o seu próprio banco de dados.

Também podemos ter o Shared Service database:

Shared service database

- Problema: Às vezes precisamos centralizar os dados (até por motivos contratuais)
- Solução: Trate esse banco em cada serviço como se ele fosse separado.

Exemplo: os dados do aluno precisam estar no mesmo lugar que os dados financeiro do aluno (pagamento, etc.) - isso contratualmente. Então, o mesmo banco de dados usado pelo serviço de alunos será usado também pelo serviço financeiro.

A solução é tratar como se fossem bancos diferentes (a conexão será diferente, a forma de acesso é diferente, não temos um sidecar que acesse o banco), para que possamos ter o princípio de os

microsserviços serem independentes cumprido. Vamos ignorar o fato de que é o mesmo banco fisicamente, vamos fingir que são bancos separados.

Dessa forma, se no futuro fomos evoluir para usar bancos separados, vamos ter um impacto bem menor.

Vamos sempre buscar o Single service database, mas se não tiver como, usar esse padrão do Shared service database.

Vantagem principal de se usar bancos diferentes:

The screenshot shows a mobile application interface titled "Bancos diferentes". At the top, there is a back arrow icon and a blue button labeled "PRÓXIMA ATIVIDADE". Below the title, there is a question: "Entendemos neste vídeo que o ideal é que cada serviço tenha acesso a seu próprio banco de dados apenas. Qual a vantagem de termos bancos de dados diferentes para cada serviço que precisa de acesso a dados?"

Below the question, there are three options listed vertically:

- A** Praticidade
- B** Segurança
- C** Escalabilidade

The option **C** is highlighted with a green background and a checkmark icon. A callout box points to this option with the text: "Alternativa correta! Com cada serviço tendo seu próprio banco, a escalabilidade do serviço e banco pode ser feita em conjunto. Assim, serviços que recebem poucos acessos podem ter bancos menos potentes e mais baratos, e vice-versa."

Vamos falar agora sobre padrões de codificação: um específico de código que pode nos ajudar na parte de microsserviços (vamos falar superficialmente) -> CQRS.

CQRS (Command Query Responsibility Segregation)

"At its heart is the notion that you can use a different model to update information than the model you use to read information. For some situations, this separation can be valuable, but beware that for most systems CQRS adds risky complexity."

Podemos usar modelos diferentes para ler e escrever em um database - temos modelos de escrita e modelos de leitura.

Em algumas situações, podemos ter um banco de dados de escrita e outro de leitura e temos que ter também um tipo de sincronização entre eles.

Ex: temos muito mais pessoas vendo os cursos do que novos cursos sendo cadastrados - podemos ter um banco de leitura otimizado para receber consultas e outro banco mais fraquinho para cadastrar cursos e depois sincronizo os dois.

Essa ideia é muito facilitada quando utilizamos o padrão CQRS no momento de codificação.

CQRS

- Com leitura e escrita separados, cada parte pode realizar operações mais complexas
- O modelo de leitura pode ter informações agregadas de outros domínios
- O modelo de escrita pode ter dados sendo automaticamente gerados
- Aumenta (MUITO) a complexidade de um sistema



Mas é perigoso, porque isso aumenta a complexidade do sistema.

Em resumo: nesse padrão, temos 1 modelo de escrita e 1 modelo de leitura (uma classe que representa o que vai ser lido e outra classe que representa o que vai ser escrito) - dessa forma, posso agregar dados de outros domínios quando eu for ler e já trazer, posso gerar novos dados

quando for cadastrar. Fica bem mais fácil fazer essas agregações, mas aumenta muito a complexidade do código.

Conhecimento: CQRS

<https://www.youtube.com/watch?v=yd6V4w19iJU>



Para entender esse conceito, é importante entender o CQS:



Um método em uma classe deve ser apenas um comando (que modifica informações do sistema, faz alguma coisa no sistema e não devolve nada) ou uma consulta (apenas busca informações no sistema e não altera nada e devolve alguma coisa). Logo, um método nunca pode ser um comando e consulta ao mesmo tempo. E os métodos de comando, devem ser sempre void (não retornando nada e se algo acontecer, lança uma exceção).

O CQRS já é pensando em servidor. Os métodos serão comandos quando mudar algo no servidor ou consultas quando apenas busca algo do servidor.

Agora, vamos falar sobre Eventos.
Especialmente sobre Eventos Assíncronos.

Asynchronous eventing

- Determinados problemas NÃO PODEM ser resolvidos na hora (em tempo real)
- Um serviço emite um evento que será tratado em seu devido tempo
- Tecnologias como mensagerias e serviços de stream de dados brilham

Um exemplo desses problemas que não podem ser resolvidos na hora é: quando eu me matrículo na Alura, existe um processo de anti fraude, de verificação dos dados, garantir que o cartão é válido, etc. e isso pode ser um processo muito demorado.

E aí eu recebo a mensagem de que o pagamento está sendo processado. E é para isso que temos eventos assíncronos, ou seja, o serviço de matrícula emite um evento - que é o aluno realizou o pagamento - e outro serviço fica ouvindo esses eventos e faz todos os processamentos necessários. Mas, para isso, o cliente não ficou esperando, e por isso chamamos de eventos assíncronos.

Ferramentas como RabbitMQ, Kafka, são ferramentas muito utilizadas nesses cenários, para distribuição de mensagens assíncronas, etc.

Agora vamos falar sobre como fazer operações com microsserviços:

Falando sobre logs - que já são muito importantes, mas, quando falamos de microsserviços, a importância triplica dos logs.

Podemos ter um serviço específico para logs, podemos ter um sidecar de logs.

Mas como agregar logs de todos os microsserviços?

Agregando logs

- Formatos de log DEVEM ser compartilhados entre os serviços
- Uma taxonomia comum deve ser compartilhada
- Logs de monolitos são agregados por padrão. Com microsserviços o buraco é mais embaixo
- Parte da tarefa de agregação pode ser o parsing dos logs para categorizar corretamente



Podemos ter um formato padrão para que todos os microsserviços gerem logs com as mesmas características e seguindo o mesmo padrão, para que possamos unificar todos os logs de todos os serviços e ter um log completo da aplicação.

Temos um formato único para logs - todos tem que estar em xml, com a estrutura definida (por exemplo).

Além de compartilhar os formatos, os logs devem compartilhar a taxonomia, a classificação de logs, a organização de logs deve ser compartilhada. Por exemplo, o que é um log de erro, um log de warning, etc. Dessa forma, eu consigo pegar quantas vezes aconteceram erros em toda a minha aplicação.

E dessa forma, consigo ver todos os logs, de todos os microsserviços, e ter um log da aplicação toda de forma organizada.

Todos os logs de todos os microsserviços devem ser agregados em algum ponto onde possamos os enxergar com facilidade.

Um exemplo é mandar os logs para ferramentas específicas (ex: datadog, etc.). Dessa forma, lá, conseguimos ver os logs da aplicação inteira agregados.

Precisamos também rastrear as chamadas e é a partir de logs que fazemos isso:

Rastreando chamadas

- Uma parte importante de realizar logs é rastrear as chamadas de uma execução
- Devemos poder reconstruir uma operação a partir de um identificador
- Isso é o equivalente à *call stack* de um sistema monolítico
- Use padrões de *trace ID* para gerar os logs
- Use ferramentas de gerenciamento (APMs) para visualizar



Então, vamos ter por exemplo um edge service sendo chamado e gerando um trace ID para especificar que é aquela chamada que está sendo feitas para os microsserviços. Esse ID é passado para todos os microsserviços, que vão gerar os logs em cima desse ID. Dessa forma, no final, conseguimos gerar uma call stack (stack trace) - equivalente no monolito - para informar tudo o que foi chamado, cada serviço, o que aconteceu, etc, conseguindo rastrear toda o funcionamento da aplicação.

Para visualizar os logs, vamos usar as ferramentas de APM (exemplo o DataDog).

Conhecimento: importância dobrada dos logs em Microsserviços.

Como já foi exaustivamente citado, os logs são muito importantes, seja em um sistema baseado em arquitetura de microsserviços ou não.

Por que logs são ainda mais importantes em microsserviços?



Pois sem eles não poderíamos rastrear as chamadas de uma execução.



Alternativa correta! Os logs são o que nos permitem montar uma espécie de *call stack* ou *stack trace*, ou seja, através de logs conseguimos reproduzir uma execução e depurá-la.

B

Essa afirmativa não é verdadeira. Logs não são tão importantes em microsserviços.



C

Pois eles são a única forma de termos métricas de acesso ao sistema.



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora, vamos falar sobre métricas.

Precisamos saber sempre do estado do nosso microsserviço - está recebendo muitas requisições, está consumindo muitos recursos? etc.

Também precisamos ter um status da nossa aplicação como um todo.

Para visualizar esses status, precisamos de métricas.

Agregando métricas

- Enquanto logs precisam de desenvolvimento, métricas "só" precisam de instrumentação
- Métricas nos permitem saber o que está acontecendo em determinado momento
- Construa ou use dashboards de alto nível para ter uma fácil visão do status atual da aplicação
- Depois, tenha dashboards específicos para cada serviço, com mais detalhes



Ou seja, as métricas só precisam de instrumentação, já temos ferramentas que fazem as métricas, só precisamos configurá-las. Isso não é trivial, demanda estudo, etc. Por isso o “só” está entre aspas. Mas não precisa de desenvolvimento, geralmente os servidores já nos informam esses números, isso já foi desenvolvido.

Permitem saber o que está acontecendo em momentos específicos - ou seja, ao meio dia, por exemplo, tenho mais requisições e chamadas para o meu servidor. Dessa forma, posso aumentar minha aplicação nesses horários de pico.

Precisamos ter as métricas para saber a saúde dos nossos sistemas. Elas são os exames de sangue dos seres humanos.

Terceiro curso: Microsserviços: explorando os conceitos

Como podemos arquitetar um microsserviço?

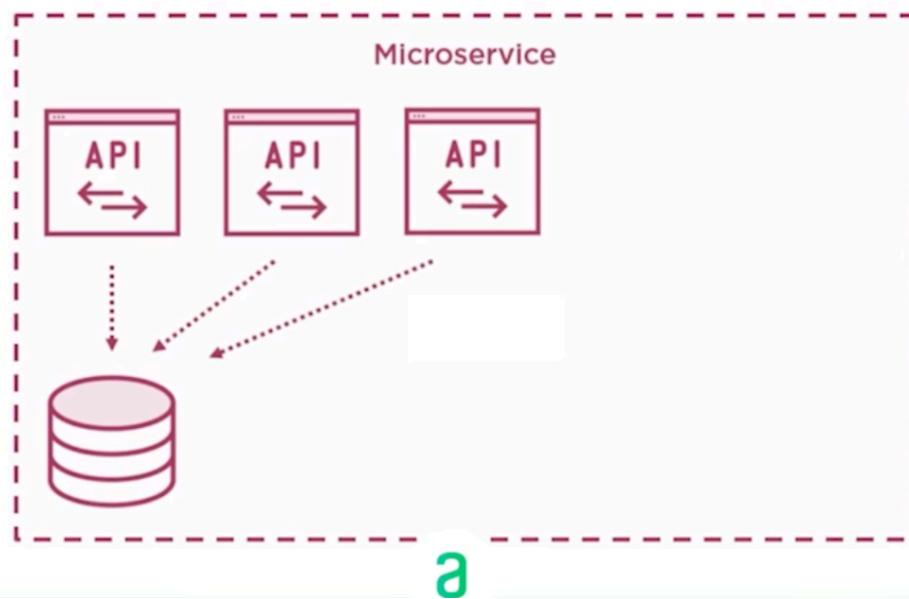
O que precisamos ter dentro de um microsserviço para que ele seja usual? Para que ele esteja completo?

De que é composto um microsserviço?

Sabemos que cada microsserviço deve ser o dono e gerenciar seus próprios dados. Então será que um microsserviço é um único processo rodando em um único servidor?

Respondendo a pergunta da imagem: Não!

Componentes de um microsserviço

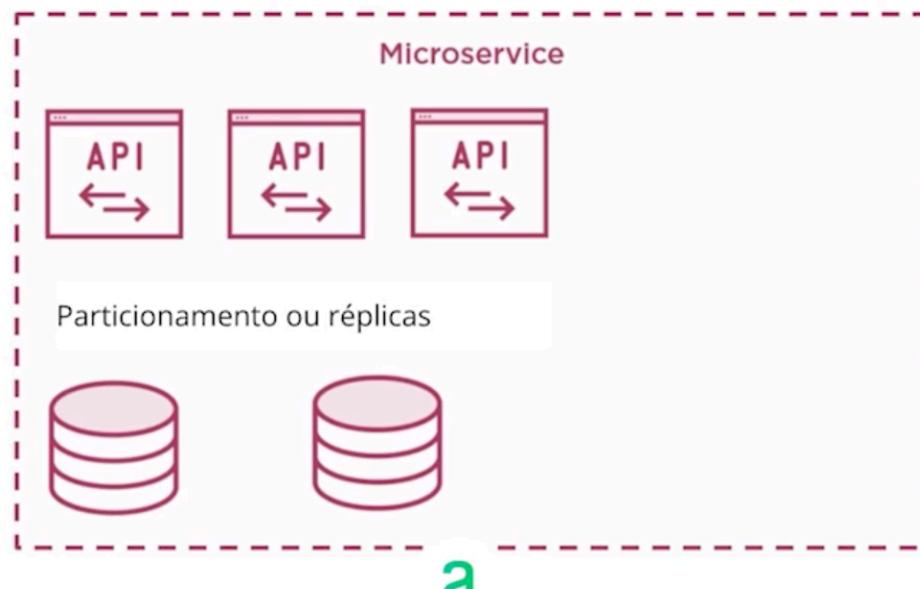


Só aqui, temos 4 servidores (3 para a aplicação e 1 para o banco de dados). Nesse caso eu escalei a aplicação para dividir as requisições entre os servidores, para não pesar muito cada um. Então, tenho um load balancer que vai controlar as requisições entre os servidores.

Mas mesmo que tivesse apenas 1 servidor de aplicação, já teria 2 servidores: 1 da aplicação e 1 do banco. E é importante separar os servidores, porque do banco vai precisar ter mais memória, enquanto que da aplicação precisará ter mais processamento.

Também posso ter particionamento de banco de dados - para melhorar escrita e leitura (CQRS) ou ter réplicas para manter backup e segurança:

Componentes de um microsserviço



Já teria mais um servidor então.

Ainda podemos ter mais um componente: um processador de mensagens, por exemplo, que poderia por exemplo processar imagens no final do dia, ou ficar escutando uma fila de mensagens e fazer os processamentos e ainda poderia ter algumas tarefas agendadas que são feitas em determinado horário do dia:

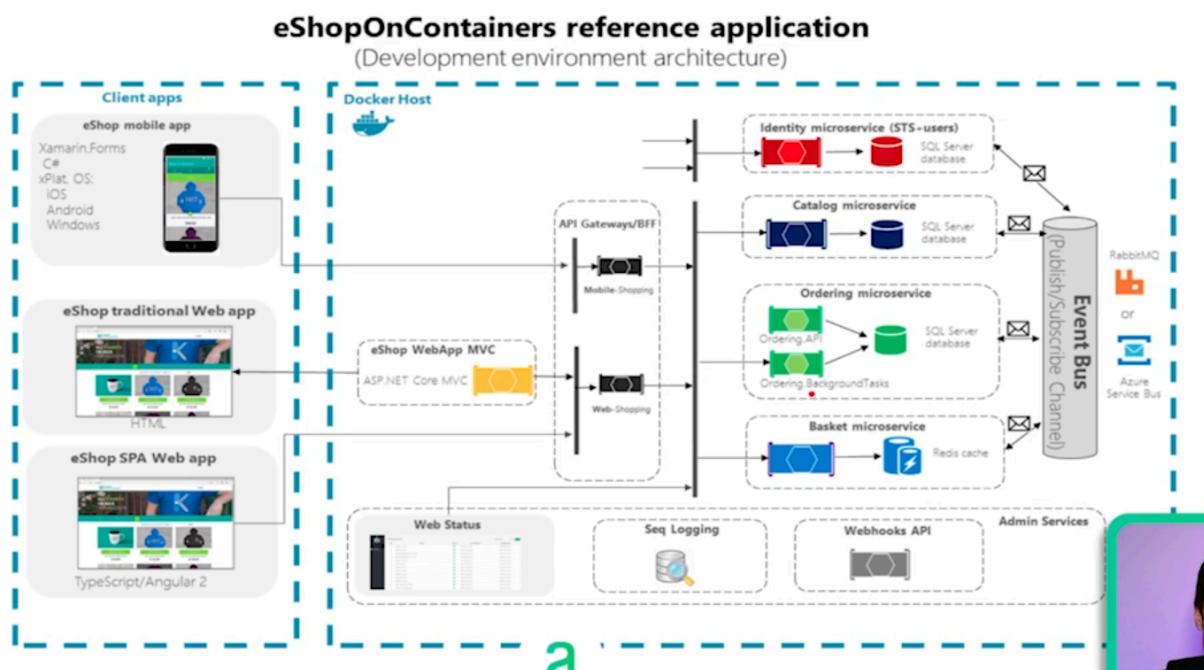
Componentes de um microsserviço



Só aqui teríamos 7 servidores.

Então, o microsserviço é muito mais que programação, é muito infraestrutura.

Um exemplo de microsserviço real é:



Olha o tanto de componentes nessa arquitetura. O microsserviço é composto por vários componentes.

Conhecimento: o que são componentes?

03 **Definição de componente** PRÓXIMA ATIVIDADE

Neste vídeo falamos bastante sobre os diferentes "componentes" de um microsserviço como API, banco de dados, processador de mensagens, etc.

O que significa um "componente" neste contexto?

A Uma aplicação, normalmente uma API. ⊗

B Um servidor, ou seja, uma máquina física ou virtual. ⊗

C Um servidor, uma aplicação ou infraestrutura de apoio. ⊗

Alternativa correta! Uma máquina (servidor) pode ser considerada um componente. Várias aplicações em uma mesma máquina podem ser vários componentes. Um serviço de apoio (como banco de dados ou fila de mensageria) pode ser um componente. Qualquer coisa que efetivamente componha o serviço, é um componente.

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Um microsserviço sozinho não traz tanto valor para o usuário final, ele é uma parte de uma aplicação maior, que ai sim vai trazer o determinado valor.

Então, precisamos ter comunicação entre os microsserviços.

Microsserviços são independentes

Um microsserviço expõe alguma forma de comunicação (uma API). Isso é o contrato entre este microsserviço e seus clientes.

Mas como manter minha API sempre a mesma se meu projeto precisa de atualizações e novas funcionalidades?

Eu tenho um contrato, uma interface com meus clientes e não pode ser quebrada. Mesmo se eu trocar de linguagem, de banco de dados, de qualquer coisa, eu tenho que garantir que meu contrato não vai mudar e eu vou continuar conseguindo essa comunicação com outras partes do meu sistema e com o cliente.

Como garantir a integridade desse contrato mesmo quando preciso fazer mudanças e evoluções?

Temos algumas técnicas:

- Se atualizamos alguma dependência, versão de linguagem, etc., isso deveria ser transparente para o cliente, e não afetá-lo.

Outras técnicas:

Microsserviços são independentes

- Apenas faça modificações aditivas
 - Novos endpoints
 - Novos campos (opcionais) em cada recurso
- Versionamento de APIs
 - Ao lançar uma v2, a v1 deve continuar funcionando, inalterada
- Manter equipes separadas, donas de cada serviço
 - A mesma equipe não vai alterar os clientes
 - Para adicionar funcionalidades que dependam de outros, solicitações formais podem ser feitas

Então, ao invés de modificar um endpoint, alterando o contrato que ele espera e devolve, eu crio um novo. Ex: agora, além de consultar um pedido, eu vou poder ver o status do pedido. Não vou

alterar o endpoint de pedido para me retornar o pedido com o status já, e sim criar um novo endpoint onde eu consiga visualizar o status. Sempre modificações aditivas.

Outro exemplo: agora para fazer um pedido, além de nome do produto, quantidade, etc., eu quero passar o desconto. Mas antes eu não precisava passar o desconto. Logo, esse campo tem que ser opcional, é obrigatório que seja opcional, para que quem já assinava esse contrato não seja prejudicado.

Também temos o versionamento de APIs - técnica já aprendida e muito utilizada. E as versões mais antigas precisam continuar funcionando, pelo menos por um tempo combinado!

Devemos ter equipes separadas também, donas de cada serviço, para que mantenham seus serviços em específico e caso o meu serviço dependa de uma alteração em outro, eu preciso enviar formalmente uma solicitação para essa alteração.

Essas são algumas regras para tornar o microsserviço independente... eles tem que ser independentes, mas devem respeitar os contratos de entrada e saída que assinam.

Conhecimento: Qual é a verdadeira definição de API?

É comum usarmos o termo API (*Application Programming Interface*) sem darmos muita atenção para ele.

Qual das seguintes alternativas NÃO pode ser chamada de API?



A Uma URL.



Alternativa correta! Uma URL não é uma API. Ela é uma interface do mundo externo com um servidor, mas não necessariamente uma interface de programação, ou seja, ela pode não expor funcionalidade alguma.

B

Uma aplicação acessível a outras aplicações.



Alternativa errada! Se uma aplicação consegue se expor a outras, através de HTTP, por exemplo, provendo acesso a funcionalidades, podemos chamá-la de API. Essa aplicação fornece uma interface de programação para outras.

C

Um método público de uma classe.



Alternativa errada! Um método ou um conjunto de métodos públicos fazem parte da API de uma classe. Isso é o que chamamos de interface pública, ou seja, o que está acessível a partir de outras classes. É um termo pouco usado, porém correto.



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora, vamos identificar as barreiras de um microsserviço. Quando termina um e começa outro? Tenho pedidos e carrinho. Eles são separados em 2 microsserviços ou estão no mesmo?

Recordar é viver

Falamos no treinamento de padrões que o ideal é uma abordagem "monolith first".

Cada módulo pode ser separado em um microsserviço.

DDD pode nos ajudar muito. Bounded contexts podem virar microsserviços.

Para ajudar nessas decisões, é ideal começar no monolith first, porque dessa forma, eu consigo identificar os módulos do monolito e separá-los em microsserviços. Caso os módulos de pedido e carrinho não estejam separados, talvez faça sentido mantê-los no mesmo microsserviço.

Por isso é importante começar com monolito, porque conhecendo o meu domínio e solucionando os problemas dele, consigo separar em microsserviços depois.

O que evitar?

Uma prática comum é transformarmos todos os "substantivos" do sistema em microsserviços, mas isso pode gerar muitos *data services* e serviços anêmicos demais.

Conhecimento: estudos sobre DDD e Bounded Contexts

The screenshot shows a dark-themed course interface. At the top, there's a header with a plus icon, the number '07', the title 'Para saber mais: Estudos', and a 'PRÓXIMA ATIVIDADE' button. Below the header is a text area containing the following content:

Identificar barreiras entre serviços é uma tarefa árdua e infelizmente não há uma regra definitiva de como fazer isso. Porém há estudos que nos dão um bom norte sobre isso.

Uma aplicação que tenha uma boa arquitetura de software com certeza é mais fácil de separar. Para entender o que é arquitetura, você pode conferir este artigo: [O que é arquitetura](#).

Algo mais específico que nos ajuda a separar conceitos de domínio são os Contextos Delimitados (*Bounded Contexts*) do DDD (*Domain-Driven Design*). Aqui na Alura existem cursos sobre esse assunto:

- [PHP e Domain Driven Design: Apresentando os conceitos](#)
- [Java e Domain Driven Design: Apresentando os conceitos](#)

At the bottom of the text area are two buttons: 'DISCUTIR NO FÓRUM' with a speech bubble icon and 'PRÓXIMA ATIVIDADE' with a right-pointing arrow icon.

Link curso java:

<https://www.alura.com.br/curso-online-arquitetura-java-aplicacoes-domain-driven-design>

Agora vamos entender como realmente criar um microsserviço.

Precisamos cuidar do host onde nossos microsserviços estarão... precisamos lidar com os servidores, isso que significa.

Onde manter cada componente de um microsserviço

Já vimos que cada microsserviço pode ter mais de um componente. Onde podemos manter todo esse sistema de forma que facilite tanto o desenvolvimento quanto o deploy?

Precisamos organizar bem, para que cada componente dos microsserviços tenham seus lugares em específico, configurados, para facilitar no desenvolvimento e no deploy.

Temos algumas opções:

Máquinas virtuais

O uso de máquinas virtuais foi e ainda é muito comum. Podemos provisionar uma VM de forma automatizada e integrar várias VMs.

O custo de processamento é alto. Dificilmente um computador de desenvolvimento rodará tranquilamente dezenas de VMs.

Podemos ter IaCs (Infra como código) para rodar nessas máquinas virtuais e já subir toda a infra específica daquele serviço. Porém, imagina 7 máquinas virtuais rodando no pc de um dev. Isso vai onerar muito a máquina, e vai ficar muito caro.

No conceito de microsserviços, elas são mais usadas onde temos poucos servidores ou aplicações monolíticas.

Outra alternativa:

Sistemas em cloud

Sistemas de computação em nuvem estão cada dia mais famosos. Manter um ambiente de produção em cloud é relativamente simples hoje em dia.

Porém como ter um ambiente de desenvolvimento simplificado? Precisamos contratar uma máquina para cada dev?

É muito bom para ambientes produtivos, mas no caso de ambiente de dev, como podemos fazer? Uma instância de EC2 para cada dev? Fica muito caro!

Uma abordagem mais comum hoje em dia e mais utilizada, é a de containers:

Containers

Essa é hoje a opção mais recomendada. Um ambiente de desenvolvimento é facilmente criado usando containers e diversos serviços de cloud possuem "container hosts", ou seja, ambientes próprios para "subirmos" containers.

Temos várias ferramentas orquestradoras de containers: Docker, Kubernets, que orquestram quantos containers vai subir, quantas imagens de cada um vai subir, etc.

Aqui é muito mais leves que máquinas virtuais.

Então, os containers hoje são muito utilizados:

1. Para ambientes de dev, eles são facilmente configuráveis.
2. Para subir para PRD, vários serviços de cloud já conseguem ler a configuração desses containers ou até do orquestrador e subir todas as máquinas necessárias para sua aplicação a partir desses containers.

Então, utilizar containers é a melhor saída para microsserviços.

Conhecimento: VMs x Containers

02 VMs vs Containers

 PRÓXIMA ATIVIDADE

Vimos neste vídeo diferentes formas de termos hosts para nossas aplicações como VMs, Clouds e Containers. Máquinas virtuais e Containers possuem muitas semelhanças.

Qual a principal diferença entre o uso de Containers e Máquinas Virtuais?



Consumo de recursos.



Alternativa correta! Containers conseguem compartilhar recursos com o sistema operacional host, enquanto cada máquina virtual é um novo sistema operacional. Isso exige muito menos recursos.

B

Não há diferença. São sinônimos.



C

Facilidade de configuração.



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Agora vamos ver como criar um novo serviço:

Quais as etapas necessárias para criar um novo serviço?

Antes de qualquer coisa, precisamos configurar o repositório para versionarmos nosso código.

Aqui devemos nos perguntar se vamos optar por uma abordagem monorepo, por exemplo.

Integração e entrega contínuas (CI e CD) são praticamente obrigatórias ao se trabalhar com microsserviços.

Devemos ter todo o processo automatizado e testes devem não só existir, mas devem ser confiáveis.

Um padrão sempre deve ser seguido para que todas as equipes estejam aptas a criar novos serviços.



Posso também automatizar a criação de um novo serviço, para já nascer com uma stack pré configurada, com ferramentas, dependências, etc. Nesse caso, eu perco um pouco em questão de flexibilidade (um serviço com uma linguagem, outro com outra, etc.), mas ganho na agilidade e padrão.

Vamos conversar sobre essa definição de padrão no momento de criar um novo serviço: é importante ter esse padrão para que qualquer pessoa consiga criar um novo serviço, sem dependências específicas com determinada pessoa ou equipe.

Padronizando a criação

Diversas tarefas devem ser realizadas de forma semelhante entre os serviços:

- Criação de logs
 - Formato
 - Destino
- Verificações de status (health checks)
- Monitoramento de métricas
- Busca por configuração e secrets



Precisamos ter um padrão na criação de logs, para que todos os logs de todos os serviços sejam padronizados e no final consigamos fazer uma agregação desses logs, para visualizar em uma ferramenta de logs.

Isso tudo precisa ser padronizado, e todos os microsserviços vão seguir essa abordagem.

Como fazer essa padronização?

Templates ou exemplos

Podemos ter um projeto "esqueleto" com tudo que qualquer microsserviço precisa.

Scripts de build, mínimo código necessário, etc.

Isso nos dá muita agilidade, mas tira um pouco da flexibilidade.



De novo, containers

Uma ótima forma de ter uma espécie de "template" e tendo uma imagem base para containers que conterão microsserviços.

A partir da imagem, só precisamos começar a codificar e rodar o código nos containers criados.

Podemos ter uma imagem que já tem todas as configs que um microsserviço precisa ter e vamos usar essa imagem como base para criar novos serviços.

Agora vamos ver sobre a comunicação entre serviços.

Como lidar com a comunicação entre serviços?

- Todo serviço pode se comunicar com qualquer outro?
- O front-end deve chamar os serviços diretamente?

Regras: *Não há regras*

Não existem regras, mas há algumas formas de se fazer para evitar problemas.

