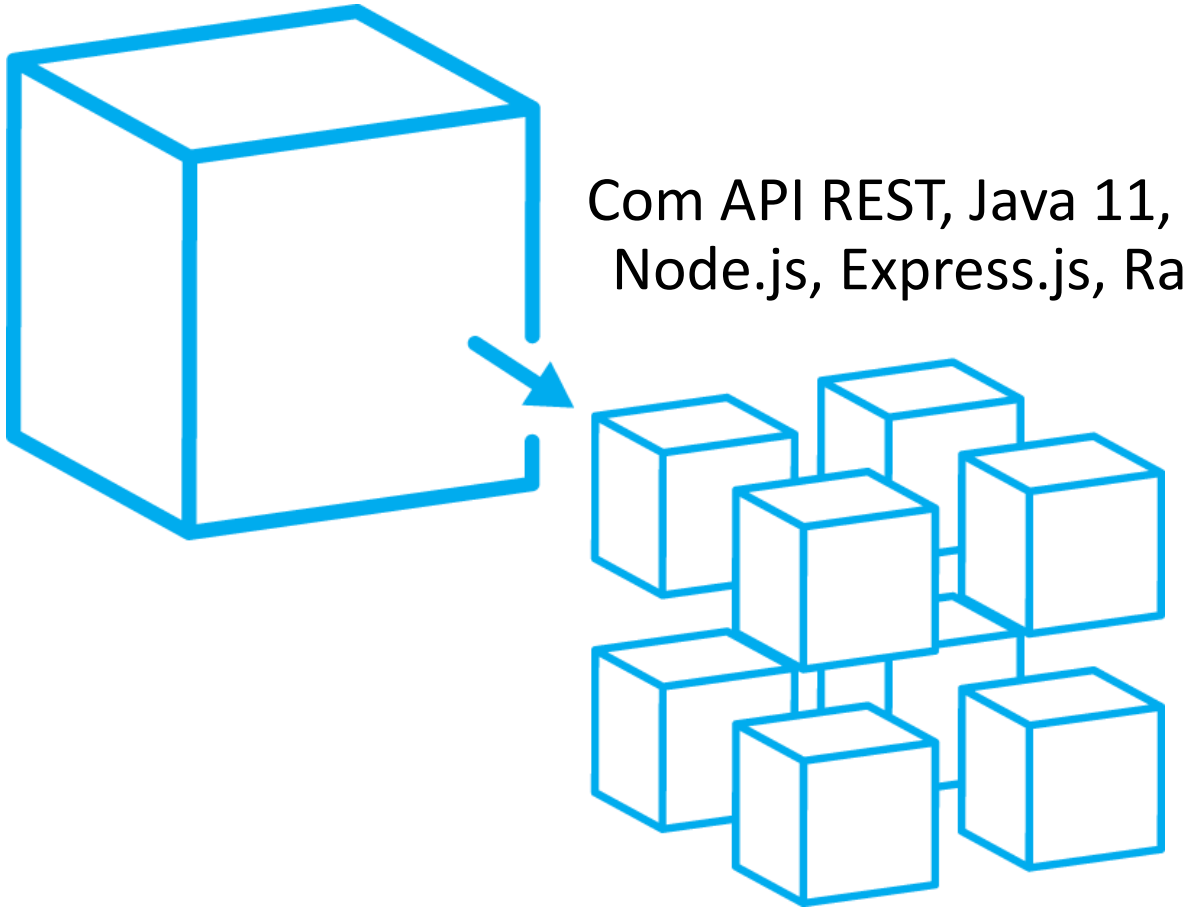
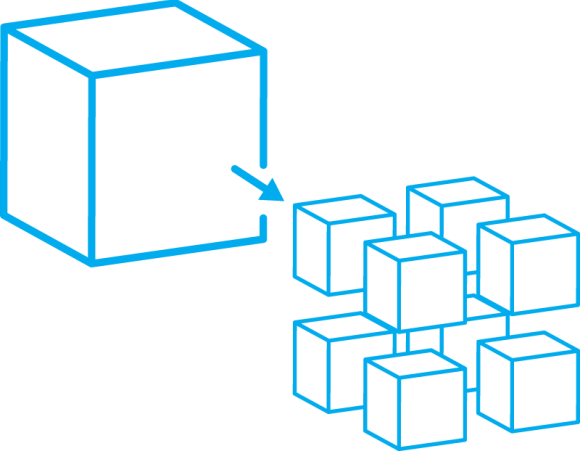
Comunicação entre Microsserviços