Alguns problemas que podem acontecer:

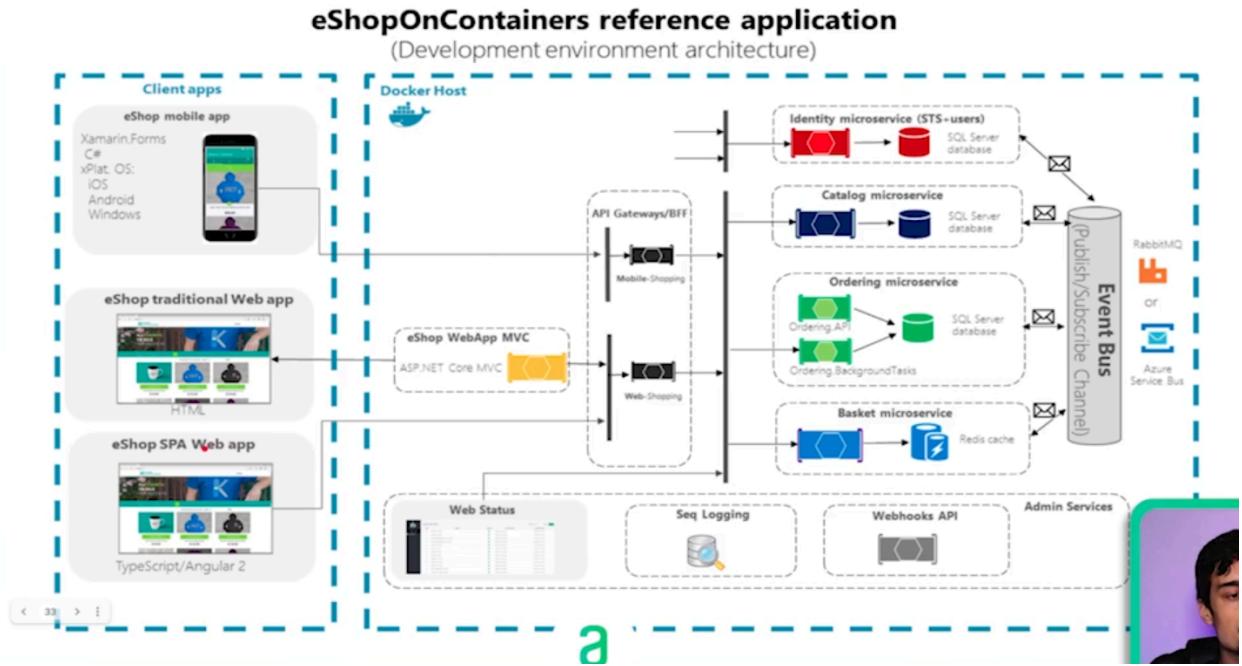
Possíveis problemas

- Dependências descontroladas
- Falhas em cascata
- Performance prejudicada

Então, não é muito legal um serviço chamar outro diretamente ou o front-end chamar diretamente os serviços.

Para isso, vamos usar Gateways.

Nota: BFF e Edge Services são a mesma coisa. Ou seja, é um API Gateway para cada cliente em específico. Isso é BFF, ou Edge Services.



Aqui, temos um exemplo de Edge Services/BFF, onde para o mobile tenho uma entrada e pela web tenho outra. Dessa forma, tenho pontos únicos de entrada, não estou chamando serviços diretamente. Então, conseguimos monitorar essas entradas e verificar se o que está chegando está certo.

Conhecimento: vantagens do API Gateway

The screenshot shows a digital learning platform interface. At the top, there's a navigation bar with three dots on the left, the text "02 API Gateway" in the center, and a blue button on the right with a circular arrow icon and the text "PRÓXIMA ATIVIDADE". Below the navigation bar, there's a dark grey header area with some text and a three-dot menu icon. The main content area has a white background and contains a question and three answer options (A, B, C). The question asks, "Vimos neste vídeo que se comunicar diretamente entre serviços pode nos trazer problemas (vamos falar mais sobre isso). Qual seria uma vantagem do uso de um API Gateway?" Below the question, option A is listed with a blue vertical bar, option B with a green vertical bar, and option C with a grey vertical bar. Option A has a note saying it's an incorrect alternative. Option B has a note saying it's a correct alternative. At the bottom, there are two buttons: "DISCUTIR NO FÓRUM" with a speech bubble icon and "PRÓXIMA ATIVIDADE" with a circular arrow icon.

Vimos neste vídeo que se comunicar diretamente entre serviços pode nos trazer problemas (vamos falar mais sobre isso). Qual seria uma vantagem do uso de um API Gateway?

A Podemos filtrar requisições em um ponto único.

Alternativa errada! É possível implementar autenticação, por exemplo, em um API Gateway ou backend-for-frontend (BFF, ou edge service). Podemos filtrar por IPs também ou fazer outras verificações de segurança.

B Podemos ter autenticação em nossa aplicação.

C Podemos monitorar toda a aplicação de um único ponto.

Alternativa correta! Através do API Gateway podemos monitorar acessos a nossa aplicação, podemos ter uma ideia geral de erros que estejam acontecendo, monitorar performance, etc.

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Vamos falar agora sobre comunicação.

Primeiramente, vamos falar da Síncrona.

É o que fazemos no início dos estudos da programação.. temos processos que dependem de uma resposta síncrona, ou seja, que ficam esperando respostas para continuar (fazemos o pedido e esperamos pela resposta).

Comunicação direta = atômica.

Comunicação direta

Diversos cenários nos obrigam a realizar "chamadas" e esperar por suas respostas.

Isso é o que conhecemos como comunicação síncrona.

Esse é o cenário de comunicação direta entre microsserviços. Ou seja, um microsserviço chama o outro e deve esperar a resposta do outro.

Como se comunicar

- HTTP
- gRPC
- Protocolos personalizados

A forma mais comum de se comunicar entre serviços é usando APIs, restfull... com o protocolo HTTP. Temos também o gRPC (com HTTP2 e HTTPS, transferência de dados binários e não em textos, tem performance e segurança, etc.) e outros personalizados.

Problema da abordagem

Ao realizar uma chamada direta para outro serviço e esperar sua resposta, problemas neste outro serviço nos afetarão diretamente.

Temos um alto acoplamento.

Talvez essa abordagem não seja a ideal... exemplo: vou fazer um pedido e para fazer esse pedido, temos várias etapas. Não seria interessante ficar esperando, pois tornaria o sistema bem lento e passível de falhas.

Para isso, temos a comunicação assíncrona:

Um clássico exemplo é a compra: imagina se quando eu apertasse o botão de comprar, a página só terminasse de carregar quando o produto estivesse em minhas mãos.

Não faz sentido!

Então, vamos seguir a abordagem da comunicação assíncrona. Ou seja, todo o processo de compra vai rodando em background e o que acontece é que meu sistema vai mandando notificações para o usuário final quando as etapas vão sendo vencidas.

Realizo o pedido e não espero a resposta.

Comunicação indireta

Existem cenários onde a resposta não precisa ser obtida imediatamente.

E como fazer isso?

Como se comunicar

- CQRS (background tasks)
- Eventos (mensageria)

Temos as técnicas para isso, ex: CQRS e Eventos.

Exemplo simples

Ao realizar uma compra:

- Dados precisam ser validados
- Pagamento deve ser processado
- Estoque deve ser separado
- Logística deve ser iniciada

Tudo isso é comunicação assíncrona... vou fazendo várias etapas de forma separada e só depois de todas as etapas feitas é que terei o meu pedido em mãos e o usuário será notificado. Mas antes de começar essas etapas, ele já recebeu a mensagem que o pedido foi feito, e está em fase de processamento.

Vimos nos últimos vídeos os 2 diferentes tipos de comunicação: síncrona e assíncrona. Cada tipo tem suas vantagens e desvantagens.

Quando usar comunicação síncrona e quando usar assíncrona, respectivamente?

A

Quando não ligamos para performance e quando precisamos de muita performance.

**B**

Quando podemos "ignorar" a resposta no momento e quando precisamos da resposta imediatamente.



Quando precisamos da resposta imediatamente e quando podemos "ignorar" a resposta no momento.



Alternativa correta! Se precisamos obter a resposta na hora antes de continuar o processamento, precisamos usar comunicação síncrona. Para exibir os dados de um curso, por exemplo, precisamos nos comunicar de forma síncrona com o banco para obter os dados e exibi-los. Já para enviar um e-mail, não precisamos ficar esperando a confirmação. Podemos enviar o pedido e ser notificados depois se deu certo ou não. Nestes cenários podemos usar comunicação assíncrona.



DISCUTIR NO FÓRUM



PRÓXIMA ATIVIDADE

Como lidamos com falhas? Tanto na síncrona como na assíncrona.

Como diz a lei de Murphy...

Microsserviços possuem operações intensas em rede. As probabilidades de falha são grandes.

Para lidar com falhas de comunicação síncrona:

Falhas em comunicação síncrona

- Circuit breaker
- Cache

Exemplo: serviço de catálogo precisa fazer uma comunicação síncrona com o serviço de carrinho.

Porém, o serviço de carrinho não está funcionando, então vai dar erro.

Posso tentar prevenir outras falhas e problemas de performance:

- Adiciono um proxy na frente do serviço de carrinho e chamo o proxy ao invés do serviço diretamente (ex: nginx). Ele vai passar essa requisição para o microsserviço e quando receber a resposta, perceber que é um erro, quando ele receber novamente uma requisição, ele não vai passar mais para o microsserviço, vai ser um circuito breaker, um curto circuito, porque o erro pode gerar outros erros, sobrecarregar servidor, pode derrubar a aplicação. Então o proxy não deixa a requisição passar e já devolve um erro. Ele, depois de um tempo configurável, vai deixar passar novamente e verificar a resposta, se for erro, abre o circuito novamente e tenta depois de um tempo maior.
- Posso usar cache também. No carrinho de compras, é muito comum usar o cache (em memória, disco rápido). Se o serviço estiver fora, mas se eu tiver um cache, eu consigo continuar fazendo minha aplicação funcionar.

Para lidar com falhas em comunicação assíncrona:

Falhas em comunicação assíncrona

- Simples Retry
- Retry com back-off
- Fila de mensagens mortas
- Mensagens devem poder ser lidas fora de ordem
- Mensagens devem poder ser recebidas repetidamente (idempotência)

- O simples retry, ele continua tentando na hora.
- O retry com back-off é parecido com um circuit-breaker, ou seja, ele dá uma segurada, espera um tempo e depois tenta novamente.
- Na fila de mensagens mortas (por exemplo um DLQ), temos um registro onde eu tenho todas as requisições que falharam. OBS: o retry com back-off, depois de algumas tentativas, manda a requisição para essa fila de mensagens mortas.
 - Serviços de mensageria: Kafka, RabbitMQ, etc. provavelmente já tem todas essas coisas implementadas, não precisamos nos preocupar.
- As mensagens tem que poder ser lidas fora de ordem, ou seja, as vezes chega a mensagem de que o pedido já foi separado antes da mensagem de confirmação do pagamento. Tenho que ver se a mensagem faz sentido naquele momento e se não fizer, dar uma segurada nela e depois aplicar um retry, por exemplo.
- As vezes podemos ter mensagens duplicadas... precisamos ter claro o conceito de idempotência, ou seja, se uma mensagem chega várias vezes, de forma duplicada, temos que processar apenas uma vez.

Como os serviços sabem onde estão cada um? Como se comunicar?
Vamos falar sobre Service Discovery - onde um serviço sabe onde encontrar o outro?

Ignorância é uma benção

Microsserviços podem estar na mesma rede ou em redes separadas, e cada serviço pode estar exposto por um IP.

Mas lidar diretamente com o IP de cada serviço pode trazer muitos problemas.

Temos um endereço para cada microsserviço, formado por um IP e uma porta. E eles podem estar até em países diferentes. Eles podem se comunicar através desses IPs e portas.

Mas, é difícil se comunicar apenas usando essa abordagem, porque pode trazer muitos problemas.. por exemplo, quando vou chamar o google, não chamo o IP dele, e sim o endereço dele.

Mas o google é exposto na internet, ou seja, registrei um domínio público para expor ele.

Mas não quero fazer isso com os meus microsserviços, nem sempre quero fazer isso. Quero ter isso de forma interna, e não externa, ou seja, apenas a minha aplicação vai saber onde estão os meus microsserviços.

Posso continuar usando o conceito de DNS, mas de forma privada e não pública:

DNS

Algo que já fazemos na internet, podemos fazer também em redes privadas:
registro de nomes.

DNS pode ser utilizado como service registry para sabermos como acessar cada serviço.

Um registro de serviço pode ter informações sobre quais processos ou máquinas estão de pé e sem falha.



O DNS é um registro de nomes (Domain Name System) - é um serviço que me dá nomes de domínio.

Posso informar pelo DNS que o IP 8.8.8.8 tem o nome de google.com, ou seja, sempre que acessar esse nome, vou cair nesse IP.

Podemos fazer isso do DNS para internet ou redes privadas.

Vários gerenciadores de containers (kubernets, Docker Compose) utilizam o DNS... vou fazer referência a outro container a partir do seu nome.

Fora da minha máquina, o nome não quer dizer nada... mas dentro da minha máquina, esse nome tem o significado dado a ele.

Nas nuvens também consigo configurar DNS internos.

Existem softwares específicos para service registry (registro de serviços).

Os softwares conseguem até verificar se determinado IP está de pé, se tiver, vai mandando requisições, se não, usa um backup, etc. O nginx pode fazer isso, podemos configurar um DNS para mandar as requisições para o load balancer o load balancer distribuir as requisições entre os servidores desses microsserviços.

08 **Para saber mais: Redes**

Em vários momentos foi citado que o estudo de microsserviços depende muito de infraestrutura e redes. Talvez até mais do que de programação em si.

O conceito de Service Discovery, por exemplo, via de regra não exige nenhum desenvolvimento. Precisamos apenas configurar servidores de DNS.

Por isso é super recomendado que os treinamentos de rede aqui da Alura sejam assistidos para uma melhor formação de arquitetura de microsserviços.

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Agora, falaremos sobre segurança.

Vamos falar sobre segurança no geral:

Segurança geral

- Segurança no transporte
 - HTTPS
- Segurança no repouso
 - Criptografia
 - Criptografia de disco
 - Bancos de dados cifrados
 - Criptografia em back-ups
 - Anonimização



Temos duas formas: segurança no transporte e segurança no repouso.

No transporte é o HTTPS, ou seja, todas as requisições, quando um gateway é chamado, quando um serviço é chamado, tudo isso temos as requisições criptografadas, ou seja, com a criptografia feita.

Na em repouso, precisamos criptografar os dados em banco de dados, em discos, em back-ups, temos que ter a anonimização também, caso alguém consiga quebrar a criptografia dos nossos dados, temos que fazer o máximo para que os dados conseguidos não façam sentido para quem quebrou essa criptografia.

Conhecimento: Segurança em transporte e destino

02 Segurança em transporte e no destino

 PRÓXIMA ATIVIDADE

Falamos neste vídeo sobre o conceito de segurança no transporte e segurança em repouso, ou no destino.

Qual das seguintes alternativas NÃO é uma tecnologia usada para segurança no destino?



A
TLS

Alternativa correta! TLS significa *Transport Layer Security* e é usada pelo protocolo HTTPS para garantir a segurança das mensagens.

B

Argon2

Alternativa errada! Argon2 é uma função de hash de senhas ganhadora de diversos prêmios. É uma das mais recomendadas para armazenamento de senhas cifradas.

C

Md5

Alternativa errada! Md5 é um algoritmo de hash muito utilizado para verificar integridade de arquivos transferidos na rede, por exemplo, ou após serem transferidos entre dispositivos para backup.

Vamos conversar sobre autenticação e autorização.

Em microsserviços, temos vários pontos que podem ser atacados, então temos que saber exatamente com quem estamos lidando.

Não devemos confiar em ninguém

Ter uma certa segurança (amplamente falando) não isenta nossa aplicação de lidar diretamente com esse assunto.

Qualquer pessoa que fizer uma requisição para a nossa aplicação deve ter nossa intenção.

Não podemos confiar no cliente que está fazendo a requisição.

Logo:

Autenticação

Cada requisição deve informar quem é o cliente. A partir dessa informação, nossa aplicação pode decidir se a operação será realizada ou não.

Em cada requisição, tem que ter a informação de quem é o cliente.

Técnicas de autenticação

- Basic HTTP
- Tokens (JWT)
- OAuth
- OpenID Connect

Basic - cabeçalho vai com usuário e senha em todas as requisições.

Tokens - manda usuário e senha uma vez só e geramos um token na aplicação e em todas as requisições, esse token deve ser mandado. O token é válido por um período de tempo configurável e se perdemos, podemos solicitar outro token.

OAuth - login a partir de certificados. Ex: fazer login na Alura através do Google, do Facebook. Delegamos a responsabilidade para alguém maior para fazer a autenticação. A minha aplicação não terá os dados do cliente, mas o Google, por exemplo, terá.

OpenID Connect - a partir de certificados também.

Autenticação vs Autorização

A **autenticação** nos permite saber quem está realizando determinada chamada.

A partir do processo de **autorização** decidiremos se a pessoa autenticada pode realizar tal ação.

A autenticação nos identifica, nos deixa entrar no sistema.

A autorização diz o que podemos fazer dentro do sistema, se podemos editar cursos, excluir, etc.

Técnicas de autorização

- ACL (Access control list)
- RBAC (Role-based access control)
- *On behalf of*

ACL - tenho listas mesmo. Listas de controle de acesso. Então, tenho várias listas, com várias permissões onde as pessoas que estão dentro de determinada lista podem realizar ações no sistema especificadas naquela lista.

RBAC - tenho papéis no sistema. Eu tenho o papel de aluno e instrutor.. então posso editar cursos porque tenho o papel de instrutor. Cada papel tem suas autorizações e podemos acumular papéis.

On behalf of - se o microsserviço de catálogo precisar se comunicar com o de carrinho, eu tenho que mandar as autorizações já realizadas aqui nesse microsserviço para o de carrinho. Então, eu microsserviço de catálogo estou me comunicando com você, microsserviço de carrinho em nome de João. Temos autorização entre vários microsserviços... ao fazer a comunicação entre microsserviços, também mando as informações de autorização.

Conhecimento:

Autenticação por tokens:

<https://www.youtube.com/watch?v=MZetkcs2xIo&t=86s>

JWT:

<https://www.youtube.com/watch?v=B-7e-ZpIWAs>

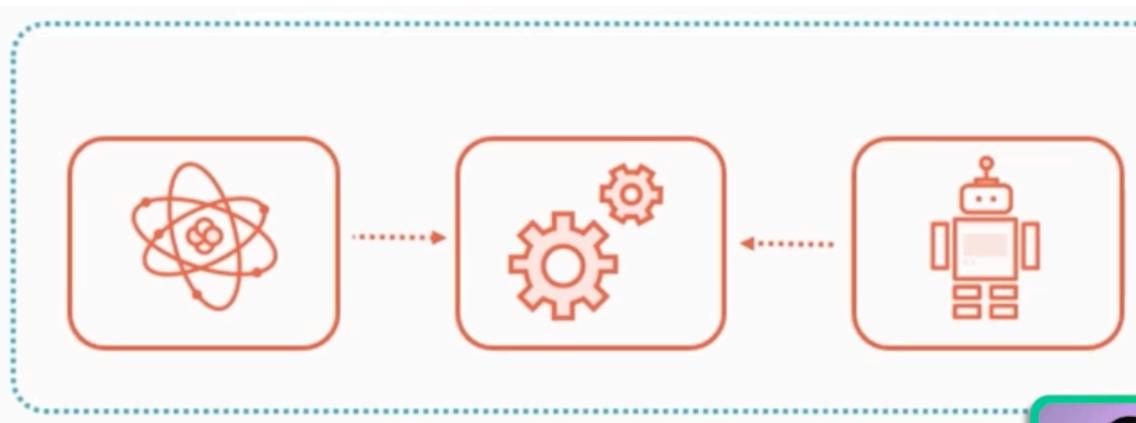
Temos que gerenciar também a segurança nas redes.

Nem só de programação vive uma aplicação

É de extrema importância que façamos uso de recursos de rede para manter nossa aplicação ainda mais segura, além do que já foi citado.

Supondo: temos 3 microsserviços que precisam se comunicar, mas não precisam se comunicar externamente, com serviços externos, somente entre si.

Dessa forma, podemos inserí-los em uma rede virtual:



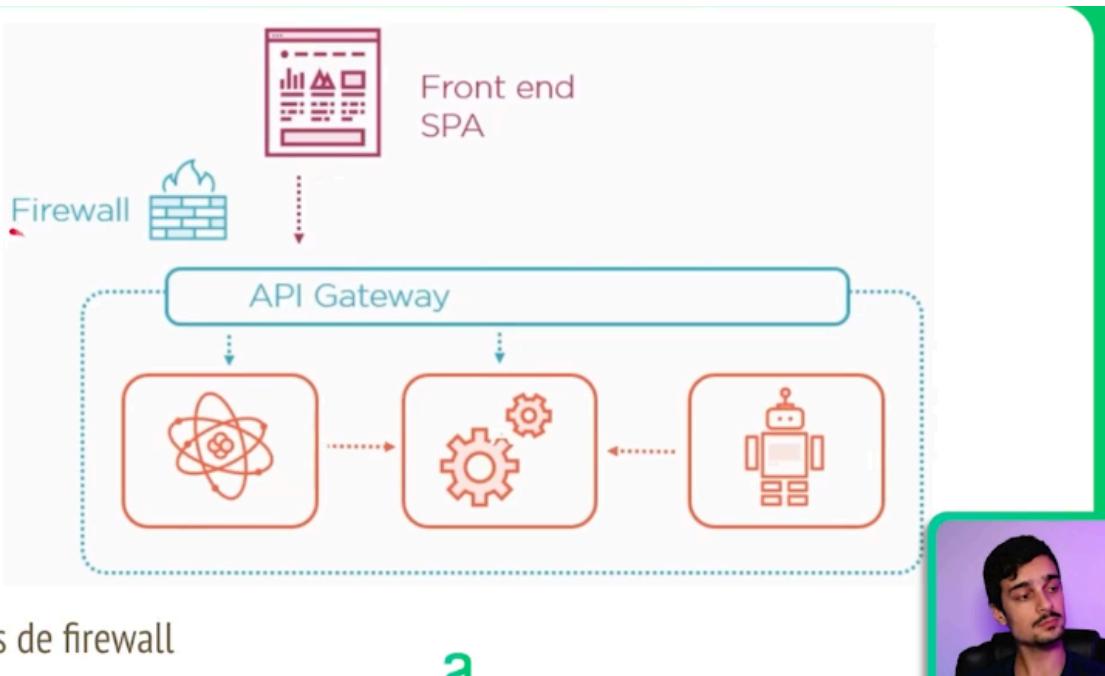
Redes virtuais



Dessa forma, permitimos conexões apenas da rede interna, e tudo que estiver fora da rede não vai nem chegar neles.

Já mitigamos muito o perigo de ataques.

E se microsserviços externos precisarem se comunicar com esses? Vamos usar o API Gateway e nele vamos inserir sistemas de Firewall:



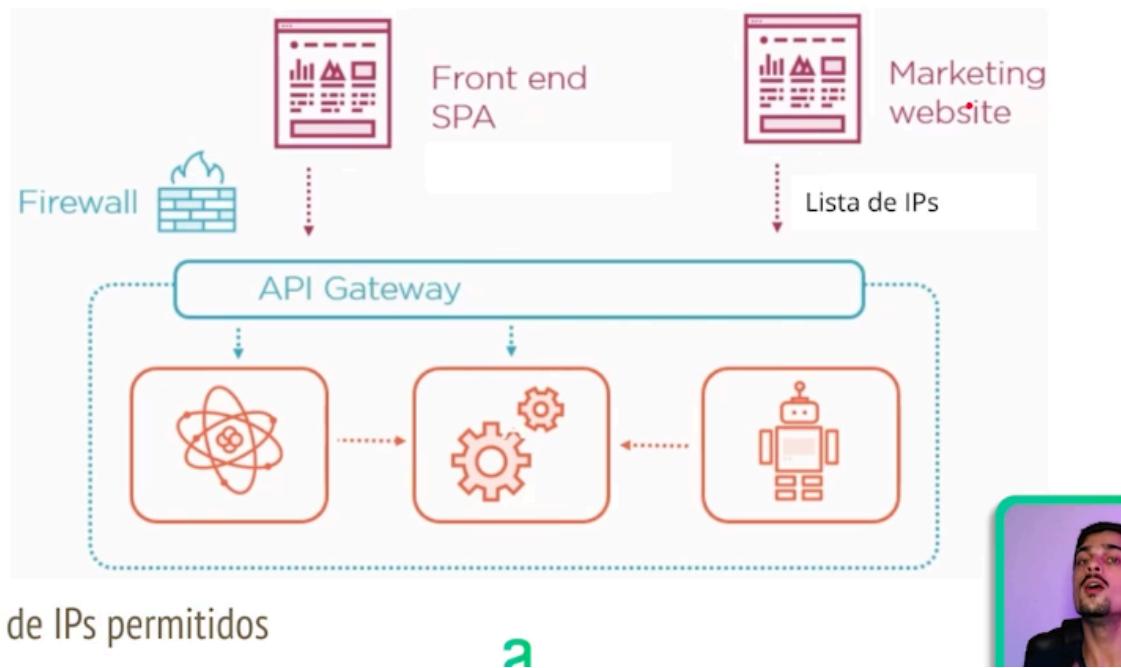
Sistemas de firewall



Podemos prevenir SQL Injections, diretamente no firewall, podemos evitar ataques distribuídos, pois teremos somente um ponto de entrada.

Outra técnica é fazer uma lista de IPs permitidos para acessar minha aplicação.

Ex: um site de admin, de marketing. Somente poderá receber requisições de alguns IPs em específico:



Lista de IPs permitidos



O conceito de VPN é esse, só permito acesso a determinados sites quando estiver em uma VPN específica, ou seja, estou restringindo a lista de IPs.

E podemos até usar todas essas práticas citadas em conjunto para que possamos proteger nossa aplicação.

Conhecimento: técnicas de segurança

The screenshot shows a mobile application interface for a learning activity. At the top, there is a navigation bar with a double-left arrow icon, the number '06' indicating the chapter or section, and a title 'Dentro ou fora da rede'. Below the title is a blue button with a right-pointing arrow and the text 'PRÓXIMA ATIVIDADE'. A vertical ellipsis icon is located on the far right of the header.

The main content area contains a text block: 'Segurança de redes é um assunto muito vasto e interessante. Neste vídeo cobrimos apenas uma pequena parte da superfície do que pode ser estudado.' Below this text is a question: 'Qual técnica está relacionada com segurança mas não é aplicada através de conceitos de redes?'.

The activity consists of three options, each with a letter label (A, B, C) and a corresponding text description. Option A is 'Bloqueio por IP.', option B is 'Anonimização.', and option C is 'Limite de acessos.'. Option B is marked with a green checkmark icon and is highlighted with a green vertical bar on the left. A callout box with a black border and white background is positioned over option B, containing the text: 'Alternativa correta! Anonimização faz parte da segurança em repouso, ou no destino.'.

At the bottom of the screen, there are two buttons: 'DISCUTIR NO FÓRUM' with a speech bubble icon and 'PRÓXIMA ATIVIDADE' with a right-pointing arrow icon.

Vamos falar agora sobre o conceito de “Defense in Depth” - que nada mais é do que usar várias técnicas de segurança, tanto em dados em tráfego, em repouso, etc, até limitando acesso dos serviços somente pelo API Gateway, lista de IPs, uso de Firewall, redes virtuais, etc.

Ou seja, eu utilizo de várias técnicas em conjunto para tentar amenizar o risco de ataque ao meu sistema.

Devemos dificultar ataques ao máximo

Todas as formas de segurança que foram citadas são muito importantes, mesmo que às vezes redundantes.

O conceito de Defense in Depth (defesa em profundidade) é esse, ou seja, se eu conseguir passar de alguma técnica de segurança, eu terei outra antes de chegar ao sistema, se conseguir passar da outra, terei outra, e assim por diante.

Tenho proteções combinadas.

Algumas outras opções de defesa:

Como me proteger mais?

- Atacantes (hackers) utilizam ferramentas modernas. Conheça essas ferramentas! Estude os possíveis ataques que podem afetar seu sistema;
- Tenha uma equipe de *infosec* e execute *pentests*;
- Automatize verificações de segurança. Faça requisições com certificados inválidos, usuários não autorizados, etc;
- Monitore e detecte ataques em tempo real;
- Tenha logs e audite os sistemas com frequência.



Vamos começar a falar agora sobre entrega - o deploy.

Muito trabalho

Um monolito pode ser entregue de forma manual, dependendo do caso. Um processo simples sendo executado com periodicidade pode atender alguns cenários.

Mas quando tratamos de microsserviços, isso se torna inviável!

Cada microsserviço tem várias infras, etc. Além de que teria que fazer o deploy de vários microsserviços, muito trabalho...

Se torna inviável.

Release pipeline

Uma release pipeline é uma linha de processos executados para gerar a entrega de um projeto.

Nesta pipeline podem estar processos de build, verificações, testes, etc.

Por isso é muito importante termos um processo de release pipeline, ou seja, de entrega, que pode ter etapas de build, verificações, testes, sonar, etc.

Ambientes diferentes?

O microsserviço pode ser implantado em vários ambientes:

- Desenvolvimento
- Staging / QA
 - Testes de performance
 - Smoke tests
- Homologação
- Produção
 - Por cliente
 - Por região



Precisamos ter a aplicação rodando em vários ambientes.

Desenvolvimento - testes - homologação - produção.

Precisamos então parametrizar as configurações:

Configurações parametrizadas

- Configurações do ambiente em si
 - Quantidade de recursos
 - Localização
- Configurações da aplicação
 - Destino de log
 - Dependências
 - Dados de acesso



Quantas instâncias vou ter, onde vou salvar os logs, o que vou usar de infraestrutura em cada ambiente.

Em ambientes não produtivos, eu não preciso ter uma infraestrutura muito parruda, com recursos caros. Posso economizar, trazer tecnologias mais baratas, mais simples, com menos complexidade.

Porém, em ambientes produtivos, precisamos ter tecnologias totalmente resilientes e parrudas.

Em cada ambiente, eu já tenho que subir as configurações parametrizadas para não precisar configurar na mão tudo isso.

Conhecimento:

The screenshot shows a mobile application interface. At the top, there is a navigation bar with a back arrow, the number '04' indicating the chapter, the title 'Parametrizar' in bold, and a blue button labeled 'PRÓXIMA ATIVIDADE' (Next Activity). Below the title, there is a video player area with a play button and a progress bar. To the right of the video player is a three-dot menu icon. The main content area contains a question and two answer options. A green vertical bar on the left side indicates the correct answer.

Neste vídeo falamos sobre a necessidade de termos configurações parametrizadas em cada ambiente de build.

Por que precisamos ter configurações parametrizadas em cada ambiente?

A Para podermos ter acesso a recursos específicos de cada ambiente sem tocar no código. (c)

Alternativa correta! Imagina que para nos conectarmos ao banco de dados tenhamos os dados de acesso direto no código. Vamos precisar mudar este código na hora do deploy e desfazer a alteração na hora do desenvolvimento. Isso não é nada prático. Serviços também podem ser diferentes em ambientes diferentes. Podemos usar "versões" menos robustas em ambientes de desenvolvimento, por exemplo.

B Não há motivo real para fazer isso. Somente um pouco mais de flexibilidade. (c)

C Para termos acesso a recursos externos que não podem ser acessados do código. (c)

Algumas estratégias agora para deploy - tudo ok, tudo testado, como colocar em produção?

Só falta botar no ar

Existem diversas estratégias para transformarmos uma entrega em um deploy, e várias etapas para cada estratégia.

Do que fazer deploy?

- Arquivo distribuível (.exe, .war, .phar, .zip)
- Tag do git
- Imagem de container

De novo, serviços são independentes

É importante que nossa pipeline possa realizar a entrega e deploy de microsserviços de forma isolada e não precisemos fazer o build de todos os serviços da aplicação

É importante que a pipeline gerencie os serviços de forma independente, porque cada um vai ter a sua particularidade. Dessa forma, temos que entregar separadamente cada um, com suas devidas particularidades e características.

Temos algumas estratégias de release:

Estratégias de releases

- Rolling upgrade
- Blue-green
- Feature toggle

Não podemos por exemplo parar a aplicação para inserir as modificações que tivemos em PRD. Ou seja, não podemos deixar a aplicação indisponível somente para subir as entregas.

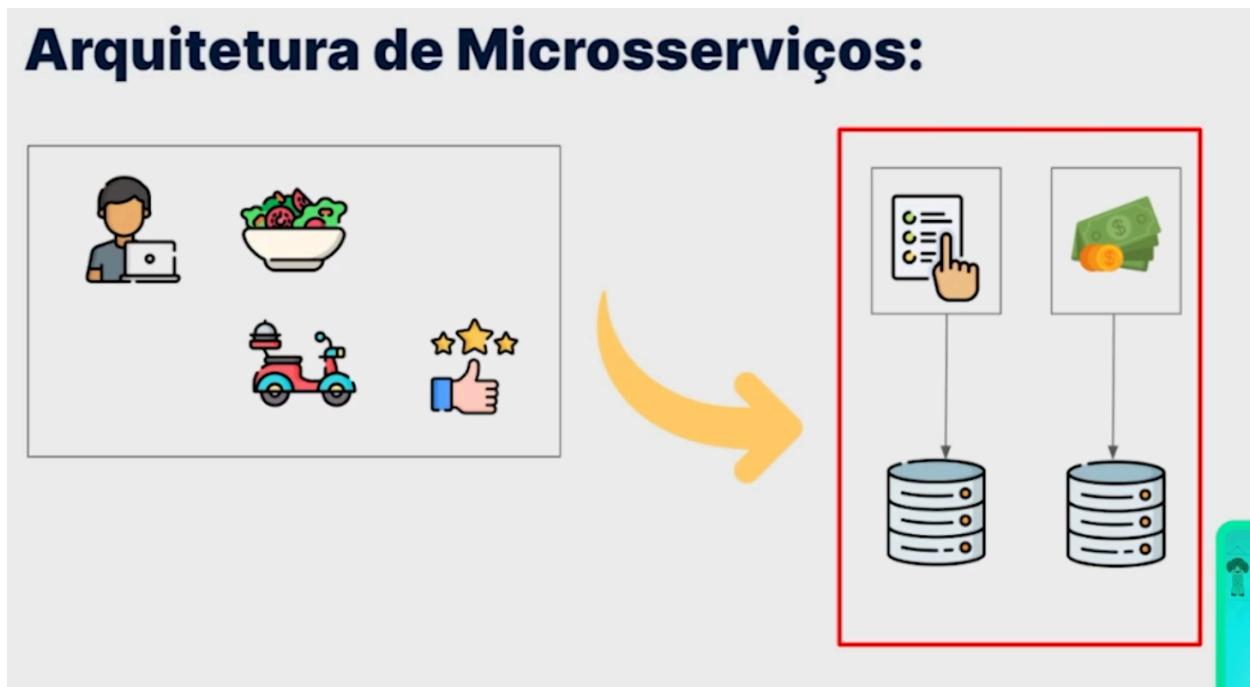
Portanto, temos algumas estratégias:

- **Rolling upgrade:** subir as alterações sem deixar o sistema indisponível. Ter uma atualização sem nenhum downtime, sem que a aplicação fique indisponível. Uma forma de fazer isso é: imaginando que temos 3 servidores de aplicação e o load balancer. Então podemos derrubar um servidor por vez e ir subindo a nova versão. Dessa forma, a aplicação não ficará indisponível. Mas precisamos ter as versões compatíveis uma com a outra para não termos nenhuma quebra de contrato.
- **Blue-green:** Temos um cluster com vários servidores. Posso subir um novo cluster com servidores, já com a nova versão e fazer com que o load balancer comece a apontar para este cluster. Dessa forma, deixo o cluster antigo ainda em caso de algum problema e vou monitorando a nova versão. Se não der nenhum problema, posso jogar o cluster antigo fora. Se der problema, voltamos para o cluster antigo.
- **Feature toggle:** aqui ativamos ou desativamos o acesso de determinada feature aos clientes. Então podemos começar com um número restrito de clientes acessando a feature, e se der tudo certo, depois libero para o restante dos clientes.

Existem outras estratégias de release, mas essas são algumas disponíveis.

Terceiro curso - Microsserviço na prática: implementando com Java e Spring.

Vamos trabalhar com o Alura Food:



Tínhamos um monolito e vamos transformar em 2 microsserviços - pagamento e pedido.
O que vamos ver?

3, 2, 1... Mão na massa



- **Criação da API do MS**
- **Uso de Migrations**
- **Service Discovery**
- **Balanceamento de carga**



3, 2, 1... Mão na massa



- API Gateway
- Comunicação síncrona
- Circuit breaker
- Fallback



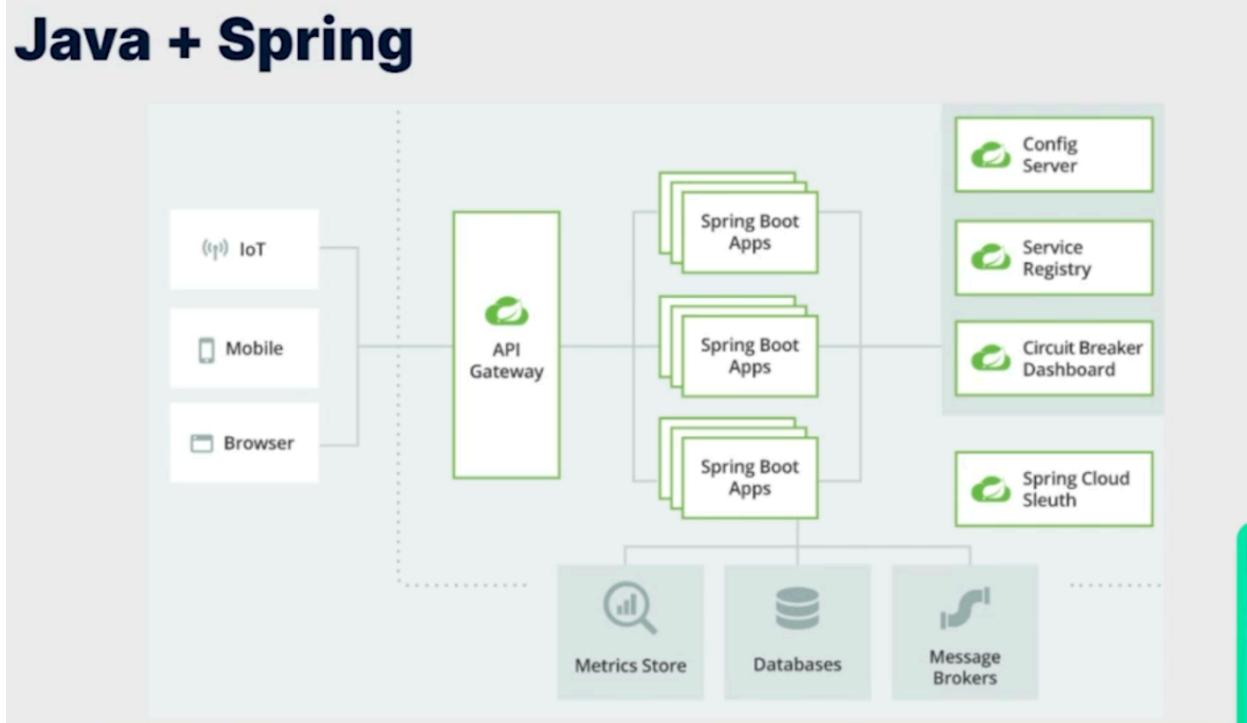
Não vamos abordar ainda...



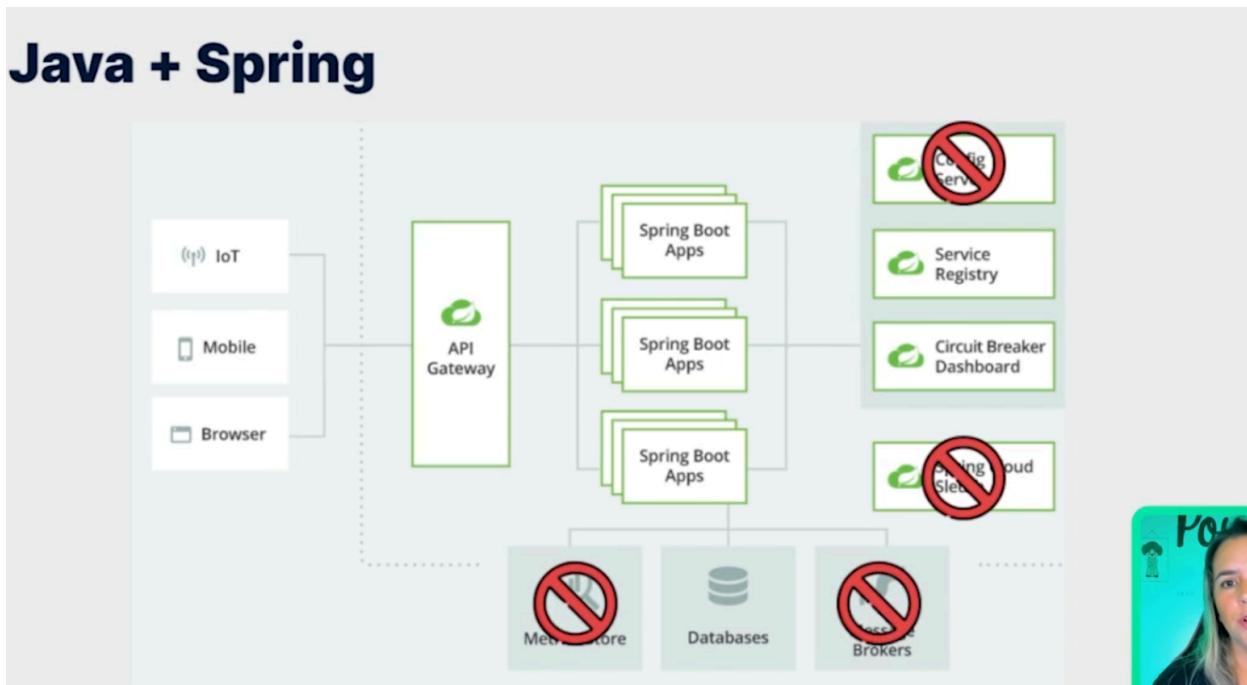
- Servidor de configuração
- Tracing distribuído
- Infra
- Segurança
- Comunicação assíncrona



Arquitetura da aplicação:



Java + Spring



Esses itens excluídos vão ficar para o próximo curso.

Relembrando as vantagens de uso do microsserviço ao invés do monolito:

Objetivos e necessidades:



- **Facilidade de mudança/manutenção**
- **Times menores e autônomos**
- **Reuso**

Objetivos e necessidades:



- **Diversidade tecnológica e experimentação**
- **Maior isolamento de falhas**
- **Escalabilidade independente e flexível**

Vamos trabalhar então com o Alura Food:

Em monolito, ele é:

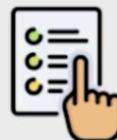
Funcionalidades



Administração



Restaurantes



Pedidos



Pagamento



Entregas



Avaliações

Dentro dessas funcionalidades, temos várias ações (o sistema foi crescendo):

Administração (Perfil):



- **Vários endereços para entrega**
- **Vários meios de pagamento**
- **Notificações e chat**
- **Programa Fidelidade**
- **Cupons**

Restaurantes (Estabelecimentos):



- **Dentro de um raio de distância**
- **Que meios de pagamento aceita**
- **Itens e valores**
- **Tipo de entrega**
- **Classificação**

Pedidos:



- **Pagamento confirmado**
- **Pedido aceito**
- **Pedido saiu para entrega**
- **Pedido concluído**
- **Problemas com o pedido**

Pagamentos:



- **Dinheiro**
- **Cartões (Débito, Crédito)**
- **Cartão corporativo**
- **Doações**

Entregas:



- **Própria**
- **Parceiros**
- **Repasso para o Entregador**
- **Gratificação para o Entregador**
- **Código de rastreio**

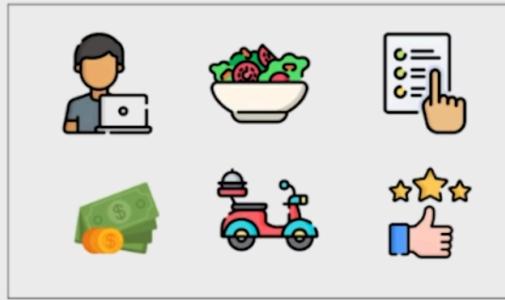
Avaliações:



- **Aplicativo**
- **Restaurante**
- **Pedido**
- **Entregador**

Agora queremos quebrar isso em microsserviços:

Monolito X Microsserviço:

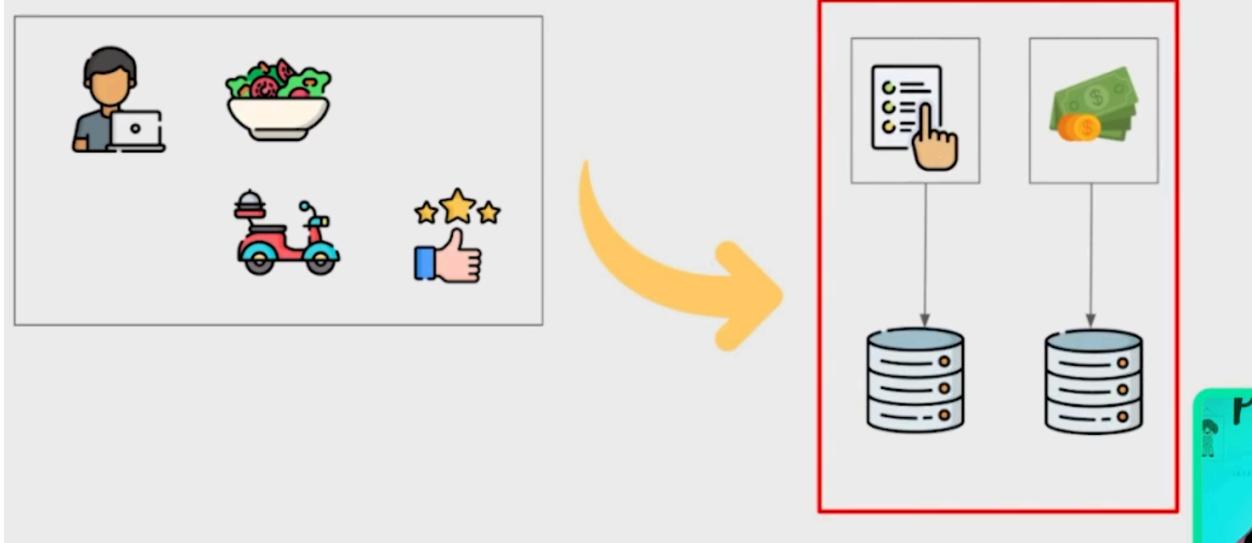


Não vamos quebrar tudo em microsserviços de uma só vez.

Temos que ir quebrando aos poucos.

Então, vamos ter que ir pegando os mais críticos e transformar em microsserviços:

Arquitetura de Microsserviços:



No nosso caso, vamos separar para microsserviços de pagamento e pedidos, nossas partes mais críticas.

Conhecimento:

O que são microsserviços?

<https://cursos.alura.com.br/extra/alura-mais/o-que-sao-microsservicos--c699>

Tipos de microsserviços?

<https://cursos.alura.com.br/extra/alura-mais/tipos-de-microservices-c698>

Martin Fowler artigos:

<https://martinfowler.com/articles/microservices.html>

Vamos criar o projeto no spring initializr com as seguintes dependências:

The screenshot shows the Spring Initializr web application. On the left, the 'Project' section is set to Maven, Java 17, and Spring Boot 3.3.4 (Snapshot). The 'Dependencies' sidebar on the right lists several optional dependencies: Spring Web (selected), Spring Data JPA, MySQL Driver, Spring Boot DevTools, Lombok, Validation, and Flyway Migration. At the bottom, there are 'GENERATE' and 'EXPLORE' buttons.

Project

- Gradle - Groovy
- Gradle - Kotlin
- Java
- Kotlin
- Groovy
- Maven

Spring Boot

- 3.4.0 (SNAPSHOT)
- 3.4.0 (M2)
- 3.3.4 (SNAPSHOT)
- 3.3.3
- 3.2.10 (SNAPSHOT)
- 3.2.9

Project Metadata

Group	br.com.alurafood		
Artifact	pagamentos		
Name	pagamentos		
Description	Módulo de pagamentos da AluraFood		
Package name	br.com.alurafood.pagamentos		
Packaging	<input checked="" type="radio"/> Jar	<input type="radio"/> War	
Java	<input checked="" type="radio"/> 22	<input type="radio"/> 21	<input checked="" type="radio"/> 17

Dependencies

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.

MySQL Driver SQL

MySQL JDBC driver.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Validation I/O

Bean Validation with Hibernate validator.

Flyway Migration SQL

Version control for your database so you can migrate from any version (incl. an empty database) to the latest version of the schema.

Buttons

- GENERATE CTRL + ⌘
- EXPLORE CTRL + SPACE
- SHARE...

Criando as classes do projeto:

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** The left sidebar displays the project structure under "pagamentos". It includes modules like ".idea", ".mvn", "src", "main", and "br.com.alurafood.pagamentos". Inside "br.com.alurafood.pagamentos", there are "model", "PagamentoApplication", "resources", "test", ".gitignore", "HELP.md", "mvnw", "mvnw.cmd", and "pom.xml".
- Code Editor:** The main window shows the file "Pagamento.java" with the following code:

```
13 import java.math.BigDecimal;
14
15 @Entity no usages
16 @Table(name = "pagamentos")
17 @Setter
18 @Setter
19 @AllArgsConstructor
20 @NoArgsConstructor
21 public class Pagamento {
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     private Long id;
26
27     @NotNull
28     @Positive
29     private BigDecimal valor;
30
31     @NotBlank
32     @Size(max = 100)
33     private String nome;
34
35     @NotBlank
36     @Size(max = 19)
37     private String numero;
38
39     @NotBlank
40     @Size(max = 7)
41     private String expiracao;
42
43     @NotBlank
44     @Size(min = 3, max = 3)
45     private String codigo;
46
47     @NotNull
48     @Enumerated(EnumType.STRING)
49     private Status status;
50
51     @NotNull
52     private Long pedidoid;
53
54     @NotNull
55     private Long formaDePagamentoId;
56 }
```

The code editor has syntax highlighting for Java and annotations. The status bar at the bottom indicates "53.5 LF LITE A".

Criei um pacote model (vou separar assim porque não faz sentido separar por nome de funcionalidade ou entidade - exemplo - ter um pacote pagamento e tudo de pagamento (enum, model, etc.) estar lá, porque como já é um microsserviço, já está separado. Então, vou usar a estratégia de ter a camada de model e tudo que for model fica aqui).

Então, criei a Entity de Pagamento e o Enum de Status.

Usei a anotação `positive` no valor para informar que deve ser positivo e a `size` é para estabelecer mínimo ou máximo (3 caracteres, 100 caracteres, etc.). No caso em que foi especificado tanto o valor mínimo quanto o máximo como sendo 3, quer dizer que quero que seja de 3 caracteres.

Conhecimento:

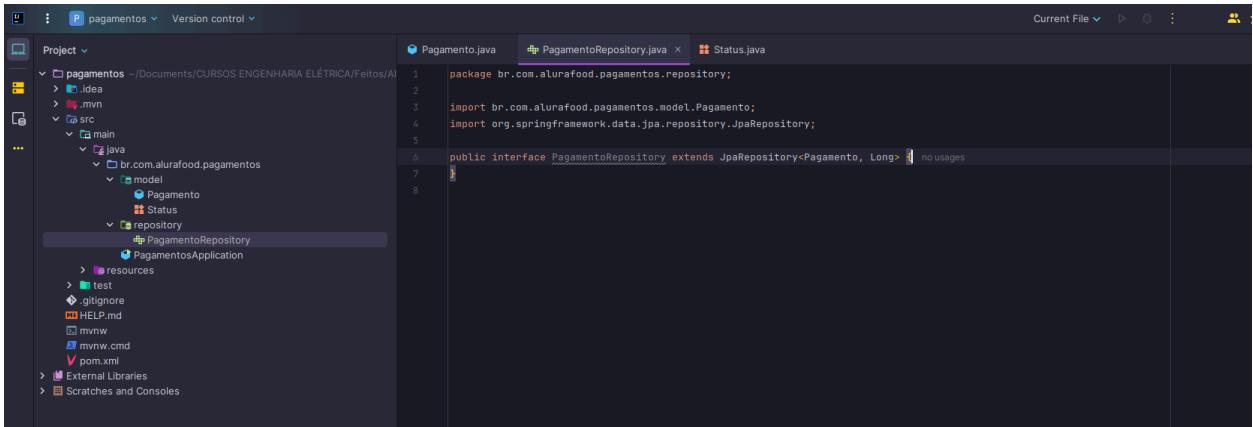
Boas práticas de desenvolvimento de uma API Rest:

<https://www.alura.com.br/artigos/rest-principios-e-boas-praticas>

Rest:

<https://cursos.alura.com.br/extra/alura-mais/o-que-e-rest--c119>

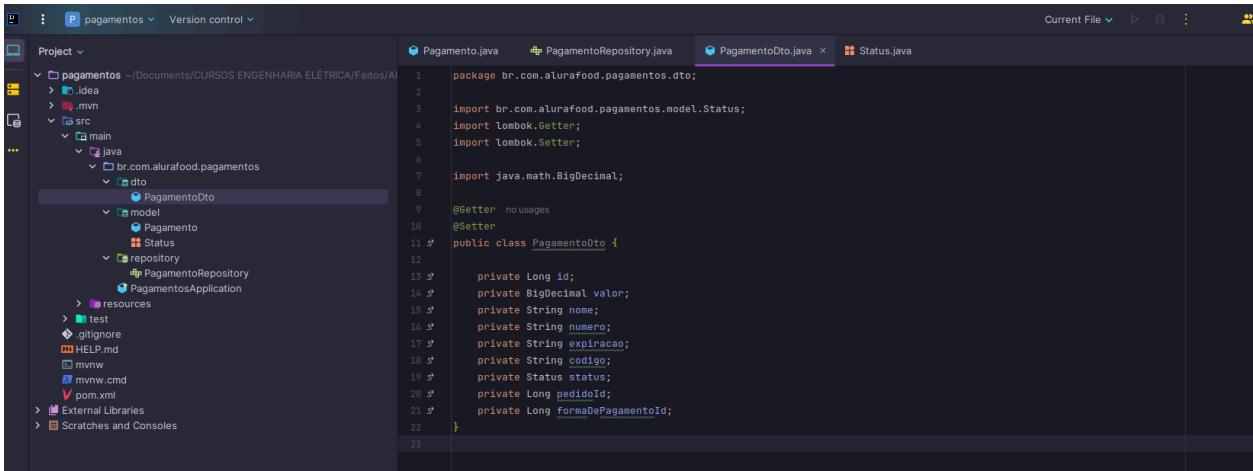
Criamos a repository também:



The screenshot shows the IntelliJ IDEA interface with the project 'pagamentos' open. The left sidebar displays the project structure, including 'src/main/java' which contains 'br.com.alurafood.pagamentos' with sub-packages 'model', 'repository', and 'application'. The 'repository' package contains a file named 'PagamentoRepository'. The right panel shows the code for this interface:

```
1 package br.com.alurafood.pagamentos.repository;
2
3 import br.com.alurafood.pagamentos.model.Pagamento;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface PagamentoRepository extends JpaRepository<Pagamento, Long> { }
```

Vamos trabalhar com DTO também:



The screenshot shows the IntelliJ IDEA interface with the project 'pagamentos' open. The left sidebar displays the project structure, including 'src/main/java' which contains 'br.com.alurafood.pagamentos' with sub-packages 'dto', 'model', and 'repository'. The 'dto' package contains a file named 'PagamentoDto'. The right panel shows the code for this class:

```
1 package br.com.alurafood.pagamentos.dto;
2
3 import br.com.alurafood.pagamentos.model.Status;
4 import lombok.Getter;
5 import lombok.Setter;
6
7 import java.math.BigDecimal;
8
9 @Getter @Setter
10 public class PagamentoDto {
11     private Long id;
12     private BigDecimal valor;
13     private String nome;
14     private String numero;
15     private String expiracao;
16     private String codigo;
17     private Status status;
18     private Long pedidoId;
19     private Long formaDePagamentoId;
20 }
```

Vamos ter esse DTO. Vou trabalhar com a lib Jackson para fazer serialização e desserialização de dados.

Conhecimento: DTO

The screenshot shows a learning interface with a dark theme. At the top, there's a navigation bar with a plus icon, the number '05', the title 'Para saber mais: objeto de transferência de dados', and a 'PRÓXIMA ATIVIDADE' button. Below the title, there's a text area containing the following content:

Nesta aula incluímos em nosso projeto a classe `PagamentoDto.java`, mas o que exatamente significa esse “DTO” e porque é interessante usá-lo?

Data Transfer Object (DTO) ou Objeto de Transferência de Dados, em português, é um padrão de projeto muito usado para o transporte de dados entre diferentes componentes de um sistema, diferentes instâncias ou processos de um sistema distribuído.

A ideia consiste basicamente em agrupar um conjunto de atributos em uma classe simples, de forma a otimizar a comunicação, bem como proteger para que alguns atributos não sejam serializados e devolvidos como resposta pelo nosso controlador, por exemplo.

Vamos imaginar uma classe de modelo que representa os produtos, na qual eu tenho vários atributos como descrição, valor, preço de custo, fornecedor, além de algumas regras de negócio e métodos. Ao devolver os dados para o controlador, não quero exibir todas essas informações, mas sim, somente alguns atributos, como código, descrição e valor. Nesse caso, não preciso devolver meu objeto de domínio, tenho a flexibilidade de criar uma nova classe, ou seja, um objeto de transferência de dados exclusivamente com esse dados e devolvê-lo na requisição, somente com esses atributos, protegendo assim o acesso aos outros.

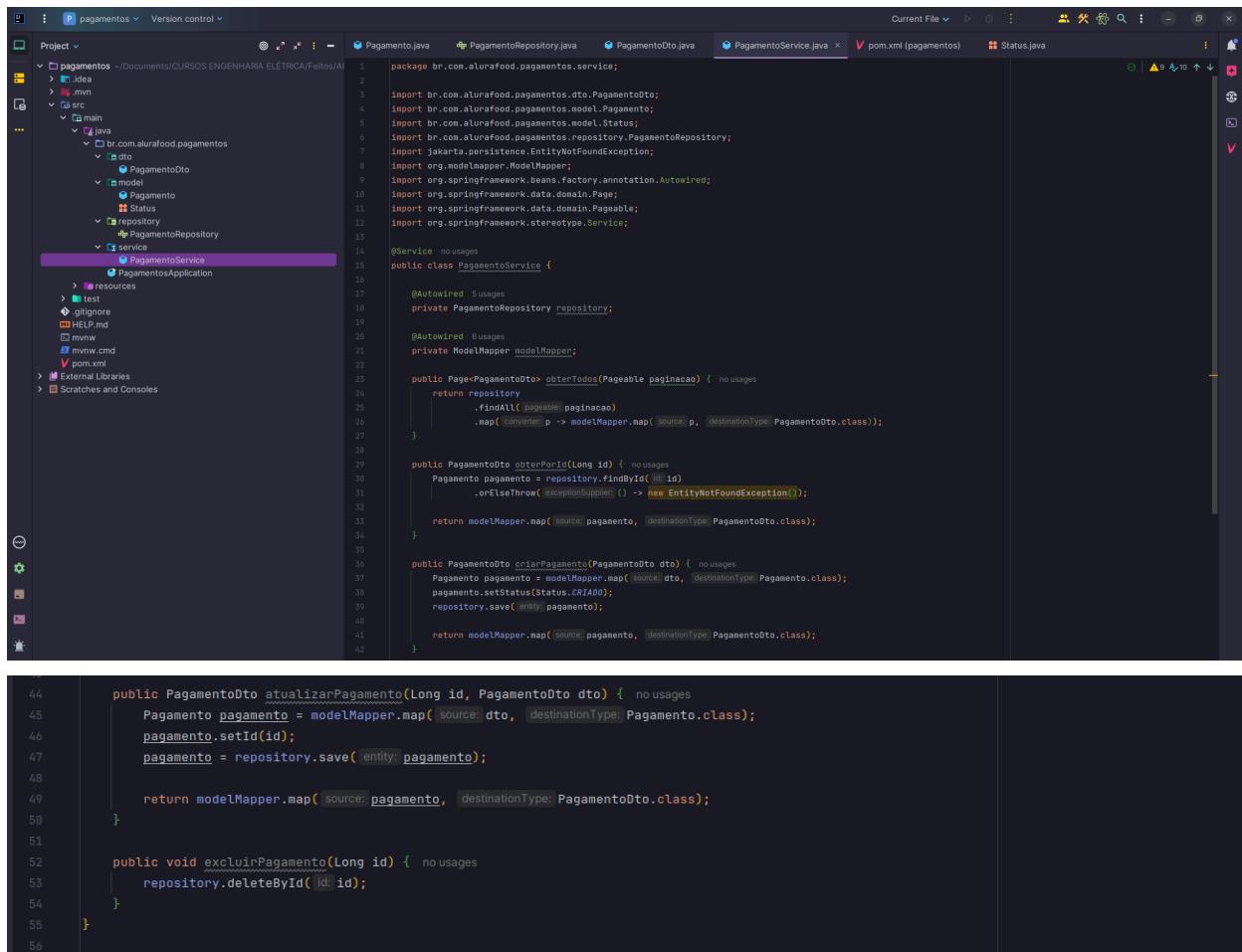
Em seu blog, Martin Fowler exemplifica o uso desse padrão, descrevendo-o como uma alternativa também para reduzir o número de chamadas a uma API, já que é possível unificar dados de diversos objetos de domínio. [Confira o artigo aqui.](#)

<https://martinfowler.com/eaaCatalog/dataTransferObject.html>

Vamos criar agora a classe Service (não é legal o meu Controller fazer operações, ele só recebe as requisições e delega).

Obs: vamos usar a classe ModelMapper para fazer as transferências de dados de forma facilidade entre minha entidade e meu DTO, para não precisarmos pegar dados de um objeto e setando no outro. Ela já faz isso para nós automaticamente, desde que os atributos tenham o mesmo nome. Tive que incluir uma dependência no pom.xml.