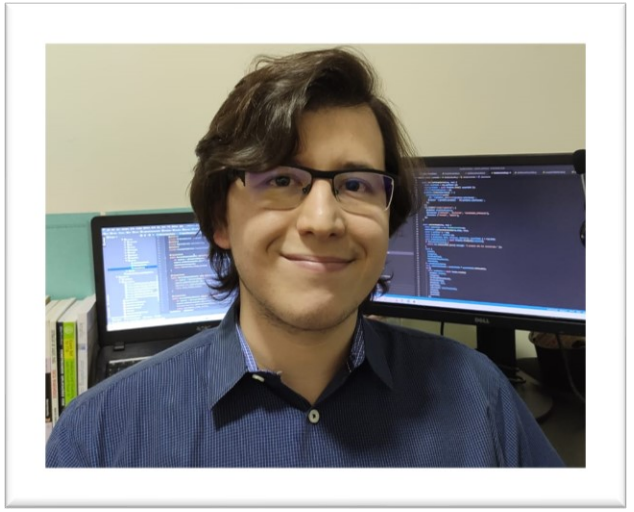
Com API REST, Java 11, Spring Boot, Javascript ES6, Node.js, Express.js, RabbitMQ, Docker e Heroku.



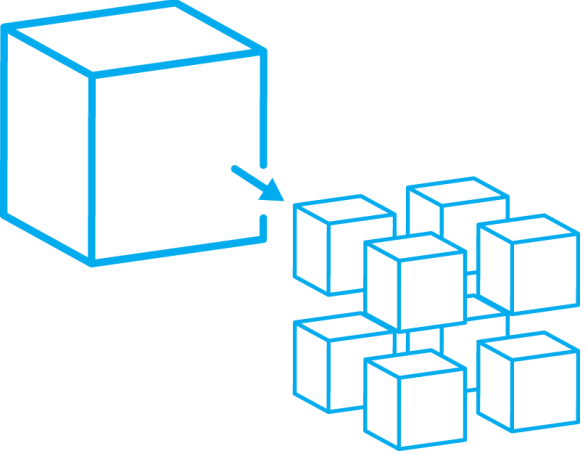
Victor Hugo Negrisoli
Desenvolvedor Back-End Sênior



Quem sou eu?



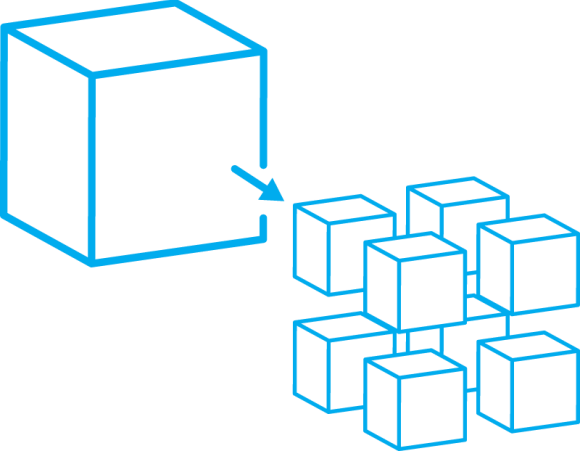
- Victor Hugo Negrisoli
- Formado em Ciência da Computação pelo Centro Universitário Filadélfia (UniFil), Londrina, PR.
- Atualmente fazendo uma pós em Ciência de Dados & Big Data na PUC-MG.
- 3 anos e meio de atuação no mercado de tecnologia
- Atualmente trabalho sou Desenvolvedor Back-End Sênior.
- Atuo com tecnologia Java | Spring e com tecnologias Node.js | Express.js
- Atuo principalmente com desenvolvimento de APIs e microsserviços, comunicações e integrações entre sistemas com Spring Cloud, Apache Kafka, RabbitMQ, entre outros.
- LinkedIn: https://www.linkedin.com/in/victorhugonegrisoli/
- Github: https://github.com/vhnegrisoli
- E-mail: victorhugonegrisoli.ccs@gmail.com
- Telefone: +55 (43) 9 9147-5826