A service ficou da seguinte forma:



```
Project: pagamentos
File: PagamentoService.java

package br.com.alurafood.pagamentos.service;

import br.com.alurafood.pagamentos.dto.PagamentoDto;
import br.com.alurafood.pagamentos.model.Pagamento;
import br.com.alurafood.pagamentos.model.Status;
import br.com.alurafood.pagamentos.repository.PagamentoRepository;
import jakarta.persistence.EntityNotFoundException;
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.AnnotationConverter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

@Service
public class PagamentoService {

    @Autowired
    private PagamentoRepository repository;

    @Autowired
    private ModelMapper modelMapper;

    public Page<PagamentoDto> obterTodos(Pageable paginacao) {
        return repository
            .findAll(paginacao)
            .map(converter::map);
    }

    public PagamentoDto obterPorId(Long id) {
        Pagamento pagamento = repository.findById(id)
            .orElseThrow(exceptionSupplier() -> new EntityNotFoundException());
        return modelMapper.map(pagamento, PagamentoDto.class);
    }

    public PagamentoDto criarPagamento(PagamentoDto dto) {
        Pagamento pagamento = modelMapper.map(dto, Pagamento.class);
        pagamento.setStatus(Status.CRIADO);
        repository.save(emby(pagamento));
        return modelMapper.map(pagamento, PagamentoDto.class);
    }

    public PagamentoDto atualizarPagamento(Long id, PagamentoDto dto) {
        Pagamento pagamento = modelMapper.map(dto, Pagamento.class);
        pagamento.setId(id);
        pagamento = repository.save(emby(pagamento));
        return modelMapper.map(pagamento, PagamentoDto.class);
    }

    public void excluirPagamento(Long id) {
        repository.deleteById(id);
    }
}
```

Ou seja, eu tenho todos os métodos aqui de um CRUD... listar todos os pagamentos, listar por id um pagamento, criar um pagamento, atualizar um pagamento e excluir um pagamento.

No de listar todos, ele já volta com um Page, ou seja, já volto implementando um padrão muito bom de projeto.

Em todos eles vou usando o modelMapper para ir mapeando as classes entre si. Ou seja, se eu tenho os mesmos nomes de campo na Pagamento e na PagamentoDto, eu consigo mapear uma na outra apenas usando o método map desse modelMapper, passando o pagamento ou pagamentoDto que eu quero converter e a classe para a qual eu quero converter como parâmetro do método map.

No de atualizar pagamento, ele criou um novo pagamento, setou o id como o que já existia e salvou no banco. Ou seja, ao invés de pesquisar no banco e alterar alguma informação, nós

criamos um novo, setamos o id como o que já existia no banco e salvamos no banco. Dessa forma, ele sobrescreve o que estava no banco com as novas informações enviadas. O modelMapper ajuda bastante nessas operações.

Conhecimento:

Modelagem de APIs:

<https://cursos.alura.com.br/extra/alura-mais/boas-praticas-na-modelagem-de-api-s-rest-c1140>

Faça uso de recursos corretamente

- Use pronomes, e não verbos
- Não misture plural e singular

Exemplos:

Não use:

```
/getAllCars  
/createNewCar  
/deleteAllRedCars
```

```
/cars ao invés de /car  
/users ao invés de /user  
/products ao invés de /product  
/settings ao invés de /setting
```

Utilize sub-recursos

Relacionamentos existentes devem estar explícitos na URL.

Exemplos:

Retorna uma lista de motoristas do carro 711:

```
GET /cars/711/drivers
```

Retorna o motorista 4 do carro 711:

```
GET /cars/711/drivers/4
```

Entenda sobre idempotência

IDEMPOTENCE

WHEN PERFORMING AN OPERATION AGAIN GIVES THE SAME RESULT

HTTP METHOD	IDEMPOTENCE	SAFETY
GET	YES	YES
HEAD	YES	YES
PUT	YES	NO
DELETE	YES	NO
POST	NO	NO
PATCH	NO	NO



Não ignore os cabeçalhos HTTP

Existem várias informações que podem (e devem) ser transmitidas através de cabeçalhos.

- Content-Type
- Accept
- Cache

Aqui, é importante para falar como aceita, se é Json, XML, cache, etc.

Use aquele nome feio (HATEOAS)

Hypermedia As The Engine Of Application State é um assunto por si só, mas basicamente, facilite a interação com a API. Forneça links.

Exemplo:

```
{
  "cursos": [
    {
      "id": 1,
      "name": "C# (C Sharp)",
      "links": [
        {
          "type": "GET",
          "rel": "self",
          "uri": "api.treinaweb.com.br/cursos/1"
        },
        {
          "type": "GET",
          "rel": "curso_aulas",
          "uri": "api.treinaweb.com.br/cursos/1/aulas"
        },
        {
          "type": "DELETE",
          "rel": "curso_exclusao",
          "uri": "api.treinaweb.com.br/cursos/1"
        }
      ]
    }
  ]
}
```

Ou seja, eu informo o que o cliente consegue fazer com os cursos... dar um get, dar um delete, etc.

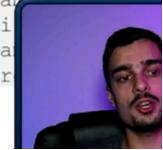
Filtro, ordenação, paginação e seleção de campos

Sua API deve ser flexível.

```
GET /cars?color=red  
GET /cars?seats<=2  
GET /cars?sort=-manufacturer,+model  
GET /cars?fields=manufacturer,model,id,color  
GET /cars?offset=10&limit=5
```

Link:

```
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=15&limit=5>; rel="next",  
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=50&limit=3>; rel="last",  
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=0&limit=5>; rel="first",  
<https://blog.mwaysolutions.com/sample/api/v1/cars?offset=5&limit=5>; rel="previous"
```



Versione sua API!

Alterações que demandam alteração no cliente
DEVEM gerar uma nova versão da API.

Exemplo:

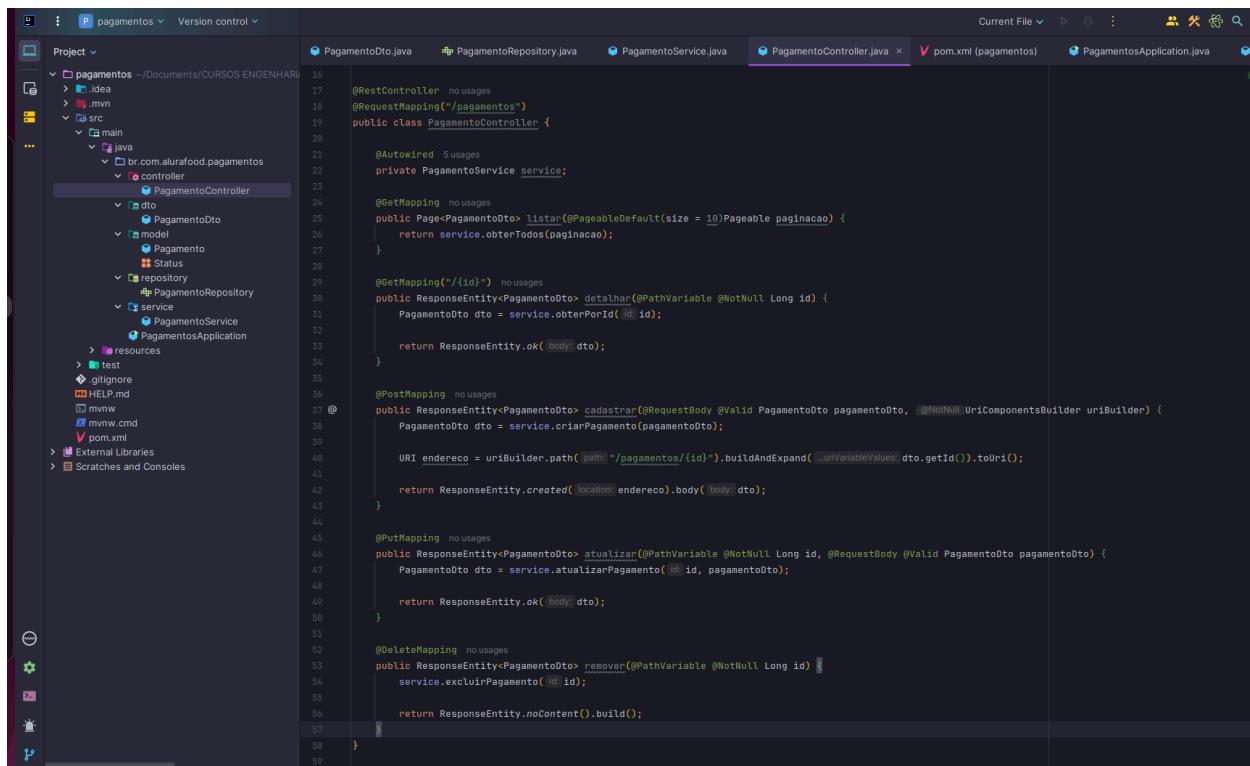
```
/blog/api/v1/posts?name=Titulo do post  
/blog/api/v2/posts?title=Titulo do post
```

Utilize os status HTTP corretamente

HTTP Status Codes



Criando a Controller:



The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "pagamentos". It contains a "src" directory with "main" and "java" sub-directories. The "java" directory has packages for "br.com.alurafood.pagamentos" (containing "controller", "dto", "model", "repository", "service", and "PagamentosApplication"), "resources", "test", and "mvnw".
- Code Editor:** The code editor displays the "PagamentoController.java" file. The code implements a REST controller for managing payments. It includes methods for listing all payments, getting a payment by ID, creating a new payment, updating an existing payment, and deleting a payment.
- Code Snippets:** Several annotations and methods are highlighted in blue, indicating they are part of the Spring framework's RESTful web services API.
- Toolbars and Status Bar:** The top bar shows tabs for other files like "PagamentoDto.java", "PagamentoRepository.java", "PagamentoService.java", and "pom.xml". The status bar at the bottom right shows "Current File" and other standard IDE icons.

Conhecimento: padrão MVC

<https://cursos.alura.com.br/extra/alura-mais/padrao-model-view-controller-c94>

Agora, vamos começar a pensar no banco de dados.

Vamos usar as migrations, o conceito de migrations (utilizando o Flyway Migrations), ou seja, tenho um histórico/evolução de tudo o que aconteceu no nosso projeto.

The screenshot shows a dark-themed IDE interface with a project tree on the left and several code files on the right. The project tree includes a 'pagamentos' folder containing 'src' and 'resources' subfolders. The 'src/main/java' folder contains packages for 'br.com.alurafood.pagamentos'. The 'resources/db.migration' folder contains a file named 'V1__criar_tabela_pagamentos.sql'. The code editor shows the contents of this SQL script, which creates a table named 'pagamentos' with various columns like id, valor, nome, numero, expiracao, codigo, status, forma_de_pagamento_id, and pedido_id. A red box highlights the entire SQL code in the editor.

Sempre vamos criar nessa pasta resources/db.migration/V{versão}__{nome_do_script}.sql.

É o Spring (Flyway) quem vai rodar esses comandos dessa pasta e criar os scripts de banco.

E vale ressaltar que se precisar alterar algo nesse script, não podemos editar esse arquivo, e sim criar um novo arquivo para editar o que for necessário.

Vamos agora criar o banco de dados:

The screenshot shows the same IDE interface with the 'application.properties' file open in the code editor. The file contains configuration for a MySQL database, specifically for the 'spring.datasource' section. A red box highlights the configuration properties: 'spring.application.name=pagamentos', 'spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver', 'spring.datasource.url=jdbc:mysql://localhost:3306/alurafood-pagamento?createDatabaseIfNotExist=true', 'spring.datasource.username=root', 'spring.datasource.password=Jaoavictor3001', and 'spring.jpa.show-sql=true'. The rest of the file contains other standard Spring Boot properties.

Cria essas propriedades no application.properties.

Vou inserir ali a propriedade para criar o banco alurafood-pagamento, caso ele não existir.

E agora vamos rodar a aplicação:

```
▶ □ | ⌂ ⌂ :  
Error starting ApplicationContext. To display the condition evaluation report re-run your application with 'debug' enabled.  
2024-09-11T22:51:02.552-03:00 ERROR 137157 --- [pagamentos] [ restartedMain] o.s.b.d.LoggingFailureAnalysisReporter :  
*****  
APPLICATION FAILED TO START  
*****  
Description:  
  
Field modelMapper in br.com.alurafood.pagamentos.service.PagamentoService required a bean of type 'org.modelmapper.ModelMapper' that could not be found.  
|  
The injection point has the following annotations:  
- @org.springframework.beans.factory.annotation.Autowired(required=true)  
  
Action:  
  
Consider defining a bean of type 'org.modelmapper.ModelMapper' in your configuration.  
  
Process finished with exit code 0
```

Deu erro no ModelMapper.

Conhecimento: Migrations

The screenshot shows a knowledge activity interface. At the top, there's a header with a menu icon, the title '03 Uso de Migrations', and a 'PRÓXIMA ATIVIDADE' button. Below the header is a question text: 'João, um dos desenvolvedores do Alura Food, viu no arquivo pom.xml do microsserviço de pedidos a dependência do **Flyway**. E aí ele levantou um questionamento interessante para o seu Tech Lead, que é como o **Flyway** identifica quais as *migrations* que já foram executadas no banco de dados?'.

The activity contains three options:

- A**: 'É necessário fazer um comando manualmente no banco para executar a *migration*'. This option is marked with a red circle and a minus sign.
- B**: 'Consultando a tabela de *migrations*'. This option is marked with a green circle and a checkmark.
- C**: 'O Flyway sempre executa todas as *migrations* no banco.'. This option is marked with a red circle and a minus sign.

Below the options is a note: 'Alternativa correta! Sim, conforme mostrei no último vídeo, o Flyway mantém uma tabela no banco (com o nome *flywayschemahistory*) para armazenar informações sobre as *migrations* executadas.'

At the bottom are two buttons: 'DISCUTIR NO FÓRUM' and 'PRÓXIMA ATIVIDADE'.

O que aconteceu com a ModelMapper foi que eu injetei ela na classe Service, mas ela não tinha um Bean criado... e preciso criar um Bean para ela, para que isso funcione.

Então, vamos criar:

The screenshot shows a Java project structure in an IDE. The 'src' directory contains a 'main' package with a 'java' subpackage. Inside 'java', there is a 'br.com.alurafood.pagamentos' package, which contains a 'config' package. Within 'config', there is a class named 'Configuracao'. A red box highlights this class. Another red box highlights the code within the class, specifically the @Bean annotation and the method 'obterModelMapper()' which returns a new ModelMapper().

```
package br.com.alurafood.pagamentos.config;

import org.modelmapper.ModelMapper;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class Configuracao {

    @Bean
    public ModelMapper obterModelMapper() {
        return new ModelMapper();
    }
}
```

Criei uma classe anotada como Configuration, onde os seus métodos devem estar anotados com @Bean.

E criei um método que sempre retorna um ModelMapper. Dessa forma, já vai parar de dar o erro e a aplicação vai subir.

The screenshot shows the terminal output of a Spring Boot application named 'PagamentosApplication'. The logs are timestamped from 2024-09-11T22:56:318-03:00 to 2024-09-11T22:57:03.203-03:00. The logs include various INFO messages from different components like DevTools, RepositoryConfigurationDelegate, and FlywayExecutor. A red box highlights the final log message: 'Tomcat started on port 8080 [http] with context path '/''. Another red box highlights the message 'Started PagamentosApplication in 7.523 seconds (process running for 8.352)' at the end.

```
2024-09-11T22:56:318-03:00 INFO 138808 --- [pagamentos] [ restartedMain] b.c.a.pagamentos.PagamentosApplication : No active profile set, falling back to 1 default profile: "default"
2024-09-11T22:56:356.387-03:00 INFO 138808 --- [pagamentos] [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2024-09-11T22:56:356.389-03:00 INFO 138808 --- [pagamentos] [ restartedMain] e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2024-09-11T22:56:357.836-03:00 INFO 138808 --- [pagamentos] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in 151 ms. Found 1 JPA repository interface.
2024-09-11T22:56:357.838-03:00 INFO 138808 --- [pagamentos] [ restartedMain] s.d.r.c.RepositoryConfigurationDelegate : Initializing Spring Data repository scanning in 151 ms. Found 1 JPA repository interface.
2024-09-11T22:56:357.839-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 ([http])
2024-09-11T22:56:357.840-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-09-11T22:56:357.841-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.28]
2024-09-11T22:56:357.842-03:00 INFO 138808 --- [pagamentos] [ restartedMain] w.c.WebServerApplicationContext : Initializing Spring embedded WebApplicationContext
2024-09-11T22:56:357.843-03:00 INFO 138808 --- [pagamentos] [ restartedMain] w.c.WebServerApplicationContext : Root WebApplicationContext: initialization completed in 2906 ms
2024-09-11T22:56:359.192-03:00 INFO 138808 --- [pagamentos] [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2024-09-11T22:56:359.216-03:00 INFO 138808 --- [pagamentos] [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@309560f9
2024-09-11T22:56:359.227-03:00 INFO 138808 --- [pagamentos] [ restartedMain] com.zaxxer.hikari.HikariPool : HikariPool-1 - Start completed.
2024-09-11T22:56:359.228-03:00 INFO 138808 --- [pagamentos] [ restartedMain] org.flywaydb.core.FlywayExecutor : Database: jdbc:mysql://localhost:3306/alurafood-pagamento (MySQL 8.0)
2024-09-11T22:56:359.229-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.f.core.internal.command.Upgrade : Successfully validated 1 migration (execution time 00:00:054s)
2024-09-11T22:56:359.230-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.f.core.internal.command.Upgrade : Current version of schema 'alurafood-pagamento': 1
2024-09-11T22:56:359.231-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.f.core.internal.command.Dbmigrate : Schema 'alurafood-pagamento' is up to date. No migration necessary.
2024-09-11T22:56:359.553-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.h..hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2024-09-11T22:56:359.624-03:00 INFO 138808 --- [pagamentos] [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.5.2.Final
2024-09-11T22:56:359.675-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.h.c.internal.RegionFactoryInitiator : HHH000020: Second-level cache disabled
2024-09-11T22:57:01.092-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2024-09-11T22:57:02.180-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.s.e.t.j.p.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform)
2024-09-11T22:57:02.182-03:00 INFO 138808 --- [pagamentos] [ restartedMain] j.l.localContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-09-11T22:57:02.632-03:00 WARN 138808 --- [pagamentos] [ restartedMain] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during live reload. LiveReload server is running on port 35729
2024-09-11T22:57:03.135-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : Tomcat started on port 8080 [http] with context path '/'
2024-09-11T22:57:03.181-03:00 INFO 138808 --- [pagamentos] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Started PagamentosApplication in 7.523 seconds (process running for 8.352)
```

Deu bom!

Agora, vamos testar a aplicação via Postman.

The screenshot shows a Postman request to 'http://localhost:8080/pagamentos'. The method is POST, and the body is a JSON object representing a payment. The response status is 201 Created, with a response time of 147 ms and a response size of 149 B. The response body is a JSON object with fields like id, valor, name, numero, expiracao, codigo, pedidoId, status, and formadePagamentoId. A red box highlights the JSON body in the request and the response body in the response tab.

```
1: {
2:     "valor": 500,
3:     "name": "Jacqueline",
4:     "numero": "12345678",
5:     "expiracao": "10/29",
6:     "codigo": "123",
7:     "pedidoId": 1,
8:     "formadePagamentoId": 1
9: }
```

```
1: {
2:     "id": 1,
3:     "valor": 500,
4:     "name": "Jacqueline",
5:     "numero": "12345678",
6:     "expiracao": "10/29",
7:     "codigo": "123",
8:     "status": "CRIADO",
9:     "pedidoId": 1,
10:    "formadePagamentoId": 1
11: }
```

201 Created 147 ms 149 B Just Now

Preview Headers (4) Cookies Timeline

NAME	VALUE
Location	http://localhost:8080/pagamentos/1
Content-Type	application/json
Transfer-Encoding	chunked
Date	Thu, 12 Sep 2024 02:05:40 GMT

Copy to Clipboard

Veio até o Location.

application File Edit View Window Tools Help

Q Search... Ctrl + P

Requisições

GET http://localhost:8080/pagamentos

Send 200 OK 12.4 ms 468 B

Preview Headers Cookies Timeline

```

1: {
2:   "content": [
3:     {
4:       "id": 1,
5:       "valor": 500.00,
6:       "nome": "Jacqueline",
7:       "numero": "12345678",
8:       "expiracao": "10/29",
9:       "codigos": "123",
10:      "status": "Aguardando",
11:      "pedidoId": 1,
12:      "formaDePagamentoId": 1
13:    }
14:  ],
15:  "pageable": {
16:    "pageSize": 0,
17:    "page": 0,
18:    "sort": {
19:      "sorted": false,
20:      "unsorted": true,
21:      "empty": true
22:    },
23:    "offset": 0,
24:    "paged": true,
25:    "unpaged": false
26:  },
27:  "totalPages": 1,
28:  "totalElements": 1,
29:  "last": true,
30:  "size": 10,
31:  "number": 0,
32:  "sort": {
33:    "sorted": false,
34:    "unsorted": true,
35:    "empty": true
36:  },
37:  "numberOfElements": 1,
38:  "first": true,
39:  "empty": false
40: }

```

Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

Introduction to Insomnia

Application File Edit View Window Tools Help

Q Search... Ctrl + P

Requisições

GET http://localhost:8080/pagamentos/1

Send 200 OK 31.1 ms 152 B

Preview Headers Cookies Timeline

```

1: {
2:   "id": 1,
3:   "valor": 500.00,
4:   "nome": "Jacqueline",
5:   "numero": "12345678",
6:   "expiracao": "10/29",
7:   "codigos": "123",
8:   "status": "Aguardando",
9:   "pedidoId": 1,
10:  "formaDePagamentoId": 1
11: }

```

Se eu tentar fazer um pagamento somente de dinheiro:

Vai dar erro.

Então, vou ter que mexer na tabela, e para isso que as migrations são de extrema importância, porque dessa forma, consigo mexer nas tabelas e ter o registro de histórico.

O delete e put tb funcionam:

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'File', 'Tools', and 'Help'. Below it, the main area has a header with 'DELETE ▾ http://localhost:8080/pagamentos/1' on the left, a 'Send ▾' button in a purple box, and a green box on the right containing '204 No Content', '266 ms', and '0 B'. To the right of the green box are tabs for 'Preview', 'Headers (1)', 'Cookies', and 'Timeline'. On the far left, there's a sidebar with icons for 'Parameters', 'Body ▾', 'Auth ▾', 'Headers (1)', and 'Docs'. The main content area below the header shows the message 'No body returned for response'.

The screenshot shows a POSTMAN interface with a successful API call. The URL is `http://localhost:8080/pagamentos/2`. The response status is `200 OK` with a `73.4 ms` latency and `146 B` transferred. The request body is a JSON object representing a payment. The response body is also a JSON object, identical to the request body, indicating the payment was created successfully.

```
PUT ▾ http://localhost:8080/pagamentos/2
Send ▾
200 OK
73.4 ms
146 B

Parameters JSON Auth Headers [2] Docs
Preview ▾ Headers [3] Cookies Timeline

1 v {
2   "valor": 999,
3   "nome": "Rodrigo",
4   "numero": "12345678",
5   "expiracao": "10/29",
6   "codigo": "123",
7   "pedidoId": 1,
8   "formaDePagamentoId": 1,
9   "status": "CRIADO"
10 }
11 }
```

Tive que passar também o status para atualizar o pedido (poderia ter sido melhor modelada a classe para não precisar passar, mas enfim...).

Conhecimento: injeção de dependência

<https://cursos.alura.com.br/extra/alura-mais/injecao-de-dependencias-o-que-e--c224>

Injeção de dependência

Pedindo ajuda pra fazer seu trabalho

Uma classe precisa de outra para funcionar.

Conhecimento: CURL

Acessando os métodos HTTP da minha API pelo terminal via curl.

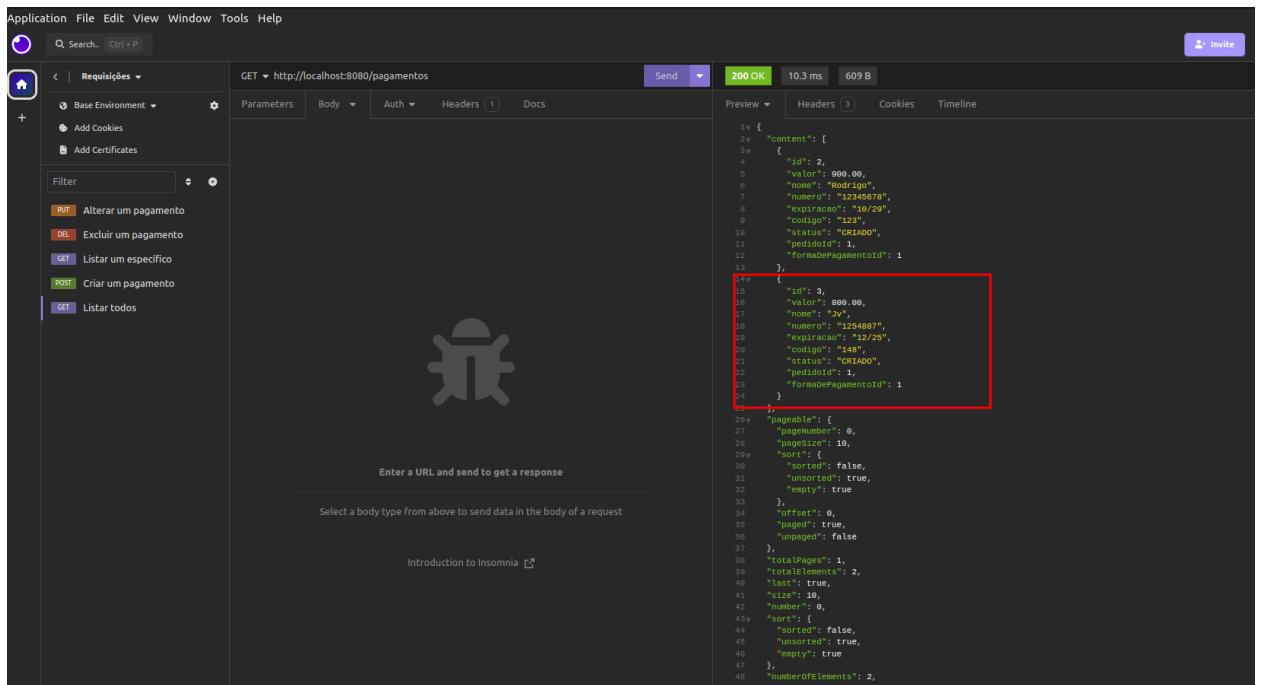
- Get

```
jooo in ~
→ curl -X GET http://localhost:8080/pagamentos
[{"content": [{"id": 2, "valor": 900.00, "nome": "Rodrigo", "numero": "12345678", "expiracao": "10/29", "codigo": "123", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}], "pageable": {"pageNumber": 0, "pageSize": 10, "sort": [{"sorted": false, "unsorted": true, "empty": true}], "offset": 0, "paged": true, "unpaged": false}, "totalPages": 1, "totalElements": 1, "last": true, "size": 10, "number": 0, "sort": [{"sorted": false, "unsorted": true, "empty": true}], "numberOfElements": 1, "first": true, "empty": false]
jooo in ~
→
```

- Post

```
jooo in ~
→ curl -X POST -H "Content-Type: application/json" -d '{"valor": 800, "nome": "Jv", "numero": "1254887", "expiracao": "12/25", "codigo": "148", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}' http://localhost:8080/pagamentos
{"id": 3, "valor": 800, "nome": "Jv", "numero": "1254887", "expiracao": "12/25", "codigo": "148", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}
jooo in ~
→
```

Ou seja, é a mesma coisa de mandar a requisição pelo Insomnia/Postman, só que pelo terminal, mandando via curl.



The screenshot shows the Insomnia REST Client interface. On the left, there's a sidebar with various actions like 'Alterar um pagamento', 'Excluir um pagamento', 'Listar um específico', 'Criar um pagamento', and 'Listar todos'. In the center, there's a 'Requisições' section with a dropdown for 'GET http://localhost:8080/pagamentos'. Below it, there are tabs for 'Parameters', 'Body', 'Auth', 'Headers', and 'Docs'. The 'Headers' tab shows 'Content-Type: application/json'. The 'Body' tab contains the JSON payload: {"valor": 800, "nome": "Jv", "numero": "1254887", "expiracao": "12/25", "codigo": "148", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}. The main area displays the API response with a status of 200 OK, a response time of 10.3 ms, and a size of 609 B. The response body is a JSON object with two items. The first item is highlighted with a red box, showing its details: id: 2, valor: 900.00, nome: 'Rodrigo', numero: '12345678', expiracao: '10/29', codigo: '123', status: 'CRIADO', pedidoId: 1, fornadePagamentoId: 1. The second item follows.

```
jooo in ~
→ curl -X GET http://localhost:8080/pagamentos
[{"content": [{"id": 2, "valor": 900.00, "nome": "Rodrigo", "numero": "12345678", "expiracao": "10/29", "codigo": "123", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}, {"id": 3, "valor": 800.00, "nome": "Jv", "numero": "1254887", "expiracao": "12/25", "codigo": "148", "status": "CRIADO", "pedidoId": 1, "fornadePagamentoId": 1}], "pageable": {"pageNumber": 0, "pageSize": 10, "sort": [{"sorted": false, "unsorted": true, "empty": true}], "offset": 0, "paged": true, "unpaged": false}, "totalPages": 1, "totalElements": 2, "last": true, "size": 10, "number": 0, "sort": [{"sorted": false, "unsorted": true, "empty": true}], "numberOfElements": 2, "first": true, "empty": false]
jooo in ~
→
```

Foi criado certinho!

O -H são os headers, o -d é o corpo (body) e no final é o endereço da API. No começo indico o verbo http e o -X é para especificar o verbo HTTP que vou mandar na requisição, que no caso é POST.

Tem mais vários outros parâmetros que podemos passar no curl, para baixar dados de uma API, etc... mas ele é basicamente para bater em uma API.

Por exemplo: chamando a API do ViaCEP:

```
joao in ~
→ curl -X GET https://viacep.com.br/ws/01001000/json/
{
  "cep": "01001-000",
  "logradouro": "Praça da Sé",
  "complemento": "lado ímpar",
  "unidade": "",
  "bairro": "Sé",
  "localidade": "São Paulo",
  "uf": "SP",
  "estado": "São Paulo",
  "regiao": "Sudeste",
  "ibge": "3550308",
  "gia": "1004",
  "ddd": "11",
  "siafi": "7107"
}%
joao in ~
```

```
joao in ~
→ curl -X GET https://viacep.com.br/ws/14790000/json/
{
  "cep": "14790-000",
  "logradouro": "",
  "complemento": "",
  "unidade": "",
  "bairro": "",
  "localidade": "Guairá",
  "uf": "SP",
  "estado": "São Paulo",
  "regiao": "Sudeste",
  "ibge": "3517406",
  "gia": "3220",
  "ddd": "17",
  "siafi": "6449"
}%
joao in ~
```

Vamos entrar agora na parte de Service Discovery.

Ele é como se fosse um catálogo de todas as instâncias, todos os microsserviços que estão registrados ali dentro.

É uma facilidade para não ter que ficar se preocupando com qual microsserviço eu devo chamar, qual o endereço, etc.

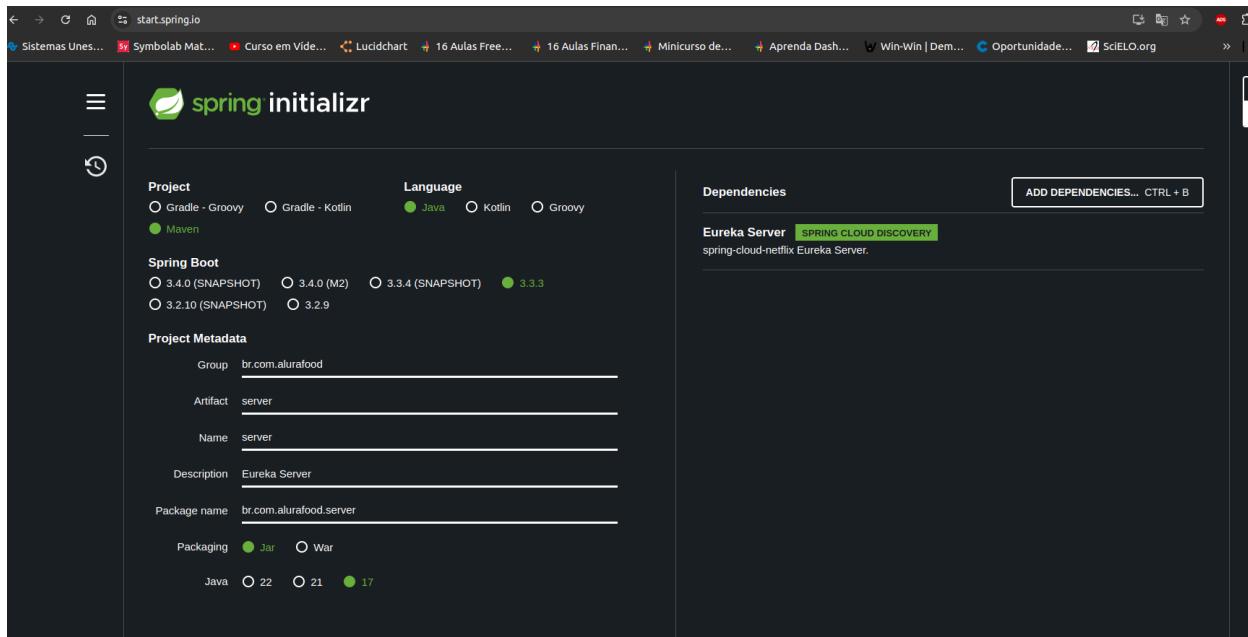
Quando chegar uma requisição, é possível distribuir corretamente para o microsserviço correspondente.

Vamos utilizar o Eureka, desenvolvido pela Netflix e hoje OpenSource:

The screenshot shows the official Spring Cloud Netflix project page. The URL in the browser is spring.io/projects/spring-cloud-netflix#overview. The page has a header with the Spring logo and navigation links for Why Spring, Learn, Projects, Training, Support, and Community. A sub-header indicates the version is 3.1.2. Below the header is a navigation bar with tabs: OVERVIEW (which is active), LEARN, SUPPORT, and SAMPLES. The main content area contains a brief description of what Spring Cloud Netflix does, mentioning its integration with Netflix OSS, autoconfiguration, and service discovery via Eureka. To the right of the text is a small portrait of a woman with blonde hair. On the left side of the page, there's a sidebar with a navigation menu. The 'Spring Cloud' section is expanded, showing sub-options like Spring Cloud Azure, Spring Cloud Alibaba, Spring Cloud for Amazon Web Services, Spring Cloud Bus, Spring Cloud Circuit Breaker, Spring Cloud CLI, Spring Cloud for Cloud Foundry, Spring Cloud - Cloud Foundry Service Broker, Spring Cloud Cluster, and Spring Cloud Commons.

<https://spring.io/projects/spring-cloud-netflix>

Vamos usar o site do Spring Initializr para associar o Eureka.

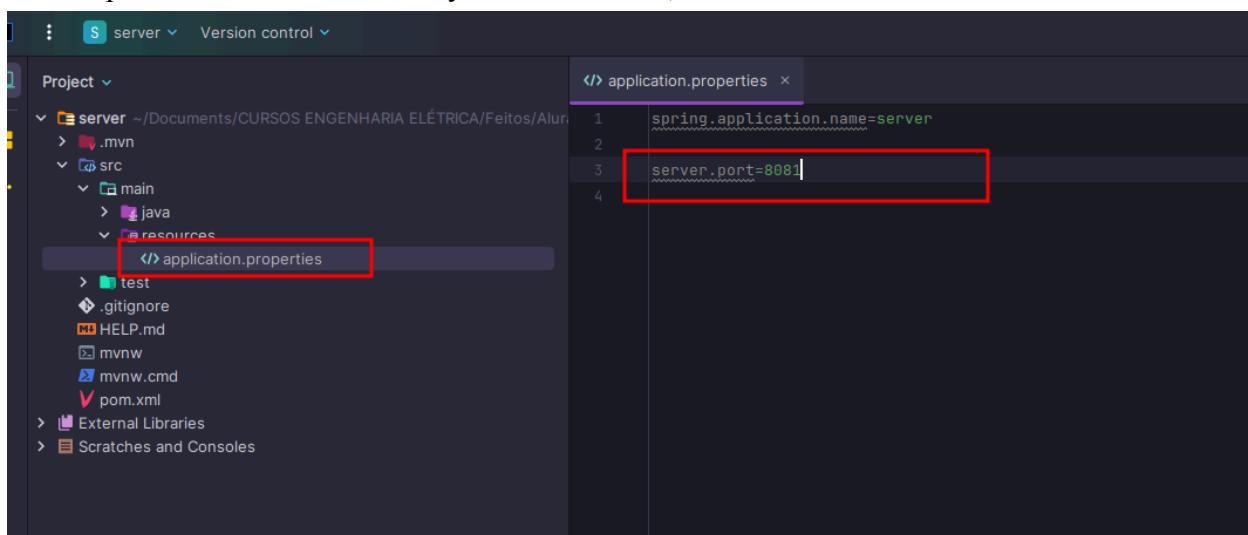


Vamos criar um novo projeto com esse Eureka.

Foi gerado um projeto, que vou abrir no IntelliJ.

A primeira coisa que vou alterar é o application.properties. Vou chumbar a porta em que ele deve subir, porque é recomendado que para o gateway e para o serviço de descoberta, é recomendável que tenhamos uma porta fixa para ele subir:

OBS: o padrão do service discovery é subir na 8761, mas vamos fazer ele subir na 8081.



E vamos incluir mais 2 propriedades.

Ele pode atuar tanto como servidor, mas tbm poderia incluir essa dependência em um projeto e atuaria como client.

Mas no nosso caso, vamos utilizá-lo somente como server, como serviço de descoberta. Então, vamos incluir duas propriedades para ele não inserir o registro dessa própria aplicação no serviço de descoberta.

```
spring.application.name=server
server.port=8081
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Essa minha aplicação vai ser de uso exclusivo como servidor.

Agora vou inserir o nome da minha aplicação, para que todas comecem a ter um nome.

```
spring.application.name=server
server.port=8081
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

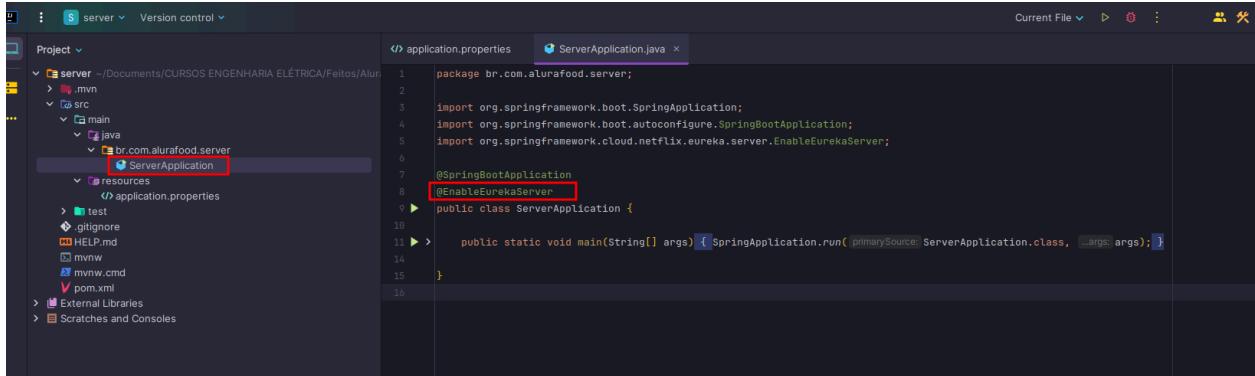
Já tinha vindo com essa propriedade!

E agora, como trocamos o endereço para a porta 8081, vamos inserir qual a url para que consigamos acessar esse serviço.

```
spring.application.name=server
server.port=8081
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
```

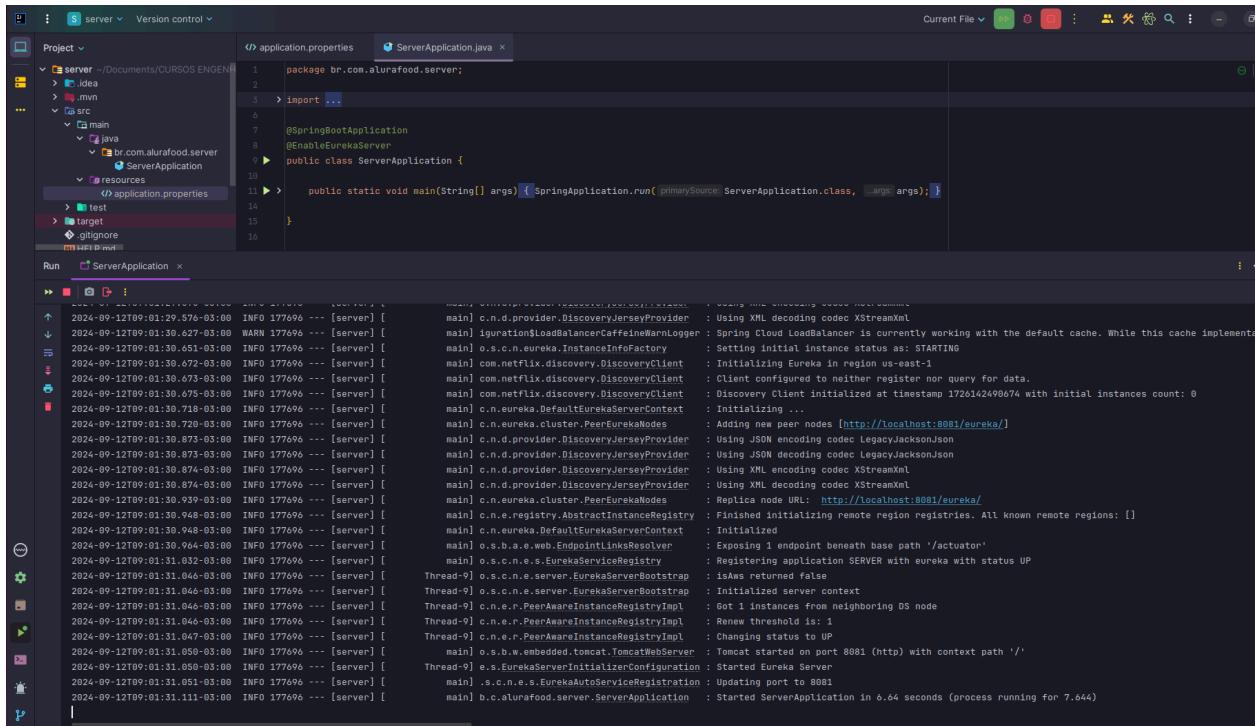
De configurações, era só isso.

Agora, precisamos alterar a classe principal do Spring, nesse caso, a `ServerApplication`, para habilitar que essa aplicação é um `EurekaServer`:



```
1 package br.com.alurafood.server;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class ServerApplication {
10
11     public static void main(String[] args) { SpringApplication.run(ServerApplication.class, args); }
12
13 }
14
15 }
```

Agora, vamos rodar a aplicação, e vamos ver no navegador nosso service discovery funcionando:



```
2024-09-12T09:01:29.576-03:00 INFO 177696 --- [server] [main] c.n.d.provider.DiscoveryJerseyProvider : Using XML decoding codec XStreamXml
2024-09-12T09:01:30.627-03:00 WARN 177696 --- [server] [main] Ignoration$LoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working with the default cache. While this cache implements
2024-09-12T09:01:30.651-03:00 INFO 177696 --- [server] [main] o.s.c.n.eureka.InstanceInfoFactory : Setting initial instance status as: STARTING
2024-09-12T09:01:30.672-03:00 INFO 177696 --- [server] [main] com.netflix.discovery.DiscoveryClient : Initializing Eureka in region us-east-1
2024-09-12T09:01:30.673-03:00 INFO 177696 --- [server] [main] com.netflix.discovery.DiscoveryClient : Client configured to neither register nor query for data.
2024-09-12T09:01:30.675-03:00 INFO 177696 --- [server] [main] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 172142498674 with initial instances count: 0
2024-09-12T09:01:30.718-03:00 INFO 177696 --- [server] [main] c.n.eureka.DefaultEurekaServerContext : Initializing ...
2024-09-12T09:01:30.720-03:00 INFO 177696 --- [server] [main] c.n.eureka.cluster.PeerEurekaNodes : Adding new peer nodes [http://localhost:8081/eureka/]
2024-09-12T09:01:30.873-03:00 INFO 177696 --- [server] [main] c.n.d.provider.DiscoveryJerseyProvider : Using JSON encoding codec LegacyJacksonJson
2024-09-12T09:01:30.873-03:00 INFO 177696 --- [server] [main] c.n.d.provider.DiscoveryJerseyProvider : Using JSON decoding codec LegacyJacksonJson
2024-09-12T09:01:30.874-03:00 INFO 177696 --- [server] [main] c.n.d.provider.DiscoveryJerseyProvider : Using XML encoding codec XStreamXml
2024-09-12T09:01:30.874-03:00 INFO 177696 --- [server] [main] c.n.d.provider.DiscoveryJerseyProvider : Using XML decoding codec XStreamXml
2024-09-12T09:01:30.939-03:00 INFO 177696 --- [server] [main] c.n.eureka.cluster.PeerEurekaNodes : Replica node URL: http://localhost:8081/eureka/
2024-09-12T09:01:30.948-03:00 INFO 177696 --- [server] [main] c.n.e.registry.AbstractInstanceRegistry : Finished initializing remote region registries. All known remote regions: []
2024-09-12T09:01:30.948-03:00 INFO 177696 --- [server] [main] c.n.eureka.DefaultEurekaServerContext : initialized
2024-09-12T09:01:30.964-03:00 INFO 177696 --- [server] [main] o.s.b.a.wm.EndpointLinksResolver : Exposing 1 endpoint beneath base path '/actuator'
2024-09-12T09:01:31.032-03:00 INFO 177696 --- [server] [main] o.s.c.e.s.EurekaServiceRegistry : Registering application SERVER with eureka with status UP
2024-09-12T09:01:31.046-03:00 INFO 177696 --- [server] [Thread-9] o.s.c.e.server.EurekaServerBootstrap : isAWS returned false
2024-09-12T09:01:31.046-03:00 INFO 177696 --- [server] [Thread-9] o.s.c.e.server.EurekaServerBootstrap : Initialized server context
2024-09-12T09:01:31.046-03:00 INFO 177696 --- [server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 instances from neighboring DS node
2024-09-12T09:01:31.046-03:00 INFO 177696 --- [server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew threshold is: 1
2024-09-12T09:01:31.047-03:00 INFO 177696 --- [server] [Thread-9] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing status to UP
2024-09-12T09:01:31.050-03:00 INFO 177696 --- [server] [main] o.s.b.a.embedded.tomcat.TomcatWebServer : Tomcat started on port 8081 (http) with context path '/'
2024-09-12T09:01:31.050-03:00 INFO 177696 --- [server] [Thread-9] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
2024-09-12T09:01:31.051-03:00 INFO 177696 --- [server] [main] l.s.c.e.s.EurekaAutoServiceRegistration : Updating port to 8081
2024-09-12T09:01:31.111-03:00 INFO 177696 --- [server] [main] b.c.alurafood.server.ServerApplication : Started ServerApplication in 6.64 seconds (process running for 7.644)
```

Subiu!

The screenshot shows a web browser window with the URL localhost:8081 in the address bar. The page title is "spring Eureka". The main content area is titled "System Status" and displays various configuration parameters:

Environment	test	Current time	2024-09-12T09:02:08 -0300
Data center	default	Uptime	00:00
		Lease expiration enabled	false
		Renews threshold	1
		Renews (last min)	0

Below this is a section titled "DS Replicas" with a sub-section for "localhost". A red box highlights the table titled "Instances currently registered with Eureka".

Application	AMIs	Availability Zones	Status
No instances available			

Further down, there are sections for "General Info" and "Instance Info", each containing a table with a single row showing "No instances available".

Não temos nenhuma instância declarada aqui por enquanto, vamos declarar.

Conhecimento: client-side e server-side discovery

Como citei nesta aula, trabalhar com uma arquitetura de microsserviços tem seus desafios, pois as instâncias dos serviços muitas vezes tem suas URLs geradas dinamicamente, podendo ser alteradas o tempo todo, inclusive por conta de atualizações ou *autoscaling*, que é o escalonamento automático, de acordo com a necessidade. Por conta disso, é necessário que o código do cliente “descubra” também de forma dinâmica quais os endereços disponíveis das instâncias dos serviços com os quais ele deseja se comunicar. A isso, damos o nome de *service discovery*.

O serviço de descoberta acontece basicamente em duas etapas. Na primeira, é necessário ter um serviço ou mecanismo onde cada instância faça o seu registro. Na segunda, é necessário que exista uma forma para que esse serviço seja localizado por outro serviço (que podemos chamar de “cliente” ou “consumidor”).

Para essa primeira etapa, que damos o nome de *service registry*, onde será gerado uma espécie de catálogo com todas as URLs dos endereços dos serviços e suas instâncias, usaremos o **Eureka Server**, que faz parte do pacote Spring Cloud Netflix.

O **Eureka Server** é um serviço REST que será responsável por “conhecer” todas as nossas instâncias de serviços. Para isso, basta criar um novo projeto pelo [site do Spring](#) e selecionar a dependência Eureka Server, conforme imagem abaixo:



Também é possível incluir a dependência diretamente no arquivo pom.xml, indicando o artefato spring-cloud-starter-netflix-eureka-server, conforme imagem abaixo:

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

Para que esse serviço conheça as nossas instâncias, utiliza-se um padrão chamado *self registration*, onde cada instância deverá se auto-registrar no servidor. É o que veremos no próximo vídeo, onde usaremos o **Eureka Client** para que o microsserviço de pagamentos seja registrado e possa ser localizado pelo nome. Esse procedimento é bem simples, como irei demonstrar, basta incluir uma dependência no arquivo pom.xml e fazer poucas configurações.

Existe também a possibilidade de aplicações de terceiros fazerem esse registro.

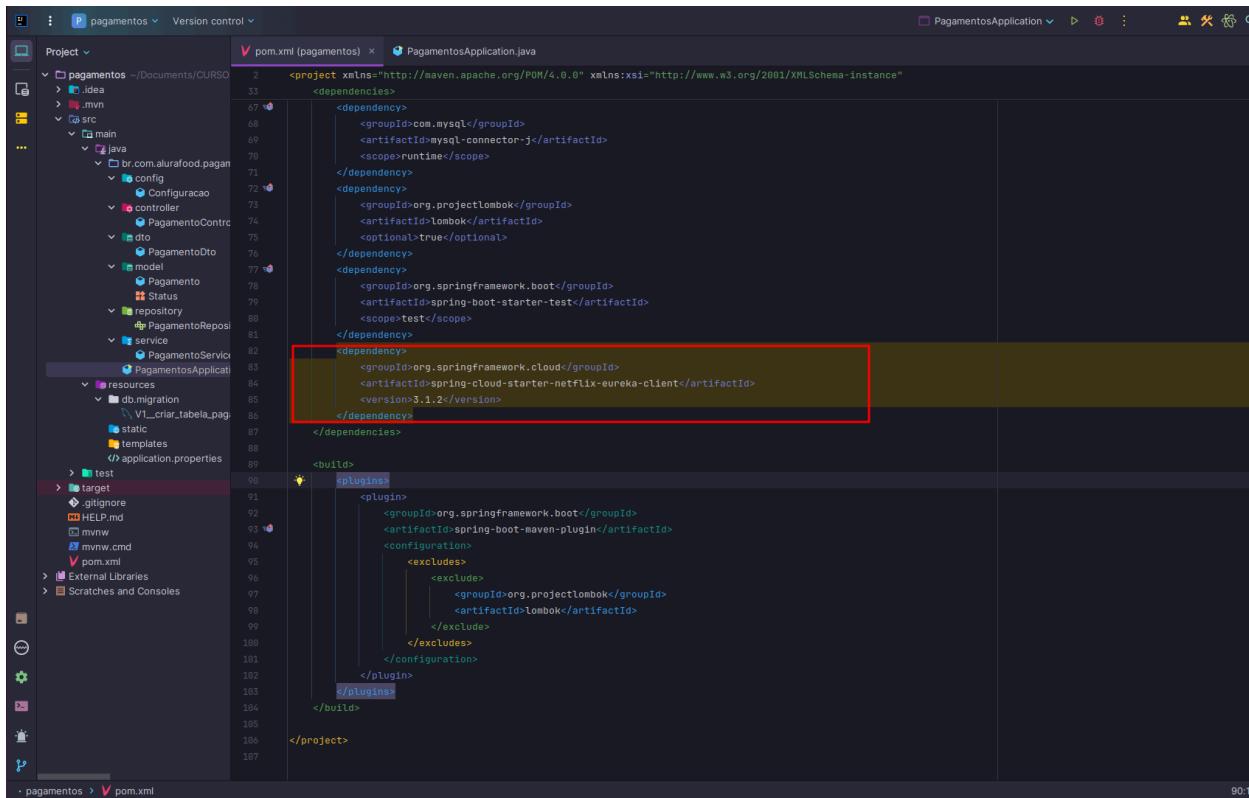
Podemos dizer que existem dois padrões principais de service discovery: client-side (descoberta do lado cliente) e server-side (descoberta do lado servidor).

No *client-side discovery*, o cliente (no caso o microsserviço ou o API Gateway) consulta o service registry, obtém a lista de instâncias do serviço que ele quer consumir e ele próprio é responsável por fazer o balanceamento de carga, escolhendo para qual instância irá direcionar a requisição (quando houver mais de uma instância do serviço que ele precisa consumir registrada). Veremos isso na aula em que abordo sobre balanceamento de carga.

No *server-side discovery* ao invés do cliente consultar diretamente o service registry, é feita uma solicitação para uma camada intermediária como um DNS ou roteador, por exemplo, que faz a consulta ao service registry e o load balancing (balanceamento de carga), já encaminhando a requisição a uma das instâncias, garantindo que nenhum servidor seja sobrecarregado e desacoplando essa lógica do cliente.

A seguir vamos configurar o nosso microsserviço de pagamento e ver todo esse fluxo de service discovery de forma prática.

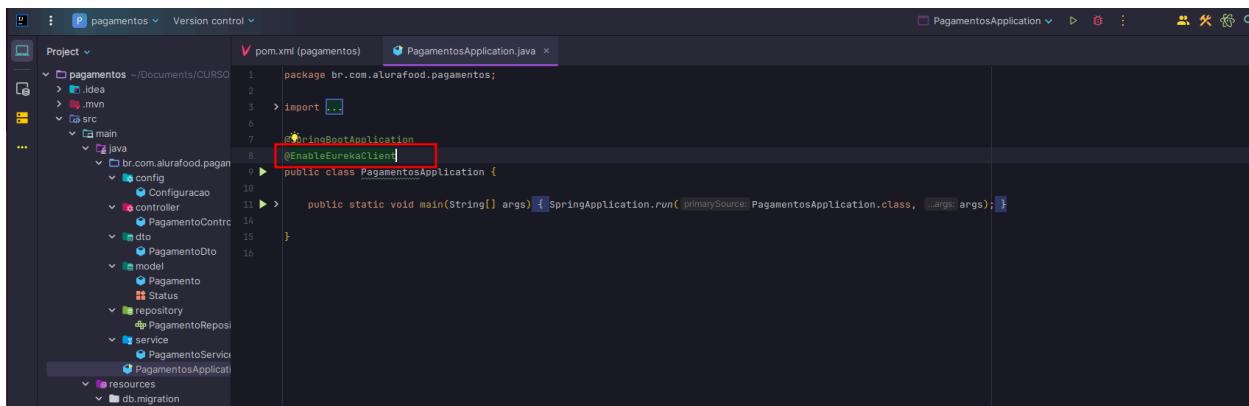
Agora, precisamos alterar o serviço de pagamento para se registrar no Eureka, no service discovery.



```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
    <version>3.1.2</version>
</dependency>
```

Instalamos essa dependência no projeto de pagamentos.

E anotamos a classe principal com `@EnableEurekaClient`:

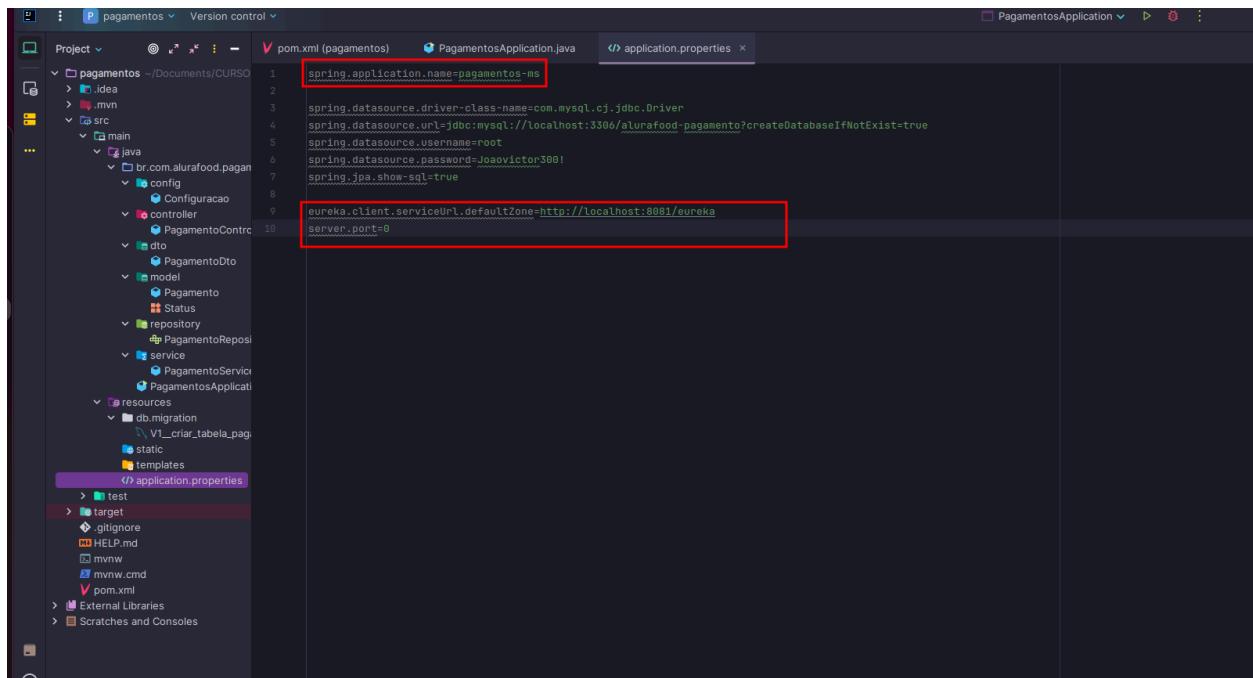


```
package br.com.alurafood.pagamentos;

import ...;

@EnableEurekaClient
public class PagamentosApplication {
```

Também tenho que inserir algumas propriedades no application.properties:



```
spring.application.name=pagamentos-ms
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/alturafood-pagamento?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=Joaovictor300!
spring.jpa.show-sql=true
eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
server.port=0
```

Estou mapeando aqui onde o meu eureka está (mesma propriedade inserida no server), o nome da aplicação e a porta estou deixando como 0 porque eu não vou me preocupar em que porta vai subir, o Eureka vai cuidar disso para mim.

Basta configurar isso e já tenho o mapeamento feito.

OBS: tive que fazer um downgrade da versão do Spring Boot para a 2.6.7 nos dois projetos, tanto de server como de client porque o Eureka fica mais estável nessa versão. E do spring cloud para 2021.0.2.