Divisão do curso

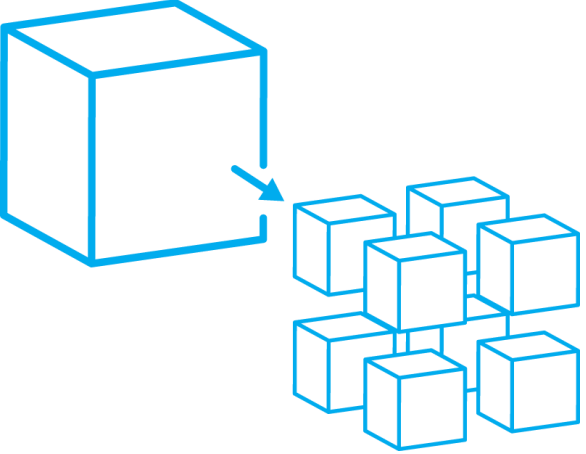
Teoria sobre o que são microsserviços, porquê utilizá-los e quais seus tipos de comunicações.

Prática implementando duas APIs e realizando comunicações síncronas e assíncronas via chamadas HTTP por meio de API REST e fila de mensagens com RabbitMQ. Por fim, iremos subir toda a aplicação no Docker com docker-compose, e iremos disponibilizar também no Heroku.



Tecnologias utilizadas

- API REST
- Java 11 e Spring Boot
- PostgreSQL
- Node.js
- Express.js
- MongoDB
- RabbitMQ
- Docker
- Docker-compose
- Heroku