Rodando as duas apps:

Screenshot of a Spring Eureka dashboard at localhost:8081:

System Status

Environment	test	Current time	2024-09-12T17:23:03 -0300
Data center	default	Uptime	0:00
		Lease expiration enabled	false
		Renew threshold	1
		Renew (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	120mb
num-of-cpus	4
current-memory-usage	61mb (50%)
server-upptime	00:00
registered-replicas	http://localhost:8081/eureka/
unavailable-replicas	http://localhost:8081/eureka/
available-replicas	http://localhost:8081/eureka/

Instance Info

Name	Value
ipAddr	192.168.100.5
status	UP

Pagamentos Application

Version control: pom.xml (pagamentos) PagamentosApplication.java Pagamento.java PagamentoController.java PagamentoService.java application.properties

Project: pagamentos /Documents/CURSO

```

public class PagamentoService {
    public PagamentoDto obterPorId(Long id) { 1 usage
        Pagamento pagamento = repository.findById(id)
        .orElseThrow( exceptionSupplied() -> new EntityNotFoundException());
        return modelMapper.map( source: pagamento, destinationType: PagamentoDto.class);
    }

    public PagamentoDto criarPagamento(PagamentoDto dto) { 1 usage
        Pagamento pagamento = modelMapper.map( source: dto, destinationType: Pagamento.class);
        pagamento.setStatus(Status.CRIADO);
        repository.save( entity: pagamento);
    }
}

```

Run: PagamentosApplication

```

2024-09-12 17:24:07.000 INFO 263395 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : MM000490: Using JtaPlatform implementation: org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform
2024-09-12 17:24:07.011 INFO 263395 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2024-09-12 17:24:07.463 WARN 263395 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2024-09-12 17:24:07.899 INFO 263395 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-09-12 17:24:08.128 INFO 263395 --- [ restartedMain] DiscoveryClientOptionalLagers : Eureka HTTP Client uses RestTemplate.
2024-09-12 17:24:08.128 INFO 263395 --- [ restartedMain] i.gurationLoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently working with the default cache. While this cache implementation is useful for most cases, it is not thread-safe.
2024-09-12 17:24:08.191 INFO 263395 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 42953 (http) with context path ''
2024-09-12 17:24:08.193 INFO 263395 --- [ restartedMain] o.s.n.e.s.EurekaAutoServiceRegistration : Updating port to 42953
2024-09-12 17:24:08.209 INFO 263395 --- [ restartedMain] o.s.n.e.s.EurekaInstanceInfoFactory : Setting initial instance status as: STARTING
2024-09-12 17:24:08.336 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Initializing Eureka in region us-east-1
2024-09-12 17:24:08.351 INFO 263395 --- [ restartedMain] c.n.d.s.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Disable delta property : false
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Application is null : false
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
2024-09-12 17:24:08.389 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Application version is -1: true
2024-09-12 17:24:08.390 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka server
2024-09-12 17:24:08.775 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : The response status is 200
2024-09-12 17:24:08.778 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Starting heartbeat executor: renew interval is: 30
2024-09-12 17:24:08.782 INFO 263395 --- [ restartedMain] c.n.d.s.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowed rate per min is 4
2024-09-12 17:24:08.792 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 1726172648791 with initial instances count: 0
2024-09-12 17:24:08.799 INFO 263395 --- [ restartedMain] o.s.n.e.s.EurekaServiceRegistry : Registering application PAGAMENTOS with eureka with status UP
2024-09-12 17:24:08.799 INFO 263395 --- [ restartedMain] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1726172648791, current=UP, previous=STARTING]
2024-09-12 17:24:08.802 INFO 263395 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_PAGAMENTOS/192.168.100.5:pagamentos:0: registering service...
2024-09-12 17:24:08.839 INFO 263395 --- [ restartedMain] b.c.a.pagamentos.PagamentosApplication : Started PagamentosApplication in 8.999 seconds (JVM running for 9.967)
2024-09-12 17:24:08.956 INFO 263395 --- [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_PAGAMENTOS/192.168.100.5:pagamentos:0 - registration status: 204

```

The screenshot shows the Spring Eureka dashboard at localhost:8081. The top navigation bar includes links to various websites like Sistemas Unes..., Symbolab Mat..., Curso em Vídeo..., Lucidchart, 16 Aulas Free..., 16 Aulas Finan..., Minicurso de..., Aprenda Dash..., Win-Win | Dem..., Oportunidade..., SciELO.org, and All Books. The main content area has a header "spring Eureka" and a "Toggle navigation" button. Under "System Status", it shows Environment as "test" and Data center as "default". It lists Current time as 2024-09-12T17:24:35 -0300, Uptime as 00:01, Lease expiration enabled as false, Renews threshold as 3, and Renews (last min) as 0. The "DS Replicas" section shows one instance registered under "localhost", labeled "PAGAMENTOS" with AMIs "n/a (1)" and Availability Zones "(1)". The status is "UP (1) - 192.168.100.5:pagamentos:0". Below this is the "General Info" section with various system metrics like total avail memory (120mb), num of cpus (4), current memory usage (80mb, 66%), server uptime (00:01), registered replicas (http://localhost:8081/eureka/), unavailable replicas (http://localhost:8081/eureka/), and available replicas (http://localhost:8081/eureka/). The "Instance Info" section shows the IP address as 192.168.100.5.

Deu bom!

A aplicação foi registrada.

E ela está operando em alguma porta aleatória. Para descobrir a porta, basta clicar no IP gerado.

The screenshot shows a browser window with the URL 192.168.100.5:42953/actuator/info. The URL bar is highlighted with a red arrow pointing to the port number 42953. The browser's top navigation bar includes links to Sistemas Unes..., Symbolab Mat..., Curso em Vídeo..., Lucidchart, 16 Aulas Free..., 16 Aulas Finan..., and Minicurso.

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Sep 12 17:26:24 BRT 2024

There was an unexpected error (type=Not Found, status=404).
No message available

Se eu chamar no Insomnia agora, vou ter que chamar essa porta:

The screenshot shows the Insomnia REST Client interface. A GET request is made to <http://localhost:8080/pagamentos>. The response status is "Error" with a 0.37 ms duration and 0 B size. The error message is "Error: Couldn't connect to server". Below the response, there is a note: "Here are some additional things that may help." with links to "Documentation" and "Contact Support".

GET ▾ http://localhost:42953/pagamentos

Send ▾ 200 OK 255 ms 609 B

Parameters Body Auth Headers 1 Docs

Preview ▾ Headers 3 Cookies Timeline

```
1y {
2y   "content": [
3y     {
4y       "id": 2,
5y       "valor": "990.00",
6y       "nome": "Rodrigo",
7y       "numero": "12345678",
8y       "expiracao": "10/20",
9y       "codigo": "123",
10y      "status": "CRIADO",
11y      "pedidoId": 1,
12y      "formaDePagamentoId": 1
13y    },
14y    {
15y      "id": 3,
16y      "valor": "800.00",
17y      "nome": "Jv",
18y      "numero": "1254887",
19y      "expiracao": "12/25",
20y      "codigo": "148",
21y      "status": "CRIADO",
22y      "pedidoId": 1,
23y      "formaDePagamentoId": 1
24y    }
25y  ],
26y  "pageable": {
27y    "sort": {
28y      "sorted": false,
29y      "unsorted": true,
30y      "empty": true
31y    },
32y    "pageNumber": 0,
33y    "pageSize": 10,
34y    "offset": 0,
35y    "paged": true,
36y    "unpaged": false
37y  },
38y  "totalPages": 1,
39y  "totalElements": 2,
40y  "last": true,
41y  "size": 10,
42y  "number": 0,
43y  "sort": {
44y    "sorted": false,
45y    "unsorted": true,
46y    "empty": true
47y  },
48y  "numberOfElements": 2,
49y  "first": true,
50y  "empty": false
51y }
```

Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

Introduction to Insomnia 

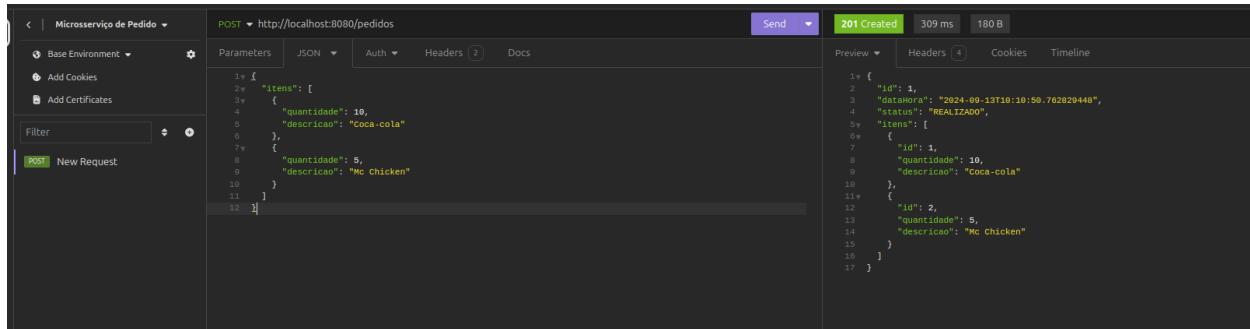
Agora os outros microsserviços que forem sendo registrados, vou mapeando ai nas portas.

Agora, vamos ter outro microsserviço, o de pedidos.

Obs: já foi disponibilizado o código.

Agora vamos testar e vamos registrar no Eureka depois.

Testando:



The screenshot shows a Postman request to `http://localhost:8080/pedidos`. The JSON body contains the following data:

```
1v [
2v   "itens": [
3v     {
4v       "quantidade": 10,
5v       "descricao": "coca-cola"
6v     },
7v     {
8v       "quantidade": 5,
9v       "descricao": "Mc Chicken"
10v    }
11v ]
12v ]
```

The response status is `201 Created` with a response time of `309 ms` and a body size of `180 B`. The response JSON is:

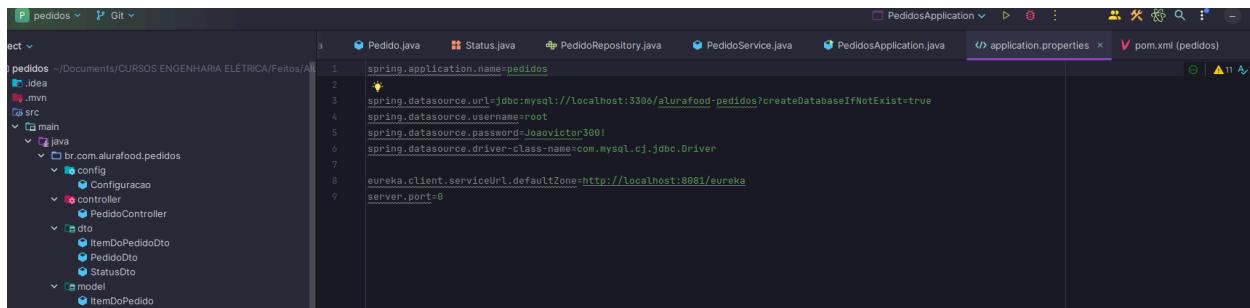
```
1v {
2v   "id": 1,
3v   "datahora": "2024-09-13T10:10:50.762Z944B",
4v   "status": "REALIZADO",
5v   "itens": [
6v     {
7v       "id": 1,
8v       "quantidade": 10,
9v       "descricao": "coca-cola"
10v    },
11v    {
12v      "id": 2,
13v      "quantidade": 5,
14v      "descricao": "Mc Chicken"
15v    }
16v  ]
17v }
```

Já cria o pedido com o Status REALIZADO.

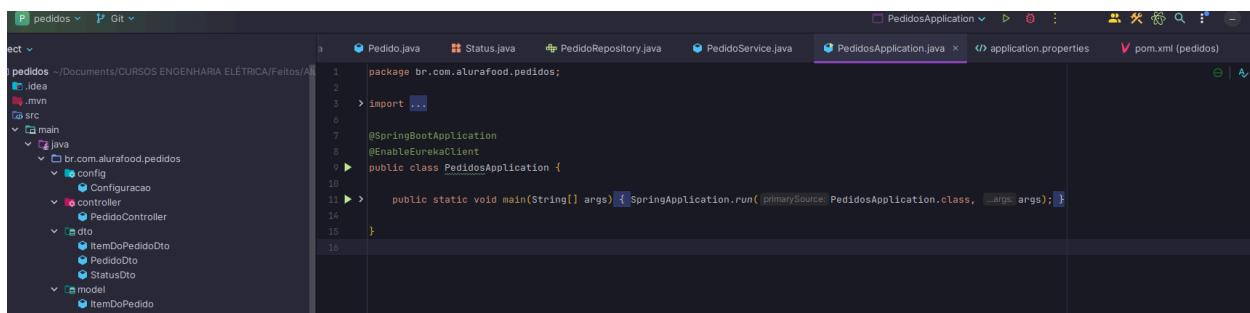
Agora, vamos registrar esse microsserviço no Eureka, para também ficar gerenciado por lá.

Vamos fazer as mesmas configs que foram feitas para o microsserviço de pagamentos.

E depois o intuito vai ser fazer com que os dois se conversem.



```
1v spring.application.name=pedidos
2v
3v spring.datasource.url=jdbc:mysql://localhost:3306/alurafood-pedidos?createDatabaseIfNotExist=true
4v spring.datasource.username=root
5v spring.datasource.password=JoaoVictor300!
6v spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
7v
8v eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
9v server.port=8080
```



```
1v package br.com.alurafood.pedidos;
2v
3v import ...
4v
5v @SpringBootApplication
6v @EnableEurekaClient
7v public class PedidosApplication {
8v
9v   public static void main(String[] args) { SpringApplication.run( PedidosApplication.class, args ); }
10v }
```

Então, já fiz as configs e agora vou subir os dois microsserviços, o de Pagamentos e o de Pedidos, para que sejam registrados no Eureka.

The screenshot shows the Spring Eureka dashboard at localhost:8081. The top navigation bar includes links to various websites like Sistemas Unes..., Symbolab Mat..., Curso em Vídeo..., Lucidchart, 16 Aulas Free..., 16 Aulas Finan..., Minicurso de..., Aprenda Dash..., Win-Win | Dem..., Oportunidade..., and SciELO.org. The main content area is titled "spring Eureka" and "Toggle navigation".

System Status

Environment	test	Current time	2024-09-13T14:22:23 -0300
Data center	default	Uptime	0:00
		Lease expiration enabled	false
		Renews threshold	5
		Renews (last min)	0

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMis	Availability Zones	Status
PAGAMENTOS	n/a (1)	(1)	UP (1) - 192.168.100.5:pagamentos
PEDIDOS	n/a (1)	(1)	UP (1) - 192.168.100.5:pedidos

General Info

Name	Value
total-avail-memory	120mb
num-of-cpus	4
current-memory-usage	81mb (67%)
server-upptime	00:00
registered-replicas	http://localhost:8081/eureka/
unavailable-replicas	
available-replicas	

Instance Info

Os dois foram registrados então!

Tudo certinho.

A cada vez que sobe a instância dos microsserviços, e eles são registrados no Eureka, sobem em uma porta diferente.

Então, não tem como saber em que porta subiram, precisaremos contar com a ajuda de um Gateway, para ter um ponto único de entrada e conseguir chamar os serviços internamente.

Agora vamos ver sobre o Gateway e Load Balancer.

Teremos um ponto único de entrada para a nossa aplicação.

Não é legal ficar controlando em que porta, em que endereço eu faço minhas solicitações... temos que ter um ponto único de entrada para o sistema para que possamos usar a aplicação.

Vamos fazer isso utilizando um API Gateway, do nosso Spring Cloud.

The screenshot shows the official Spring Cloud Gateway project page on the spring.io/projects/spring-cloud-gateway website. The page has a dark header with the Spring logo and navigation links for Why Spring, Learn, Projects, Academy, Solutions, and Community. Below the header, the main title "Spring Cloud Gateway" is displayed with a "4.1.5" badge. A navigation bar with tabs for OVERVIEW (which is active), LEARN, SUPPORT, and SAMPLES follows. The "OVERVIEW" section contains a brief description of the project: "This project provides a library for building an API Gateway on top of Spring WebFlux or Spring WebMVC. Spring Cloud Gateway aims to provide a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as: security, monitoring/metrics, and resiliency." Below this is a "Features" section listing various capabilities. At the bottom of the page is a "Getting Started" section with a code snippet for a Spring Boot application and a "COPY" button.

Então, no Spring initializr:

Project

- Maven Project
- Gradle Project

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.0.0 (SNAPSHOT)
- 3.0.0 (M2)
- 2.7.0 (SNAPSHOT)
- 2.7.0 (RC1)
- 2.6.8 (SNAPSHOT)
- 2.6.7
- 2.5.14 (SNAPSHOT)
- 2.5.13

Project Metadata

Group: br.com.alurafood

Artifact: gateway

Name: gateway

Description: Gateway

Dependencies

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.

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.

Adicionou o Gateway, do spring cloud e o eureka client, porque também vamos registrar o nosso Gateway no nosso Service Registry, que é o Eureka.

Agora, alterando o application.properties do gateway:

```

spring.application.name=gateway
server.port=8082
eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
spring.cloud.gateway.discovery.locator.enabled=true
spring.cloud.gateway.discovery.locator.lowerCaseServiceId=true

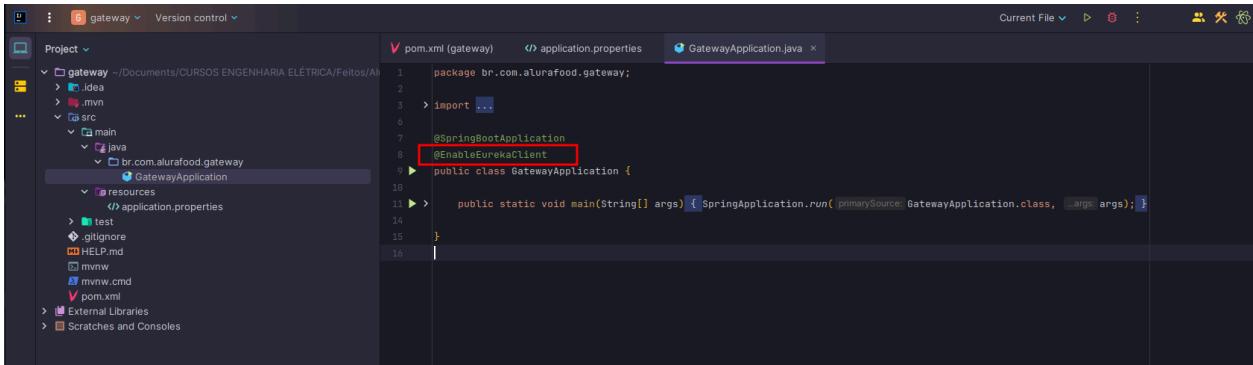
```

Inseri a porta em que ele vai rodar... deixei chumbada, assim como fiz no de server. Deixei chumbada aqui na 8082.

Inseri as informações do eureka também.

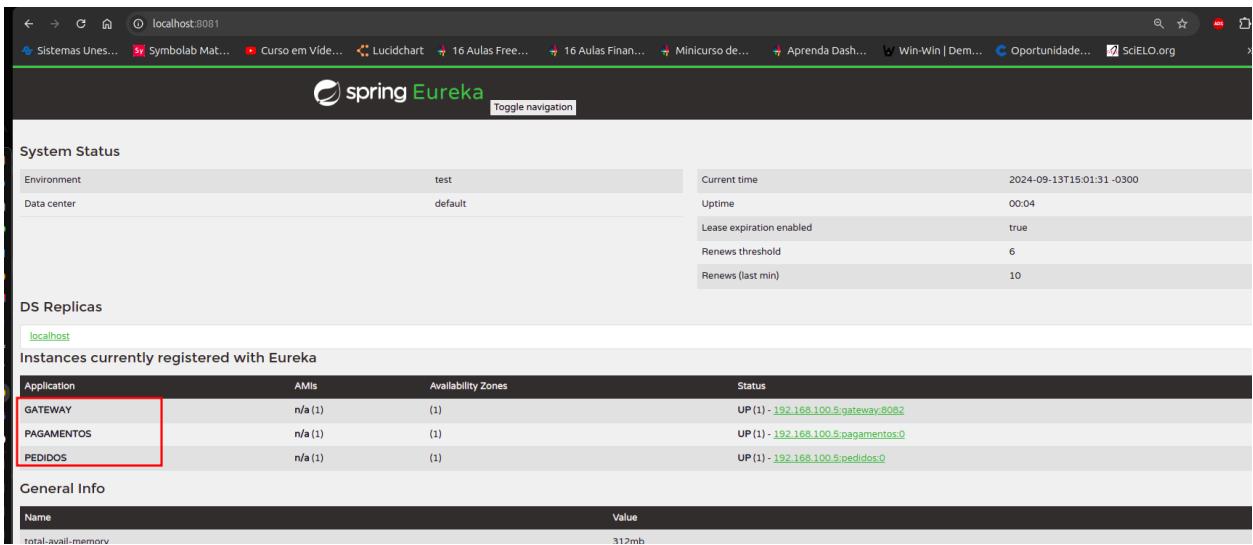
E também o application.name e duas propriedades do gateway, uma que é para deixar o discovery.locator.enabled=true, que é a mágica, onde eu vou conseguir achar os serviços registrados no registry, e o lowerCaseServiceId=true, ou seja, deixar os nomes dos serviços em letra minúscula.

E no application para registrar no service registry:



```
1 package br.com.alurafood.gateway;
2
3 import ...
4
5 @SpringBootApplication
6 @EnableEurekaClient
7
8 public class GatewayApplication {
9
10    public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }
11
12    }
13
14
15
16 }
```

Agora, já podemos inicializar a aplicação do gateway e vamos ver ele sendo registrado juntamente com os outros serviços no Eureka.



Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - 192.168.100.5/gateway:8082
PAGAMENTOS	n/a (1)	(1)	UP (1) - 192.168.100.5/pagamentos:0
PEDIDOS	n/a (1)	(1)	UP (1) - 192.168.100.5/pedidos:0

Pronto, todos registrados no Eureka.

Vou trocar de pagamentos para pagamentos-ms e de pedidos para pedidos-ms no application.properties a propriedade name deles:

The screenshot shows the Spring Eureka dashboard at localhost:8081. The top navigation bar includes links to various websites like Sistemas Unies..., Symbolab Mat..., Curso em Vide..., Lucidchart, 16 Aulas Free..., 16 Aulas Finan..., Minicurso de..., Aprenda Dash..., Win-Win | Dem..., Oportunidade..., scielo.org, and All Bookmarks.

System Status

Environment	test	Current time	2024-09-13T15:03:34 -0300
Data center	default	Uptime	00:06
		Lease expiration enabled	true
		Renews threshold	6
		Renews (last min)	11

DS Replicas

Instances currently registered with Eureka			
Application	AMs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - 192.168.100.5:gateway:8082
PAGAMENTOS-MS	n/a (1)	(1)	UP (1) - 192.168.100.5:pagamentos-ms:0
PEDIDOS-MS	n/a (1)	(1)	UP (1) - 192.168.100.5:pedidos-ms:0

General Info

Name	Value
total-avail-memory	312mb
num-of-cpus	4
current-memory-usage	63mb (20%)
server-upptime	00:06
registered-replicas	http://localhost:8081/eureka/
unavailable-replicas	http://localhost:8081/eureka/
available-replicas	http://localhost:8081/eureka/

Agora, para fazer uma requisição no postman, ou no insomnia, eu sempre colocava:
[http://localhost:\\${porta}/pedidos](http://localhost:${porta}/pedidos) ou [http://localhost:\\${porta}/pagamentos](http://localhost:${porta}/pagamentos).

Agora, vamos usar o gateway, não vamos mais nos preocupar em chamar a porta.
Para tanto, fica assim:

The screenshot shows the Insomnia REST Client interface. The URL field contains `GET http://localhost:8082/pagamentos-ms/pagamentos`. The response status is **200 OK**, with a response time of **1.31 s** and a size of **609 B**. The response body is a JSON array of payment records:

```

[{"content": [
    {"id": 2,
     "valor": 999.00,
     "nome": "Jv",
     "numero": "12345678",
     "expiracao": "10/29",
     "codigo": "123",
     "status": "CRIADO",
     "pedidoId": 1,
     "formaDePagamentoId": 1
    },
    {"id": 3,
     "valor": 800.00,
     "nome": "Jv",
     "numero": "12345678",
     "expiracao": "10/29",
     "codigo": "123",
     "status": "CRIADO",
     "pedidoId": 1,
     "formaDePagamentoId": 1
    }
  ],
  "pageable": {
    "sort": {
      "sorted": false,
      "unsorted": true,
      "empty": true
    },
    "pageNumber": 0,
    "pageSize": 10,
    "totalPages": 1,
    "paged": true,
    "unpaged": false
  },
  "numberOfElements": 2,
  "last": true,
  "size": 10,
  "number": 0,
  "first": true,
  "sorted": false,
  "unsorted": true,
  "empty": true
},
{"numberOfElements": 2,
  "first": true,
  "empty": false
}]

```

<http://localhost:8082/nome-registrado-do-microservice/path-do-microservice>.

Então, chamo a porta do gateway, a 8082, chamo o nome registrado no service registry (pagamentos-ms ou pedidos-ms) e depois chamo o próprio pagamentos ou pedidos.
Então, não preciso mais me preocupar com a porta em que os microsserviços estão.

The screenshot shows the Insomnia REST client interface. At the top, the menu bar includes Application, File, Edit, View, Window, Tools, Help, and a search bar. The main area has tabs for Requisições, Parameters, Body, Auth, Headers, and Docs. A URL preview shows the endpoint `http://localhost:8082/pedidos-ms/pedidos`. The response status is 200 Ok, with a response time of 416 ms and a body size of 172 B. The response content is displayed in a code editor-like pane:

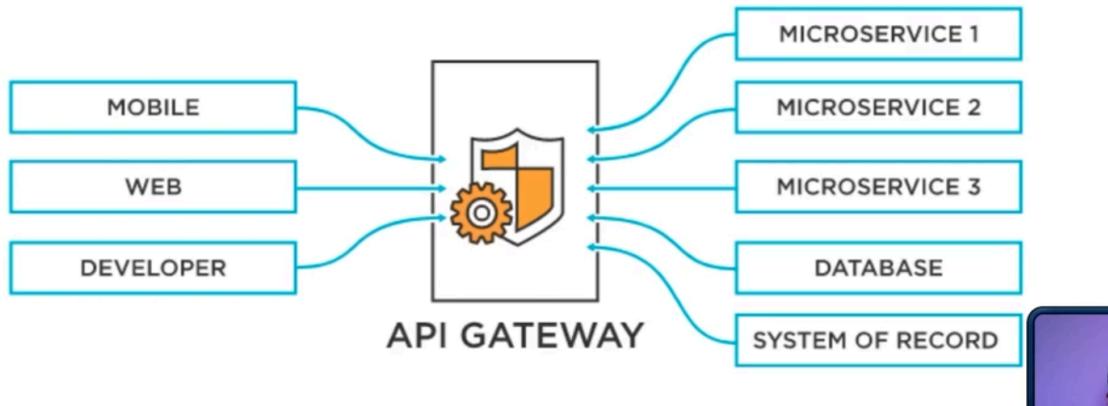
```
1: [
2:   {
3:     "id": 1,
4:     "datahora": "2024-09-13T18:18:51",
5:     "status": "REALIZADO",
6:     "itens": [
7:       {
8:         "id": 1,
9:         "quantidade": 10,
10:        "descricao": "Coca-cola"
11:      },
12:      {
13:        "id": 2,
14:        "quantidade": 5,
15:        "descricao": "Mc Chicken"
16:      }
17:    ]
18:  }
19: ]
```

Mesma coisa para o de pedidos.

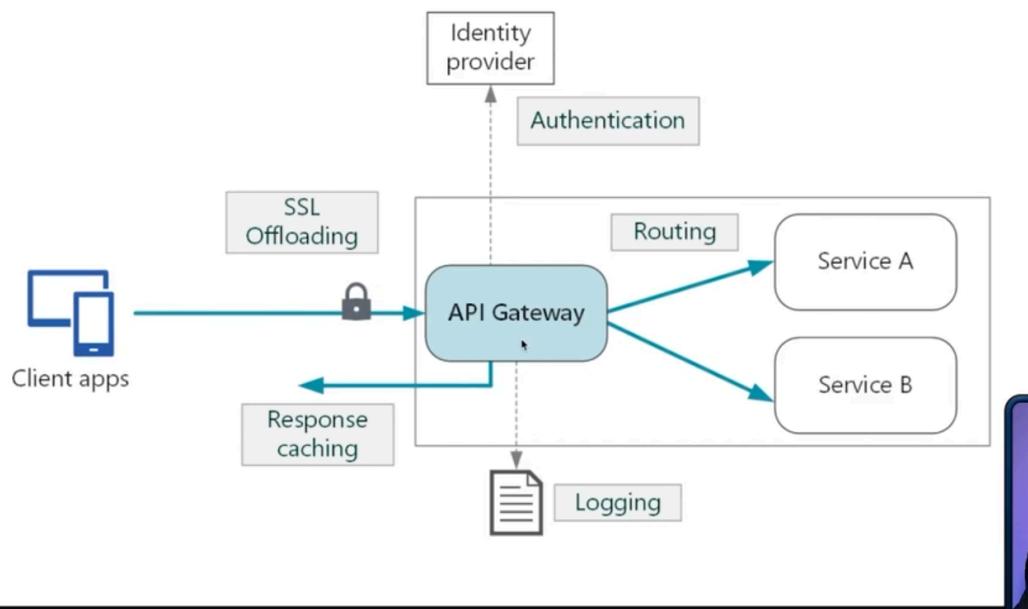
Conhecimento: API Gateway

<https://cursos.alura.com.br/extra/alura-mais/o-que-e-um-api-gateway--c1138>

API Gateway



Comportamentos do Gateway



Aqui, normalmente são API Gateways de serviços (Azure, AWS, etc.), podem fazer várias coisas... logging, authentication, routing, etc.

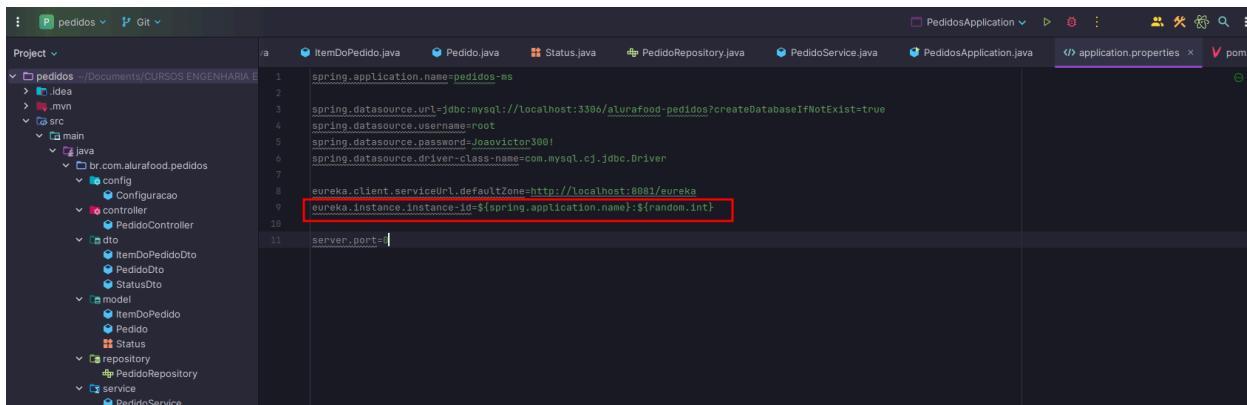
Agora, vamos falar sobre Balanceamento de carga.

Uma das vantagens de se usar uma abordagem de microsserviços, é a facilidade de fazer uma escalabilidade horizontal somente de uma determinada parte da aplicação. Ex: na black friday, vou escalar horizontalmente meu serviço de pedidos, porque vou ter bem mais. Então, vou criar várias instâncias daquele serviço.

O Gateway do Spring faz esse balanceamento de carga, onde podemos subir várias instâncias de um serviço, e no momento da requisição, ele distribui as chamadas para as instâncias disponíveis.

Vamos fazer isso para o microsserviço de pedidos.

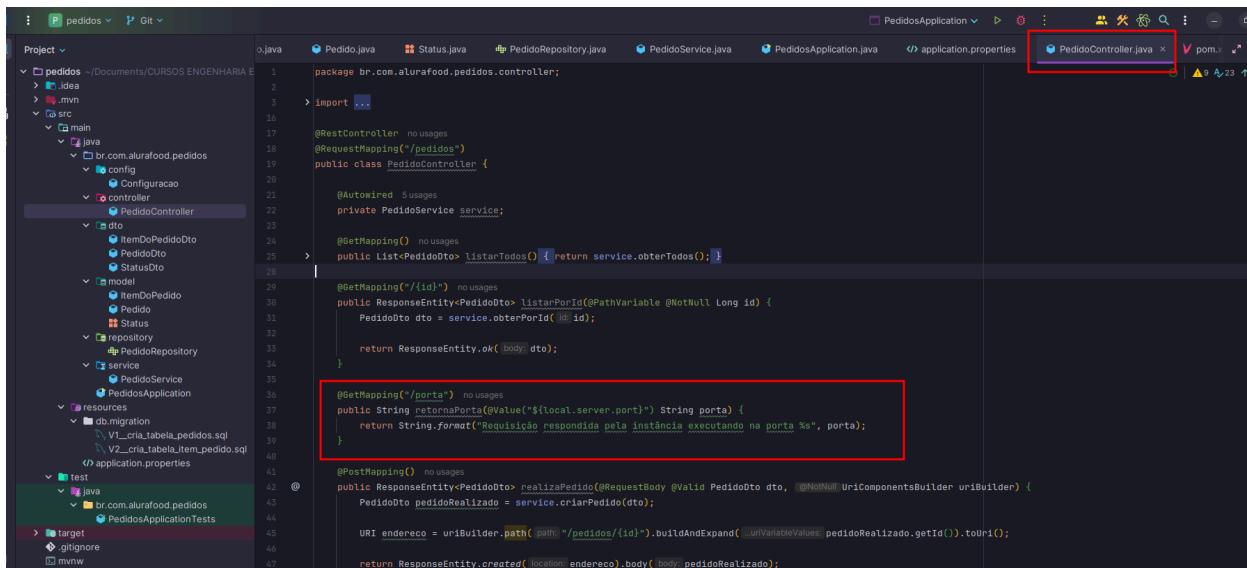
Teremos que identificar cada instância do microsserviço a partir de um id:



```
spring.application.name=pedidos-ms
spring.datasource.url=jdbc:mysql://localhost:3306/aluraFood-pedidos?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=JoaoVictor3001
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
eureka.instance.id=${spring.application.name}:${random.int}
server.port=
```

O instance-id vai ser composto pelo nome da aplicação : um número aleatório inteiro.

Para conferir em que porta ele chamou, vou mandar imprimir no controlador a porta que foi chamada a partir de um método:



```
package br.com.alurafood.pedidos.controller;
import ...
@RestController
@RequestMapping("/pedidos")
public class PedidoController {
    @Autowired
    private PedidoService service;
    @GetMapping()
    public List<PedidoDto> listarTodos() { return service.obterTodos(); }
    @GetMapping("/{id}")
    public ResponseEntity<PedidoDto> listarPorId(@PathVariable @NotNull Long id) {
        PedidoDto dto = service.obterPorId(id);
        return ResponseEntity.ok(dto);
    }
    @GetMapping("/porta")
    public String retornaPorta(@Value("${local.server.port}") String porta) {
        return String.format("Requisição respondida pela instância executando na porta %s", porta);
    }
    @PostMapping()
    public ResponseEntity<PedidoRealizado> realizaPedido(@RequestBody @Valid PedidoDto dto, @NotNull UriComponentsBuilder uriBuilder) {
        PedidoRealizado pedidoRealizado = service.criarPedido(dto);
        URI endereco = uriBuilder.path("{path}/pedidos/{id}").buildAndExpand(new UriVariableValues(pedidoRealizado.getId())).toUri();
        return ResponseEntity.created(location: endereco).body(pedidoRealizado);
    }
}
```

Aqui, fiz um GetMapping, no endereço /porta, onde eu vou atribuir na variável porta o valor do server.port (application.properties) e imprimir essa porta.

Subindo novamente o pedidos:

The screenshot shows the Spring Eureka dashboard at localhost:8081. The 'System Status' section displays environment details like 'Environment: test', 'Data center: default', and system metrics like 'Uptime: 00:47'. The 'Instances currently registered with Eureka' section lists three instances: 'GATEWAY' (UP(1) - 192.168.100.5:gateway:8082), 'PAGAMENTOS-MS' (UP(1) - 192.168.100.5:pagamentos-ms:0), and 'PEDIDOS-MS' (UP(1) - 192.168.100.5:pedidos-ms:1392316203). The 'General Info' section provides detailed server statistics.

Já subiu diferente!

Agora, precisamos subir outra instância do pedidos-ms.

Vamos fazer isso via terminal:

The screenshot shows the IntelliJ IDEA interface with the 'PedidosApplication' project open. The 'target' folder in the project tree contains the generated JAR file 'pedidos-0.0.1-SNAPSHOT.jar'. The terminal window shows the execution of 'mvn clean package' followed by the command 'java -jar target/pedidos-0.0.1-SNAPSHOT.jar'.

Eu dou um mvn clean package, para buildar o projeto e depois executo o jar gerado dentro da pasta target.

Abri 2 terminais e rodei e mais o do IntelliJ.

Dessa forma, tenho 3 instâncias do ms de pedidos:

The screenshot shows the Spring Eureka dashboard at localhost:8081. The 'System Status' section displays environment information like 'Environment: test' and 'Data center: default'. The 'DS Replicas' section lists registered instances under 'localhost':

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - 192.168.100.5/gateway:8082
PAGAMENTOS-MS	n/a (1)	(1)	UP (1) - 192.168.100.5/pagamentos-ms:0
PEDIDOS-MS	n/a (3)	(3)	UP (3) - pedidos-ms:162047417, pedidos-ms:3993202885, pedidos-ms:1615622238

The row for 'PEDIDOS-MS' is highlighted with a red box and three red arrows pointing to the three listed ports (162047417, 3993202885, and 1615622238).

Temos 3 instâncias, rodando em 3 portas.

Agora, vamos testar:

The screenshot shows an Insomnia API client interface. A GET request is made to <http://localhost:8082/pedidos-ms/pedidos/porta>. The response status is 200 OK, with a response time of 130 ms and 65 B. The response body contains the message: "Requisição respondida pela instância executando na porta 35357".

A primeira requisição foi na porta 35357.

The screenshot shows another Insomnia API client interface. A GET request is made to <http://localhost:8082/pedidos-ms/pedidos/porta>. The response status is 200 OK, with a response time of 161 ms and 65 B. The response body contains the message: "Requisição respondida pela instância executando na porta 43621".

A segunda foi na porta 43621.

The screenshot shows the Insomnia REST client interface. In the top right corner, there's a purple button labeled 'Invite' and a user icon with the name 'joao.vieira'. The main area displays a successful API call with the following details:

- Method: GET
- URL: `http://localhost:8082/pedidos-ms/pedidos/porta`
- Status: 200 OK
- Time: 180 ms
- Size: 65 B

Below the status bar, a message indicates the response was sent from an instance running on port 37483. A red arrow points to this message.

Depois foi na porta 37403.

Enfim, ele vai mudando as portas, fazendo o balanceamento de carga.

E tudo isso sozinho, o próprio gateway faz isso! Não precisamos fazer nada.

O Spring realmente é fantástico.

04 Load balancing

PRÓXIMA ATIVIDADE

Carla assistiu às aulas do curso e entendeu que *load balancing* é o processo de distribuir as requisições vindas do cliente para as várias instâncias disponíveis de um serviço. Na arquitetura que estamos usando, qual dos projetos está fazendo esse papel de *balancear a carga*?

A O projeto de microsserviço de Pedidos, que está com o Eureka Client configurado.

B O Eureka Server.

C O Gateway.

Alternativa correta! Sim, o Spring Cloud Gateway, que usamos no curso, obtém a lista de endereços de todos os serviços registrados no Eureka Server, configura uma rota dinâmica para esses serviços e já faz o balanceamento de carga nas requisições.

DISCUTIR NO FÓRUM PRÓXIMA ATIVIDADE

Agora vamos tratar sobre a comunicação entre os microsserviços.

Temos a síncrona e a assíncrona.

No próximo curso trataremos sobre comunicação assíncrona (que é feita através de mensageria).

Vamos tratar aqui a comunicação síncrona, usando o OpenFeign, que é uma solução do spring, um cliente HTTP, para fazermos integrações de back-end para back-end, então conseguimos através de anotações configurar essa comunicação.

<https://spring.io/projects/spring-cloud-openfeign>

The screenshot shows the official Spring Cloud OpenFeign project page. The URL in the browser is spring.io/projects/spring-cloud-openfeign. The page features a dark header with the Spring logo and navigation links for Why Spring, Learn, Projects, Academy, Solutions, and Community. Below the header, there's a sidebar with links to other Spring projects like Spring Boot, Spring Framework, Spring Data, and Spring Cloud. The main content area has a title "Spring Cloud OpenFeign" with a green "4.1.2" badge. It includes tabs for OVERVIEW (which is selected), LEARN, SUPPORT, and SAMPLES. The "OVERVIEW" section contains a warning about the project being feature-complete and migrating to Spring Interface Clients. The "Features" section lists "Declarative REST Client: Feign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC".

Situação que temos:

- Temos 1 ms de pedidos e 1 de pagamentos.
- Quando o pagamento for confirmado, tenho que mandar uma mensagem para o ms de pedidos, indicando que aquele pedido está pago.
- Para fazer isso, no de pedidos, temos que ter um método no controller para receber essa requisição.

No postman:

O que o MS de pagamento terá que fazer é:

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:8082/pedidos-ms/pedidos/1/pago` is entered in the address bar. Below it, a **PUT** method is selected from a dropdown. The main content area displays the request details:

- Params:** Authorization, Headers (7), Body, Pre-request Script, Tests, Settings.
- Query Params:** (highlighted in red)
- Body:** (highlighted in red)

The **Body** section contains a table with one row:

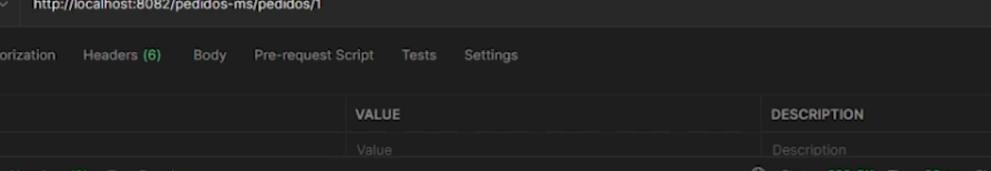
KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

At the bottom, the response is shown as a table:

Body	Cookies	Headers (2)	Test Results	Status: 200 OK	Time: 107 ms	Size: 75 B	Save Response
Pretty	Raw	Preview	Visualize	Text			

The response status is **200 OK**.

Então, fazendo um Get novamente:



The screenshot shows a Postman request for `http://localhost:8082/pedidos-ms/pedidos/1`. The response body is a JSON object with the following structure:

```
1 {  
2   "id": 1,  
3   "dataHora": "2022-05-09T21:20:48",  
4   "status": "PAGO",  
5   "itens": [  
6     {  
7       "id": 1,  
8       "quantidade": 10,  
9       "descricao": "Coca-cola"  
10    },  
11  ]  
12}
```

A red arrow points to the word "PAGO" in the "status" field of the JSON response.

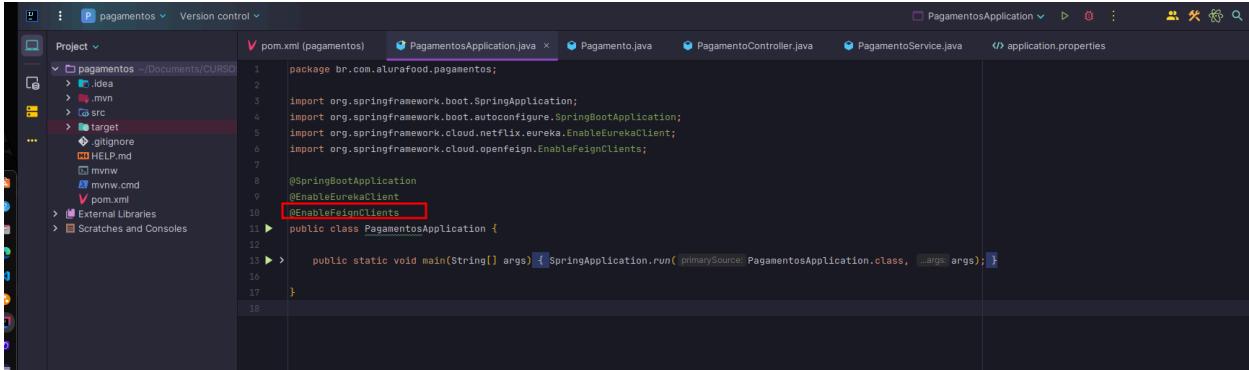
Então, o ms de pagamentos vai fazer uma requisição para o de pedidos, passando o id do pedido, passando que ele está pago, e o de pedidos vai trocar o status falando que já está pago.

Conhecimento:

Spring cloud: <https://spring.io/projects/spring-cloud>

Circuit breaker: <https://resilience4j.readme.io/docs/getting-started-3>

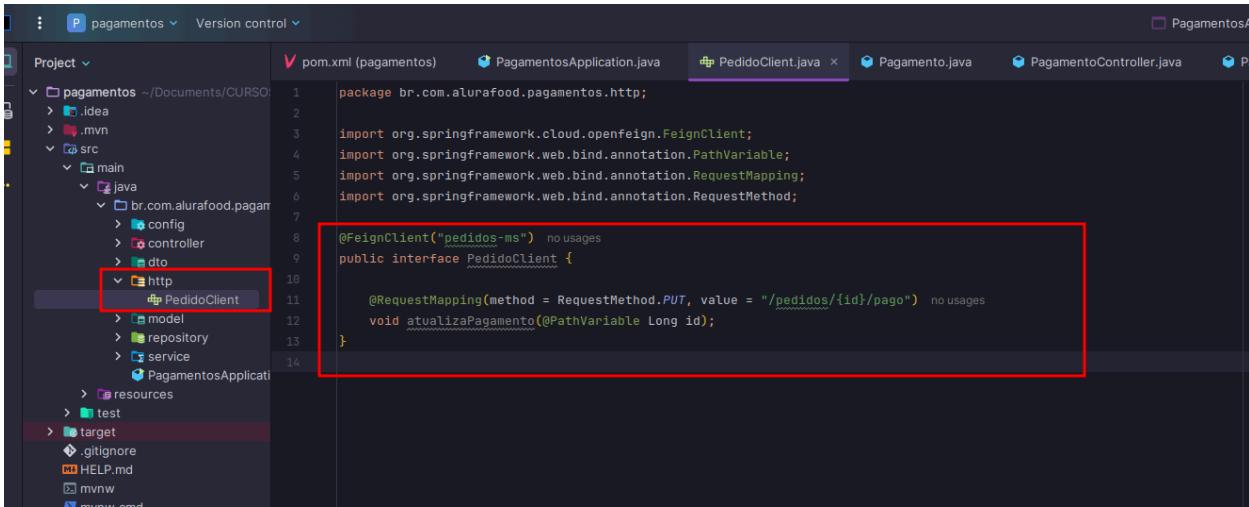
No microsserviço de pagamentos, inseri uma anotação do OpenFeign:



```
package br.com.alurafood.pagamentos;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;

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

Para configurar isso, vamos criar um novo pacote: http (client). Vai ser uma chamada a outro microsserviço.



```
package br.com.alurafood.pagamentos.http;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@FeignClient("pedidos-ms")
public interface PedidoClient {
    @RequestMapping(method = RequestMethod.PUT, value = "/pedidos/{id}/pago")
    void atualizaPagamento(@PathVariable Long id);
}
```

Foi criada uma interface, PedidoClient, e usei a anotação FeignClient, passando o nome do microsserviço como sendo o mesmo que especifiquei no application.properties do microsserviço de pedidos.

E vou mapear o método atualizaPagamento, como sendo um RequestMapping, do tipo put e o value da url que vou chamar é o mesmo que chamei no Postman: /pedidos/{id}/pago.

E recebo um Path Variable no método, que é o id.

Vou usar essa interface no meu service.

```

public void confirmarPagamento(Long id) { no usages
    Optional<Pagamento> pagamento = repository.findById(id);

    if (pagamento.isEmpty()) {
        throw new EntityNotFoundException();
    }

    pagamento.get().setStatus(Status.CONFIRMADO);
    repository.save(entity: pagamento.get());
    pedido.atualizaPagamento( entity: pagamento.get().getPedidoId());
}

```

Criei esse método, com um Optional, porque não sei se teremos o pagamento com aquele id.

```

import br.com.alurafood.pagamentos.HttpPedidoClient;
import br.com.alurafood.pagamentos.model.Pagamento;
import br.com.alurafood.pagamentos.model.Status;
import br.com.alurafood.pagamentos.repository.PagamentoRepository;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import javax.persistence.EntityNotFoundException;
import java.util.Optional;

```

```

@Service 2 usages
public class PagamentoService {

    @Autowired 7 usages
    private PagamentoRepository repository;

    @Autowired 6 usages
    private ModelMapper modelMapper;

    @Autowired 1 usage
    private PedidoClient pedido;

    public Page<PagamentoDto> obterTodos(Pageable paginacao) { 1 usage
        return repository
            .findAll(paginacao)
            .map(converter: p -> modelMapper.map(source: p, destinationType: PagamentoDto.class));
    }

    public PagamentoDto obterPorId(Long id) { 1 usage
        Pagamento pagamento = repository.findById(id);
        .orElseThrow(exceptionSupplier: () -> new EntityNotFoundException());

        return modelMapper.map(source: pagamento, destinationType: PagamentoDto.class);
    }

    public PagamentoDto criarPagamento(PagamentoDto dto) { 1 usage
        Pagamento pagamento = modelMapper.map(source: dto, destinationType: Pagamento.class);
}

```

Injetei o PedidoClient aqui para usar no método.

Agora, na Controller, vou ter que ter um método para chamar esse método de confirmarPagamento.

```

19     public class PagamentoController {
20
21         ...
22
23         @GetMapping("/{id}")
24         public ResponseEntity<PagamentoDto> detalhar(@PathVariable @NotNull Long id) {
25             PagamentoDto dto = service.obterPorId(id);
26
27             return ResponseEntity.ok().body(dto);
28         }
29
30         @PostMapping
31         public ResponseEntity<PagamentoDto> cadastrar(@RequestBody @Valid PagamentoDto pagamentoDto, @NotNull UriComponentsBuilder uriBuilder) {
32             PagamentoDto dto = service.criarPagamento(pagamentoDto);
33
34             URI endereco = uriBuilder.path("/pagamentos/{id}").buildAndExpand(dto.getId()).toUri();
35
36             return ResponseEntity.created(location).body(dto);
37         }
38
39         @PutMapping("/{id}")
40         public ResponseEntity<PagamentoDto> atualizar(@PathVariable @NotNull Long id, @RequestBody @Valid PagamentoDto pagamentoDto) {
41             PagamentoDto dto = service.atualizarPagamento(id, pagamentoDto);
42
43             return ResponseEntity.ok().body(dto);
44         }
45
46         @DeleteMapping("/{id}")
47         public ResponseEntity<Void> remover(@PathVariable @NotNull Long id) {
48             service.excluirPagamento(id);
49
50             return ResponseEntity.noContent().build();
51         }
52
53         @PatchMapping("/{id}/confirmar")
54         public void confirmarPagamento(@PathVariable @NotNull Long id) {
55             service.confirmarPagamento(id);
56         }
57     }
58
59
60
61
62
63
64

```

Vai ser um Patch, porque é uma modificação parcial do recurso... um PUT iria alterar ele inteiro.

Vamos subir o serviço, e testar.

Subiu e vamos testar no Postman.

Preview	Headers	Cookies	Timeline
<pre> 1 [2 { 3 "id": 1, 4 "dataHora": "2024-09-13T18:10:51", 5 "status": "REALIZADO", 6 "item": [7 { 8 "id": 1, 9 "quantidade": 10, 10 "descricao": "Coca-Cola" 11 }, 12 { 13 "id": 2, 14 "quantidade": 5, 15 "descricao": "Mc Chicken" 16 } 17] 18] 19 </pre>			

Temos o Pedido 1, que está como realizado.

Agora vamos testar:

```

1y {
2y   "content": [
3y     {
4y       "id": 2,
5y       "value": 999.99,
6y       "name": "Rodrigo",
7y       "numero": "12345678",
8y       "expiração": "2024/09",
9y       "codigos": "123",
10y      "status": "CRIADO",
11y      "pedidoId": 1,
12y      "formadoPagamento": 1
13y    }
14y  ],
15y  "pageable": {
16y    "sort": {
17y      "sorted": false,
18y      "unsorted": true,
19y      "empty": true
20y    },
21y    "pageNumber": 0,
22y    "pageSize": 10,
23y    "offset": 0,
24y    "paged": true,
25y    "unpaged": false
26y  },
27y  "totalPages": 1,
28y  "totalElements": 1,
29y  "last": true,
30y  "size": 10,
31y  "number": 0,
32y  "sort": {
33y    "sorted": false,
34y    "unsorted": true,
35y    "empty": true
36y  },
37y  "numberOfElements": 1,
38y  "first": true,
39y  "empty": false
40y }

```

E temos o Pagamento 2, que tem o pedido 1 como seu pedido.

Fazendo o PATCH:

Agora, vamos consultar denovo, para ver o pagamento ficou como Confirmado e se o Pedido mudou de Status tbm.

```

{
  "id": 2,
  "valor": 999.99,
  "numero": "987654321",
  "expiracao": "10/20",
  "codigo": "123",
  "status": "PAGAMENTO",
  "formaPagamentoId": 1,
  "formaDePagamento": {
    "pagavel": {
      "sort": 1,
      "sorted": false,
      "unsorted": true,
      "empty": true
    }
  },
  "pageNumber": 0,
  "pageSize": 10,
  "offset": 0,
  "paged": true,
  "unpaged": false
},
{
  "totalPages": 1,
  "totalElements": 1,
  "last": true,
  "size": 10,
  "empty": 0,
  "sort": {
    "sorted": false,
    "unsorted": true,
    "empty": true
  },
  "numberOfElements": 1,
  "first": true,
  "empty": false
}

```

Deu bom!

Vamos ver o pedido agora:

```

{
  "id": 1,
  "datahora": "2024-09-13T10:10:51",
  "status": "PAGO"
}

```

Deu certinho!

Essa foi a comunicação síncrona.

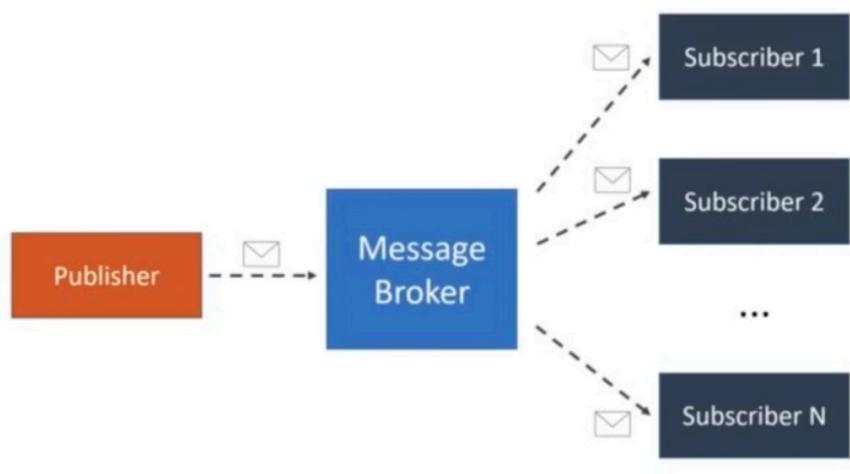
Eu chamei um microsserviço dentro de outro, passando o verbo que eu vou chamar, etc. E um influencia no outro.

Deu tudo certo!

Conhecimento: comunicação assíncrona - mensageria

<https://cursos.alura.com.br/extra/alura-mais/o-que-e-mensageria--c1136>

OBS: vamos tratar isso nos próximos cursos.



Pub / Sub

O Pub/Sub é um exemplo... eu tenho um publicador da mensagem e outros que vão receber e essa mensagem passa por um Message Broker.

O Message Broker é o carteiro... ele vai entregar as cartas para os destinatários. Eu sou o publisher e os receptores das cartas são os subscribers. Eu não preciso esperar o carteiro entregar e me confirmar que o pessoal recebeu, posso voltar para casa e continuar realizando outras tarefas. Se o pessoal não estiver em casa, o carteiro vai saber lidar com essa situação e vai tentar entregar novamente, tudo de forma assíncrona.

Essa é a ideia de mensageria.

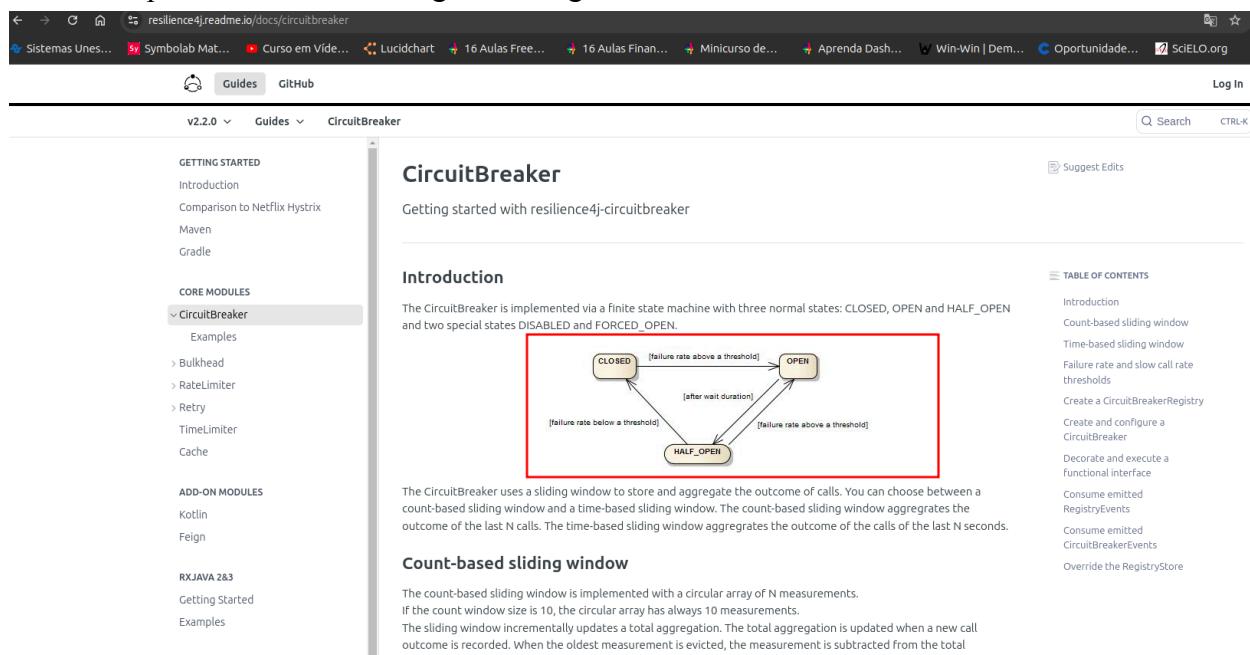
Agora, vamos ver sobre circuit breaker.

Por exemplo, no caso anterior, imaginando que o microsserviço de pedidos estivesse fora, como ficaria? Posso causar uma falha em cascata na aplicação por conta dessa indisponibilidade.

Para isso, vamos usar uma técnica de circuit breaker.

E para tanto, vamos usar o resilience4j, que trabalha muito bem junto ao spring.

Ele é exemplificado através da seguinte imagem:



Um circuit breaker tem 3 estados:

- Fechado: o sistema está operando perfeitamente, e todas as comunicações estão sendo feitas com sucesso.
- Open: quando temos uma taxa de falhas acima do que determinamos como “normal”. Para de tentar comunicar com o serviço que está falhando. E dentro disso, temos o fallback, que é basicamente o que o circuit breaker vai fazer caso esteja nesse estado open.
- Half-open: semi aberto, ou seja, as requisições começam a ser mandadas novamente, e caso ele estiver com sucesso, volta para o estado de fechado, mas se caso continuar dando falhas, volta para o estado de aberto.

Temos todos esses fluxos que o resilience4J já implementa para nós automaticamente.

Basicamente incluiremos 2 dependências no projeto e temos um sistema que podemos fazer o controle por instâncias. Então cada tentativa, cada método de comunicação com um serviço, posso dar um nome para essa instância e configurá-lo de acordo com minhas necessidades.

Getting Started

Getting started with resilience4j-spring-boot2

Setup

Add the Spring Boot 2 Starter of Resilience4j to your compile dependency.

The module expects that `org.springframework.boot:spring-boot-starter-actuator` and `org.springframework.boot:spring-boot-starter-aop` are already provided at runtime. If you are using webflux with spring boot2, you also need `io.github.resilience4j:resilience4j-reactor`.

```

Groovy
repositories {
    jCenter()
}

dependencies {
    compile "io.github.resilience4j:resilience4j-spring-boot2:${resilience4jVersion}"
    compile("org.springframework.boot:spring-boot-starter-actuator")
    compile('org.springframework.boot:spring-boot-starter-aop')
}

```

Demo

Setup and usage in Spring Boot 2 is demonstrated into a [demo](#).

Configuration

You can configure your CircuitBreaker, Retry, RateLimiter, Bulkhead, Thread pool bulkhead and TimeLimiter instances in Spring Boot's `application.yml` config file.

For example

Dependências que devo inserir.

E agora vamos inserir no pom.xml do microsserviço de pagamentos.

```

<dependency>
    <groupId>io.github.resilience4j</groupId>
    <artifactId>resilience4j-spring-boot2</artifactId>
    <version>1.7.0</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>

```

Inseri as duas dependências.

E agora, vamos configurar no método onde eu chamo o microsserviço de pedidos.