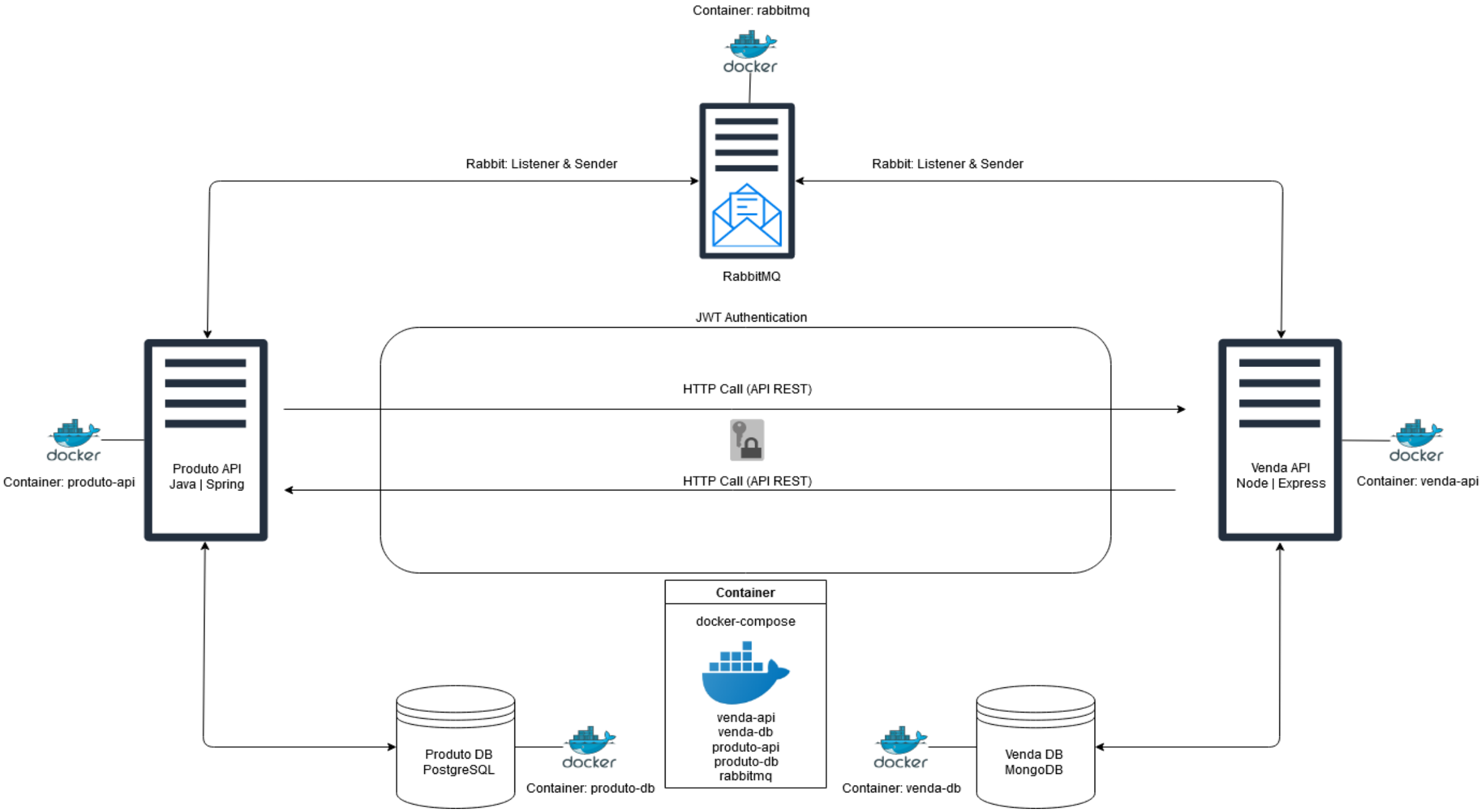


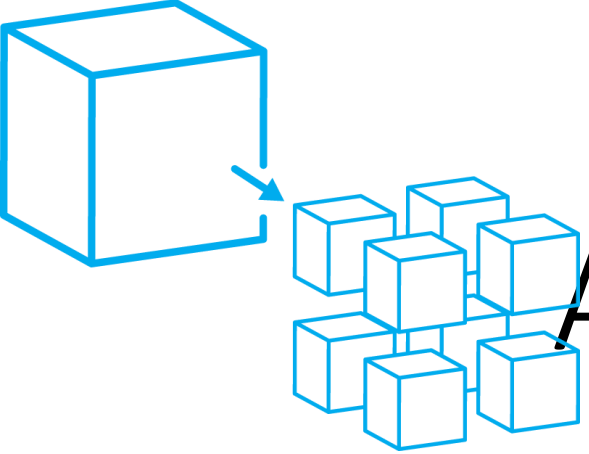
Mas...
O que iremos criar com toda essa stack?!

Boa pergunta!

- Temos que ter uma finalidade!
- Então, iremos simular um pequeno sistema de vendas.
- A API em Node.js irá ser responsável por registrar **vendas**.
- A API em Spring será responsável por cuidar do estoque de **produtos**.
- Toda vez que uma venda for realizada, será enviada uma **mensagem** da API de vendas para a API de produtos para que o estoque seja atualizado.
- Para a realização de cada venda, será necessário **requisitar** para a a API de produtos se o produto consta em estoque, caso contrário, não realizaremos a venda.
- Ao atualizar o estoque de um produto, a gente retorna uma **mensagem** para a aplicação de vendas informando que está tudo ok com a venda que podemos atualizar a venda para CONCLUÍDA.