```


Project: pagamentos



File: PagamentoController.java



```

20 public class PagamentoController {
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 }

```


```

Então, onde eu chamo de fato o método que faz comunicação com outro microsserviço, o de pedidos, é no controller.

Dessa forma, necessito colocar essa anotação (CircuitBreaker), com um nome, que eu escolho, e o fallbackMethod por enquanto deixo sem nada, deixo vazio.

Além disso, precisamos inserir algumas propriedades no application.properties, para fazer configuração da instância:

```

spring.application.name=pagamentos-ms

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/alurafood-pagamento?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=Joaovictor300!
spring.jpa.show-sql=true

eureka.client.serviceUrl.defaultZone=http://localhost:8081/eureka
server.port=0

resilience4j.circuitbreaker.instances.atualizaPedido.slidingWindowSize: 3
resilience4j.circuitbreaker.instances.atualizaPedido.minimumNumberOfCalls: 2
resilience4j.circuitbreaker.instances.atualizaPedido.waitDurationInOpenState: 50s

```

O nome da instância é atualizaPedido (mesmo nome que foi definido anteriormente).

A primeira (slidingWindowSize) é para saber com quantas requisições ele vai fazer uma estatística, para passar de aberto para fechado, etc... ou seja, para passar de um estado para outro (inserimos 3).

A segunda (minimumNumberOfCalls) é para saber quantas requisições o circuit breaker vai começar a atuar. No caso, ele vai tentar fazer 2... na terceira já bloqueia.

A terceira (waitDurationInOpenState) é quanto tempo vai manter em estado aberto.

Para testar, vou derrubar o de Pedidos.

The screenshot shows the Spring Eureka dashboard at localhost:8081. The top navigation bar includes links to Sistemas Unes..., Symbolab Mat..., Curso em Vídeo..., Lucidchart, 16 Aulas Free..., and All Bookmarks. The main header features the Spring Eureka logo and a "Toggle navigation" button.

System Status

Environment	test	Current time	2024-09-14T09:15:41 -0300
Data center	default	Uptime	02:54
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	20

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - 192.168.100.5:gateway:8082
PAGAMENTOS-MS	n/a (1)	(1)	UP (1) - 192.168.100.51:pagamentos-ms:0

General Info

Name	Value
total-avail-memory	312mb
num-of-cpus	4
current-memory-usage	101mb (32%)
server-upptime	02:54
registered-replicas	http://localhost:8081/eureka/
unavailable-replicas	http://localhost:8081/eureka/,

Desliguei o de pedidos.

Tentando fazer requisição para ele:

The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with navigation links: 'Requisições' (Requests), 'Base Environment', 'Add Cookies', 'Add Certificates', 'Filter', 'GET Identificar a porta - Lo...', 'GET Listar Pedidos', and 'POST Cadastrar Pedidos'. The main area has tabs for 'Parameters', 'Body', 'Auth', 'Headers', 'Docs', and 'URL PREVIEW' which displays 'http://localhost:8082/pedidos-ms/pedidos'. Below these are sections for 'QUERY PARAMETERS' (with 'Import from URL' and 'Bulk Edit' buttons) and 'PATH PARAMETERS' (with a note about colon-segment path parameters). At the top right, the status bar shows '500 Internal Server Error', '3.12 s', and '145 B'. The bottom right contains 'Preview', 'Headers', 'Cookies', and 'Timeline' buttons.

```
1v {  
2  "timestamp": "2024-09-14T12:16:15.375+00:00",  
3  "path": "/pedidos-ms/pedidos",  
4  "status": 500,  
5  "error": "Internal Server Error",  
6  "requestId": "64eeb570a-19"  
7 }
```

Está fora mesmo.

Agora, vamos ver o comportamento do circuit breaker:

Chamando o método do de pagamentos que chama o de pedidos

Primeira vez deu erro.

Segunda tbm.

Na terceira, já abriu o circuito.

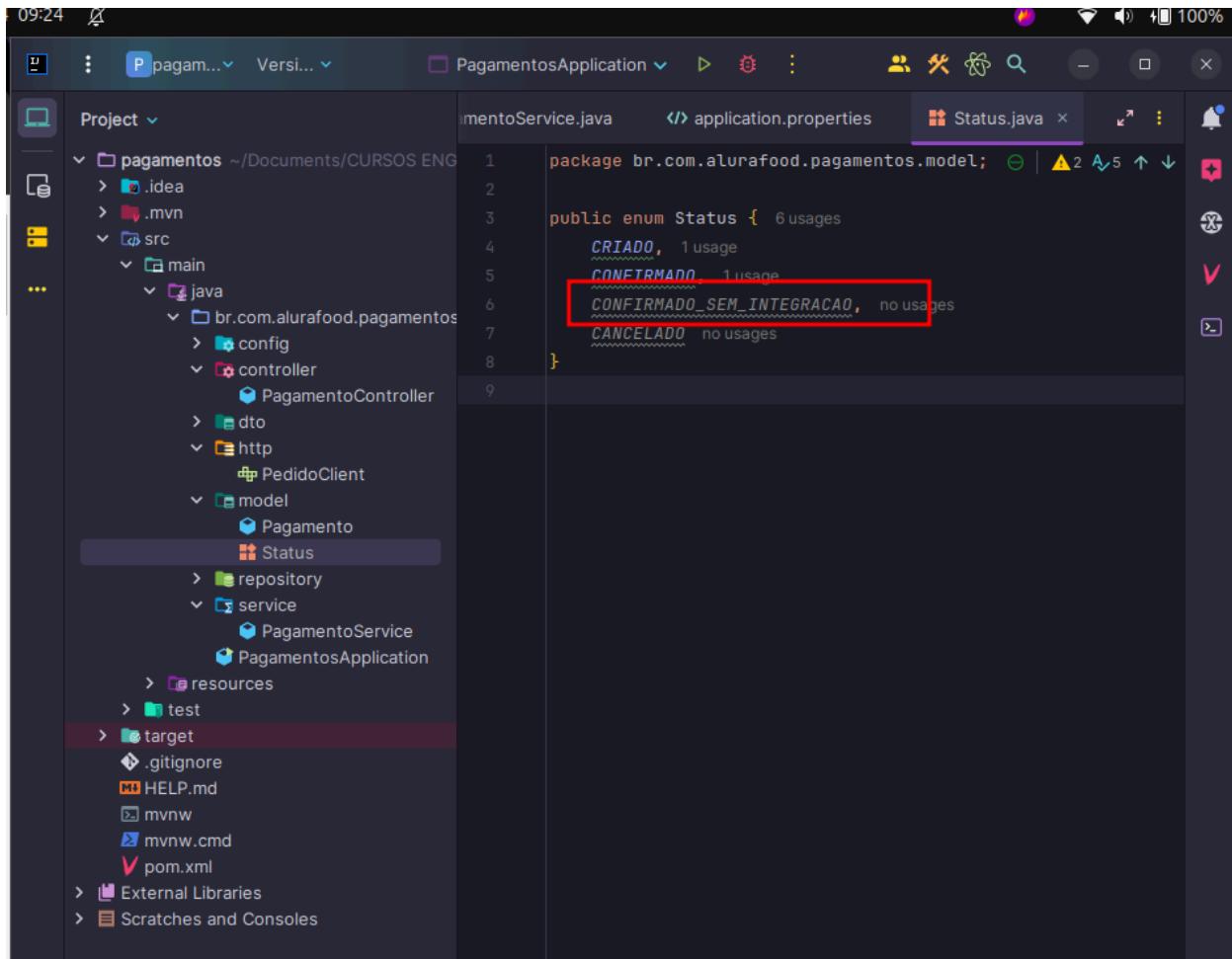
Ok, o circuit breaker está funcionando!

Mas, temos que ter o plano B de quando o circuito abre... que é o que chamamos de FallBack.

Ou seja, quando o circuito abre, o que vou fazer? Nesse caso, o pagamento foi confirmado, então o pedido deveria ser alterado, mas eu não consegui avisar o pedido.

Uma ideia é ter um Status Diferenciado (aprovado mas não integrado), que significa que o pagamento está aprovado, só que ainda não consegui avisar para o pedido e em segundo momento, poderíamos ter um serviço rodando em background e verificando todos os caras que estão com esse status e tentar atualizá-lo de forma assíncrona.

Vamos implementar essa ideia:



```
09:24 P pagam... Versi... PagamentosApplication Project : pagamentos ~/Documents/CURSOS ENGENHARIA DE SOFTWARE - 2023.1 /src/main/java br.com.alurafood.pagamentos config controller PagamentoController dto http PedidoClient model Pagamento Status repository service PagamentoService PagamentosApplication resources test target .gitignore HELP.md mvnw mvnw.cmd pom.xml External Libraries Scratches and Consoles
```

```
09:24 P pagam... Versi... PagamentosApplication Project : pagamentos ~/Documents/CURSOS ENGENHARIA DE SOFTWARE - 2023.1 /src/main/java br.com.alurafood.pagamentos config controller PagamentoController dto http PedidoClient model Pagamento Status repository service PagamentoService PagamentosApplication resources test target .gitignore HELP.md mvnw mvnw.cmd pom.xml External Libraries Scratches and Consoles
```

```
1  package br.com.alurafood.pagamentos.model;
2
3  public enum Status {
4      CRIADO, 1 usage
5      CONFIRMADO, 1 usage
6      CONFIRMADO_SEM_INTEGRACAO, no usages
7      CANCELADO
8  }
```

Inserimos mais um status nos pagamentos.

Agora, vamos na controller, onde indicamos o circuit breaker e vamos indicar um nome para o meu fallbackMethod.

E esse método de Fallback que será criado, precisa ter a mesma assinatura do método de confirmarPagamento (método onde eu configurei o circuit breaker e o fallback), tem que ser do tipo VOID e receber um Long id de parâmetro. E posso ter um parâmetro adicional que é uma Exception, que caso não consiga realizar o fallback, lança a exceção.

```

public class PagamentoController {
    ...
    @PostMapping("/{id}/confirmar")
    public ResponseEntity<PagamentoDto> atualizar(@PathVariable @NotNull Long id, @RequestBody @Valid PagamentoDto pagamentoDto, @NotNull UriComponentsBuilder uriBuilder) {
        PagamentoDto dto = service.criarPagamento(pagamentoDto);
        URI endereco = uriBuilder.path("/pagamentos/{id}").buildAndExpand(ultimoValor).toUri();
        return ResponseEntity.created(location(endereco).body(dto));
    }

    @PutMapping("/{id}")
    public ResponseEntity<PagamentoDto> atualizar(@PathVariable @NotNull Long id, @RequestBody @Valid PagamentoDto pagamentoDto) {
        PagamentoDto dto = service.atualizarPagamento(id, pagamentoDto);
        return ResponseEntity.ok(dto);
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> remover(@PathVariable @NotNull Long id) {
        service.excluiPagamento(id);
        return ResponseEntity.noContent().build();
    }

    @PatchMapping("/{id}/confirmar")
    public ResponseEntity<Void> confirmarPagamento(@PathVariable @NotNull Long id) {
        service.confirmarPagamento(id);
        return ResponseEntity.noContent().build();
    }

    public void pagamentoAutorizadoComIntegracaoPendente(Long id, Exception e) { no usages
        service.alteraStatus(id);
    }
}

```

Ele não precisa de anotações, etc... é só um método mesmo.

Na service, criando esse método:

```

public class PagamentoService {
    ...
    public PagamentoDto atualizarPagamento(Long id, PagamentoDto dto) {
        Pagamento pagamento = modelMapper.map(sourceDto, destinationType(Pagamento.class));
        pagamento.setId(id);
        pagamento = repository.save(entity(pagamento));
        return modelMapper.map(pagamento, destinationType(PagamentoDto.class));
    }

    public void excluirPagamento(Long id) {
        repository.deleteById(id);
    }

    public void confirmarPagamento(Long id) {
        Optional<Pagamento> pagamento = repository.findById(id);
        if (pagamento.isEmpty()) {
            throw new EntityNotFoundException();
        }

        pagamento.get().setStatus(Status.CONFRMADO);
        repository.save(entity(pagamento.get()));
        pedido.atualizaPagamento(id, pagamento.get().getPedidoId());
    }

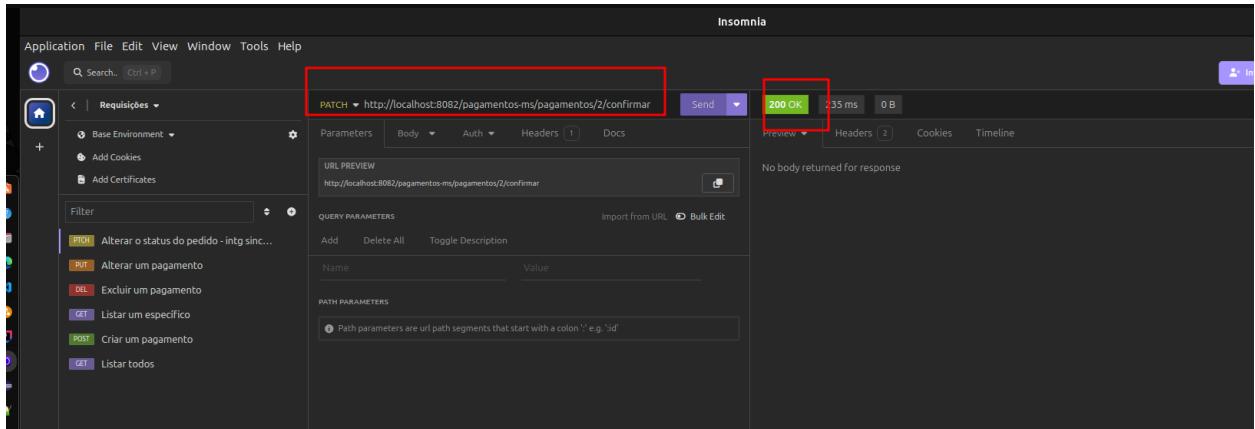
    public void alteraStatus(Long id) { 1 usage
        Optional<Pagamento> pagamento = repository.findById(id);
        if (pagamento.isEmpty()) {
            throw new EntityNotFoundException();
        }

        pagamento.get().setStatus(Status.CONFRMADO_SEM_INTEGRACAO);
        repository.save(entity(pagamento.get()));
    }
}

```

É quase a mesma coisa do método acima dele, mas sem chamar o serviço de atualizar pagamentos.

Vamos fazer um teste agora:

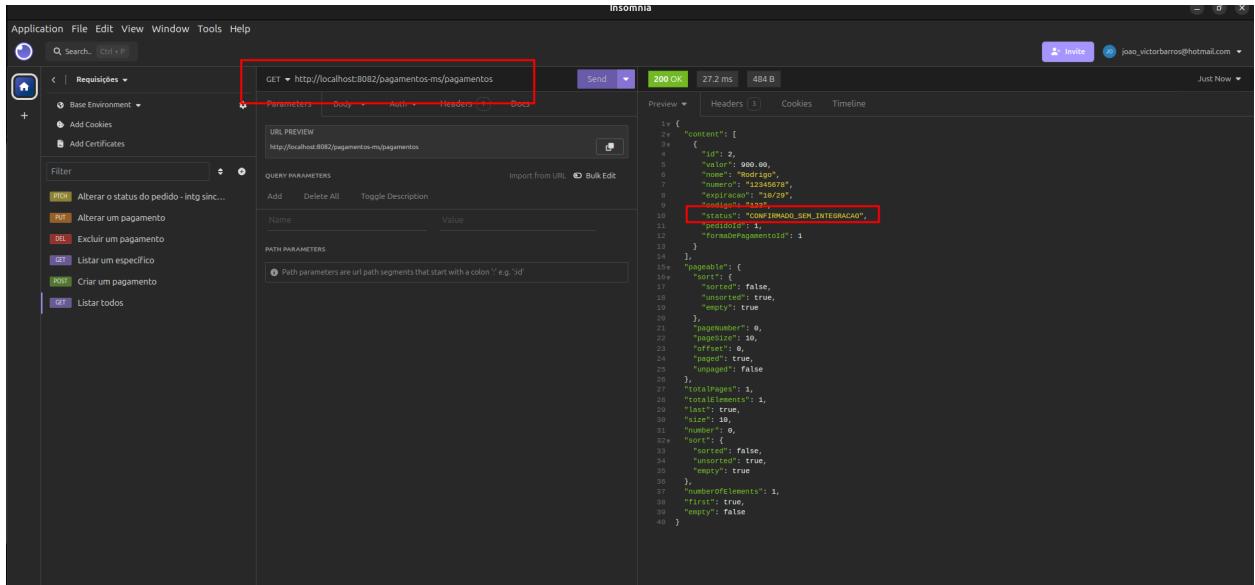


The screenshot shows the Insomnia REST client interface. A red box highlights the URL field containing "PATCH http://localhost:8082/pagamentos-ms/pagamentos/2/confirmar". Another red box highlights the status bar at the top right showing "200 OK". The main panel displays the response body: "No body returned for response".

O ms de pedidos continua fora.

Mas agora, o método de atualizar o pagamento foi chamado e retornou 200.

Logo, ele atuou no fallback.



The screenshot shows the Insomnia REST client interface. A red box highlights the URL field containing "GET http://localhost:8082/pagamentos-ms/pagamentos". Another red box highlights the status bar at the top right showing "200 OK". The main panel displays the response body as a JSON object:

```
1 {  
2     "content": [  
3         {  
4             "id": 2,  
5             "value": "900.00",  
6             "name": "Cartão de Crédito",  
7             "number": "12345678",  
8             "expiration": "10/29",  
9             "image": "IMAGEM SEM INTEGRACAO",  
10            "paid": true,  
11            "formadePagamentoId": 1  
12        }  
13    ],  
14    "pageable": {  
15        "sort": {  
16            "sorted": false,  
17            "unsorted": true,  
18            "empty": true  
19        },  
20        "pageNumber": 0,  
21        "pageSize": 10,  
22        "offset": 0,  
23        "paged": true,  
24        "unpaged": false  
25    },  
26    "totalPages": 1,  
27    "totalElements": 1,  
28    "last": true,  
29    "size": 10,  
30    "number": 0,  
31    "sort": {  
32        "sorted": false,  
33        "unsorted": true,  
34        "empty": true  
35    },  
36    "numberOfElements": 1,  
37    "first": true,  
38    "empty": false  
39 }  
40 }
```

E aí teríamos que ter um serviço de background rodando para visualizar todos esses pagamentos que estão com esse status e tentar realizar novamente a atualização do pedido.

Agora vamos subir a aplicação de pedidos para ver se vai realizar corretamente...

The screenshot shows the Insomnia REST client interface. A PATCH request is made to `http://localhost:8082/pagamentos-ms/pagamentos/2/confirmar`. The response is successful (200 OK) with a duration of 13.2 ms and 0 B. The response body is empty.

Chamei novamente.

Mudou o status do pagamento para confirmado:

The screenshot shows the Insomnia REST client interface. A GET request is made to `http://localhost:8082/pagamentos-ms/pagamentos`. The response is successful (200 OK) with a duration of 14.2 ms and 469 B. The response body is a JSON object representing a payment, with the `"status": "CONFIRMADO"` field highlighted in red.

```

1y {
2y   "content": [
3y     {
4y       "id": 2,
5y       "valor": 990.00,
6y       "nome": "Rodrigo",
7y       "numero": "12345678",
8y       "expiracao": "10/29",
9y       "status": "PAGO",
10y      "status": "CONFIRMADO",
11y      "pedidoId": 1,
12y      "formadoPagamentoId": 1
13y    }
14y  ],
15y  "pageable": {
16y    "sort": {
17y      "sorted": false,
18y      "unsorted": true,
19y      "empty": true
20y    },
21y    "pageNumber": 0,
22y    "pageSize": 10,
23y    "offset": 0,
24y    "paged": true,
25y    "unpaged": false
26y  },
27y  "totalPages": 1,
28y  "totalElements": 1,
29y  "last": true,
30y  "size": 10,
31y  "number": 0,
32y  "sort": {
33y    "sorted": false,
34y    "unsorted": true,
35y    "empty": true
36y  },
37y  "numberOfElements": 1,
38y  "first": true,
39y  "empty": false
40y }

```

E atualizou o pedido:

The screenshot shows the Insomnia REST client interface. A GET request is made to `http://localhost:8082/pedidos-ms/pedidos`. The response is successful (200 OK) with a duration of 16.1 ms and 167 B. The response body is a JSON array of orders, with the `"descricao": "coca-cola"` field highlighted in red.

```

1y [
2y   {
3y     "id": 1,
4y     "data": "2024-09-13T10:10:51",
5y     "status": "PAGO",
6y     "items": [
7y       {
8y         "id": 1,
9y         "quantidade": 10,
10y        "descricao": "coca-cola"
11y      },
12y      {
13y        "id": 2,
14y        "quantidade": 5,
15y        "descricao": "Mc Chicken"
16y      }
17y    ]
18y  ]

```

Deu certo.

Então é imprescindível aplicar os conceitos de resiliência nos microserviços (um exemplo é o circuit breaker e fallback).

Conhecimento: comunicação síncrona

06 Comunicação síncrona

 PRÓXIMA ATIVIDADE

João percebeu que tem um método específico do seu projeto que precisa da resposta imediata do outro microsserviço de **Clientes**, e por isso optou por implementar a comunicação síncrona usando o Open Feign. Se ele deseja recuperar por exemplo um status sobre o cliente, se está adimplente ou não, qual dessas implementações seria a mais apropriada para o caso?



```
@FeignClient("clientes-ms")
public interface ClienteHttpClient {
    @RequestMapping(method = RequestMethod.GET, value = "/clientes/{id}")
    Status verificaAdimplencia(@PathVariable Long id);
}
```

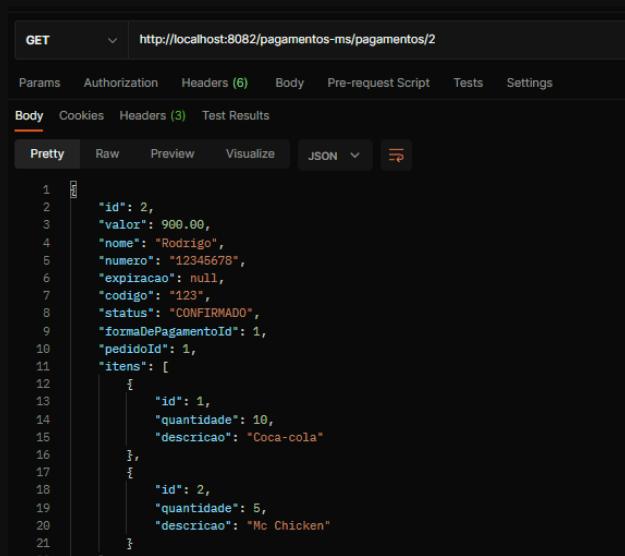
Alternativa correta! Sim, esse é um exemplo de implementação que vai fazer uma requisição do tipo GET para outro microsserviço de clientes, passando o id do cliente que se deseja consultar, e irá retornar um status, de adimplência ou não.

Desafio: fazer essa implementação:

Concluímos nosso projeto e fizemos coisas incríveis, como permitir que um microsserviço faça a comunicação com outro, mudando um status de um pedido. Ainda prevenimos as possíveis falhas quando o serviço de pedidos estava inoperante. Demais, não é mesmo? Que tal agora fazermos algo parecido com a nossa atividade avaliativa e recuperar algum dado do microsserviço de pedidos ao fazer a consulta do pagamento?

O desafio é o seguinte: quando eu consultar um pagamento através da URI

<http://localhost:8082/pagamentos-ms/pagamentos/2> por exemplo, deverá listar todos os dados do pagamento e os itens que compõem o pedido referente a esse pagamento, conforme a imagem abaixo:



A screenshot of a Postman request interface. The method is GET, and the URL is <http://localhost:8082/pagamentos-ms/pagamentos/2>. The response body is displayed in Pretty JSON format:

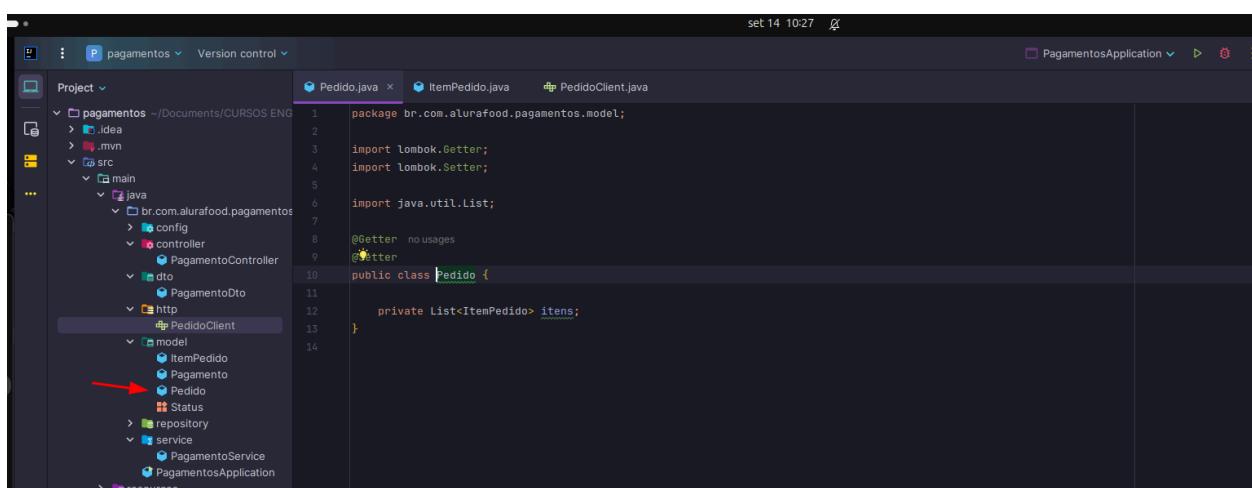
```
1  {
2      "id": 2,
3      "valor": 900.00,
4      "nome": "Rodrigo",
5      "numero": "12345678",
6      "expiracao": null,
7      "codigo": "123",
8      "status": "CONFIRMADO",
9      "formaDePagamentoId": 1,
10     "pedidoId": 1,
11     "itens": [
12         {
13             "id": 1,
14             "quantidade": 10,
15             "descricao": "Coca-cola"
16         },
17         {
18             "id": 2,
19             "quantidade": 5,
20             "descricao": "Mc Chicken"
21         }
22     ]
23 }
```

Que tal? Vamos implementar juntos?

Obs: não inserimos técnicas de resiliência aqui, somente integração síncrona entre serviços.

Para isso, basta fazer:

- Criei os modelos:



```

set 14 10:28 ✘
Project ▾ pagamentos ~/Documents/CURSOS ENG Version control ▾
  > idea
  > mvn
  > src
    > main
      > java
        > br.com.alurafood.pagamentos
          > config
          > controller
            > PagamentoController
          > dto
            > PagamentoDto
          > http
            > PedidoClient
              > model
                > ItemPedido
                > Pagamento
                > Pedido
                > Status
                > repository
                > service
                  > PagamentoService
                  > PagamentosApplication
                > resources
              > test
            > target
            > .gitignore

```

```

 1 package br.com.alurafood.pagamentos.model;
 2
 3 import lombok.Getter;
 4 import lombok.Setter;
 5
 6 @Getter 3 usages
 7 @Setter
 8 public class ItemPedido {
 9
10     private Long id;
11     private Integer quantidade;
12     private String descricao;
13 }
14

```

- Adicionei mais um método na interface http client do microsserviço de pedidos:

```

set 14 10:28 ✘
Project ▾ pagamentos ~/Documents/CURSOS ENG Version control ▾
  > idea
  > mvn
  > src
    > main
      > java
        > br.com.alurafood.pagamentos
          > config
          > controller
            > PagamentoController
          > dto
            > PagamentoDto
          > http
            > PedidoClient
              > model
                > ItemPedido
                > Pagamento
                > Pedido
                > Status
                > repository
                > service
                  > PagamentoService
                  > PagamentosApplication
                > resources
              > test
            > target
            > .gitignore

```

```

 1 package br.com.alurafood.pagamentos.http;
 2
 3 import br.com.alurafood.pagamentos.model.Pedido;
 4 import org.springframework.cloud.openfeign.FeignClient;
 5 import org.springframework.web.bind.annotation.PathVariable;
 6 import org.springframework.web.bind.annotation.RequestMapping;
 7 import org.springframework.web.bind.annotation.RequestMethod;
 8
 9 @FeignClient("pedidos-ms") 2 usages
10 public interface PedidoClient {
11
12     @RequestMapping(method = RequestMethod.PUT, value = "/pedidos/{id}/pago") 1 usage
13     void atualizaPagamento(@PathVariable Long id);
14
15     @RequestMapping(method = RequestMethod.GET, value = "/pedidos/{id}") 1 usage
16     Pagamento buscaItensDoPedido(@PathVariable Long id);
17 }
18

```

- Onde fiz um Get para buscar um pedido em específico.

```

Project ▾ pagamentos ~/Documents/CURSOS ENG Version control ▾
  > idea
  > mvn
  > src
    > main
      > java
        > br.com.alurafood.pagamentos
          > config
          > controller
            > PagamentoController
          > dto
            > PagamentoDto
            > PedidoClient
            > model
              > ItemPedido
              > Pagamento
              > Pedido
              > Status
              > repository
              > service
                > PagamentoService
                > PagamentosApplication
              > resources
            > test
            > target
            > .gitignore
            > HELP.md
            > mvnw
            > mvnw.cmd
            > pom.xml
            > External Libraries
            > Scratches and Consoles

```

```

 1 package br.com.alurafood.pagamentos.dto;
 2
 3 import br.com.alurafood.pagamentos.model.ItemPedido;
 4 import br.com.alurafood.pagamentos.model.Status;
 5 import lombok.Getter;
 6 import lombok.Setter;
 7 import java.math.BigDecimal;
 8 import java.util.List;
 9
10 @Getter 22 usages
11 @Setter
12 public class PagamentoDto {
13
14     private Long id;
15     private BigDecimal valor;
16     private String nome;
17     private String numero;
18     private String expiracao;
19     private String codigo;
20     private Status status;
21     private Long formaDePagamentoId;
22     private Long pedidoId;
23
24     private List<ItemPedido> itens;
25 }
26

```

- Inseri no dto de PagamentoDto os itens do pedido.

```

public class PagamentoService {
    ...
    public PagamentoDto obterPorId(Long id) { ... }
    public PagamentoDto criarPagamento(PagamentoDto dto) { ... }
    public Page<PagamentoDto> obterTodos(Pageable paginacao) { ... }
}

```

The code in the screenshot shows the implementation of the `PagamentoService`. It includes methods to get a payment by ID and to create a new payment. A red box highlights the addition of code to the `obterPorId` method that retrieves items from the payment.

- E inseri essas linhas no método de `obterPagamentoPorId`, onde inclui no DTO os itens do pedido daquele pagamento.

Testando:

The screenshot shows the Insomnia REST client interface. A GET request is made to `http://localhost:8082/pagamentos-ms/pagamentos/2`. The response status is 200 OK, with a response time of 427 ms and a size of 261 B. The response body is a JSON object containing a payment and its items. A red box highlights the JSON response body.

```

{
  "id": 2,
  "valor": 900.00,
  "nome": "Rodrigo",
  "email": "rodrigo@exemplo.com.br",
  "expiracao": "18/29",
  "codigo": "123",
  "status": "CONFIRMADO",
  "formaPagamentoId": 1,
  ...
  "itens": [
    {
      "id": 1,
      "quantidade": 10,
      "descricao": "Coca-Cola"
    },
    {
      "id": 2,
      "quantidade": 5,
      "descricao": "Mc Chicken"
    }
  ]
}

```

Deu bom!