Diagrama da arquitetura que iremos construir



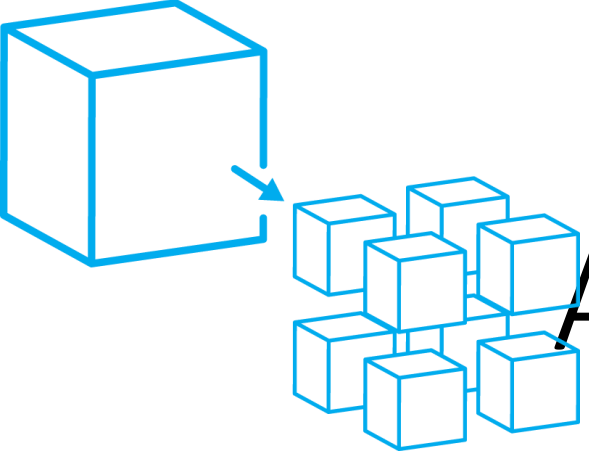


Arquitetura Monolítica

É uma aplicação de software em camadas que consiste em ter uma interface de usuário e a lógica de processamento dos dados em um único programa, geralmente feita a partir de uma única plataforma. Exemplo: Java, C#, PHP, Node.js, entre outros.

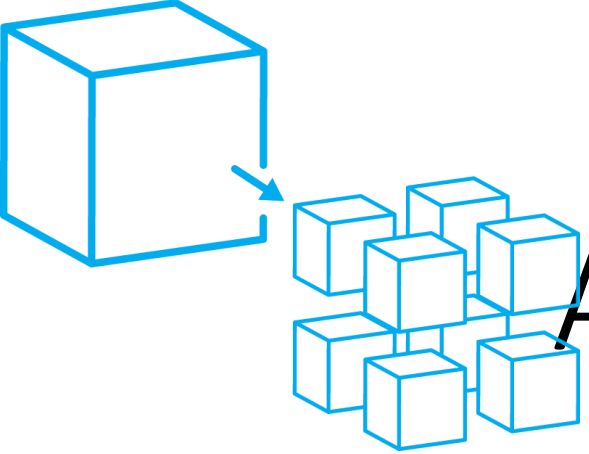
Frameworks comuns para aplicações monolíticas:

- Spring MVC
- .NET Framework
- Java Server Faces (JSF)
- Express.js (com template engines)
- Laravel
- Rails
- Django

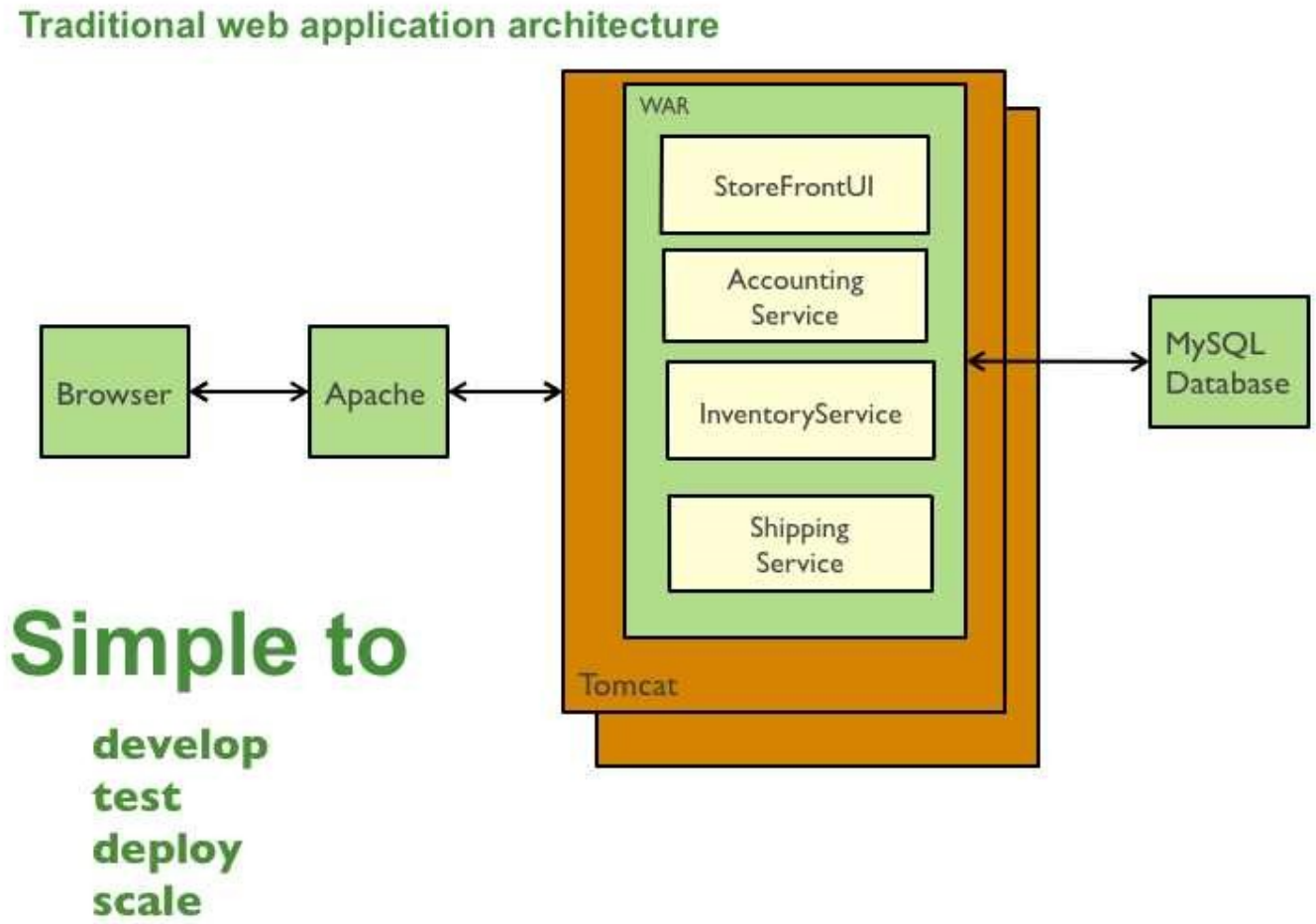


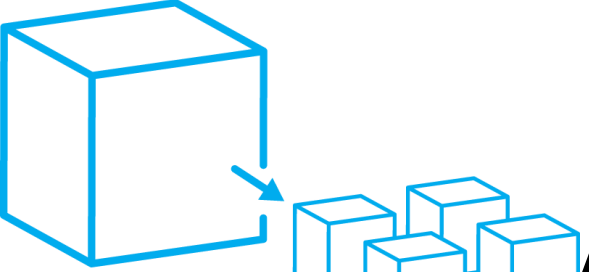
Arquitetura Monolítica

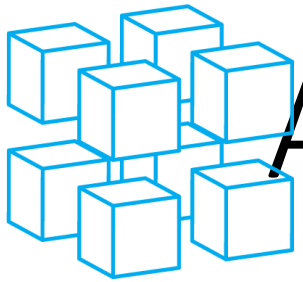
- Back-end e Front-end geralmente são o mesmo projeto e muitas vezes utilizam padrões como MVC (Model, View, Controller).
- Geralmente o back-end é responsável por renderizar as páginas HTML (no caso de aplicação web) ou de realizar a interface gráfica (no caso de aplicações desktop).
- O software por completo fica em apenas um local.
- Builds e alterações no estado devem ser bem preparadas antes, pois ao parar a aplicação, todas as funcionalidades ficam indisponibilizadas até que a aplicação volte a operar.
- Desenvolvimento geralmente mais rápido por manter back-end, banco de dados e front-end na mesma aplicação.
- Geralmente aplicações monolíticas não são modularizadas (não é uma regra).



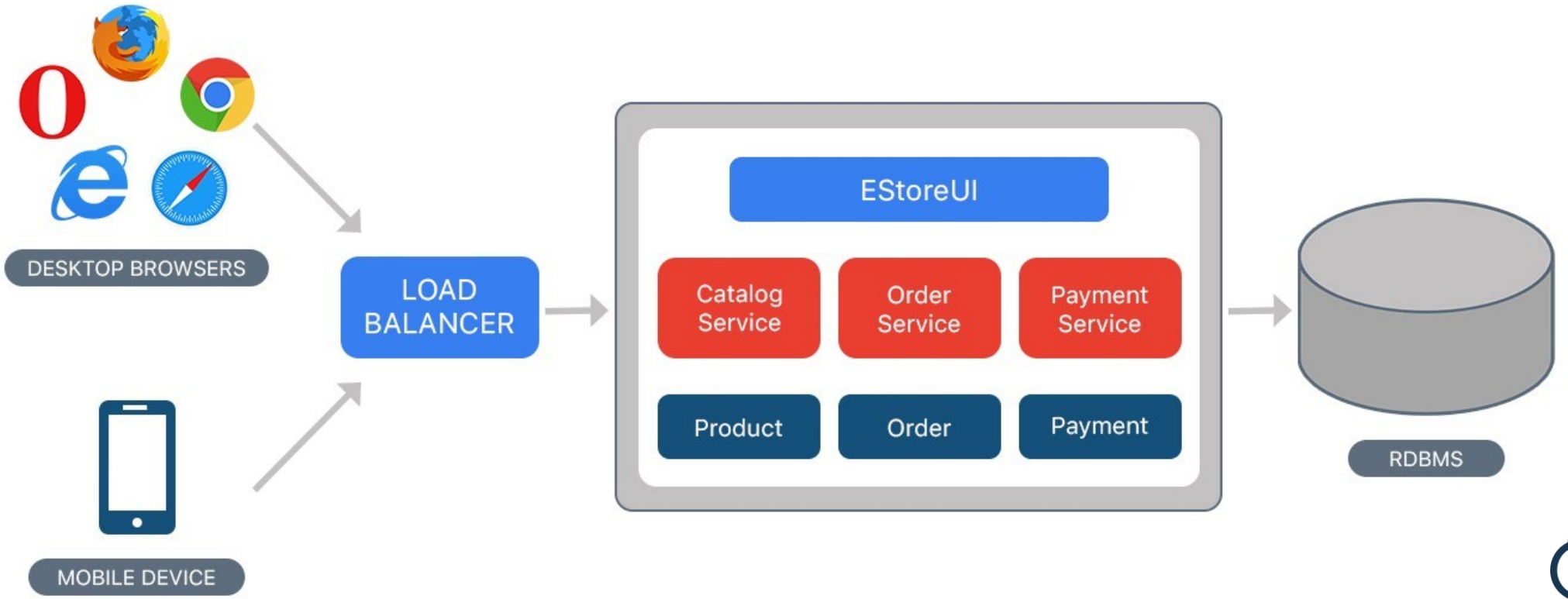
Arquitetura Monolítica

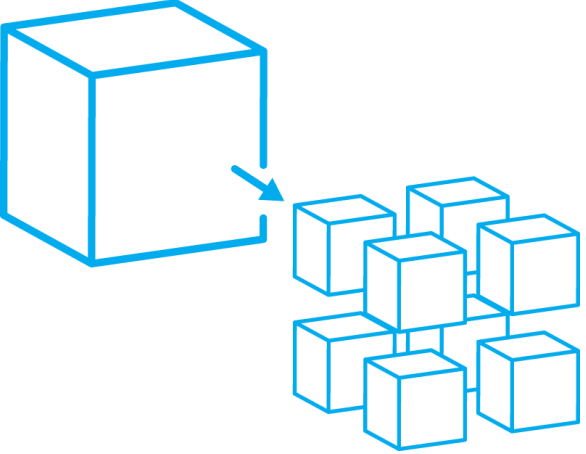






Arquitetura Monolítica

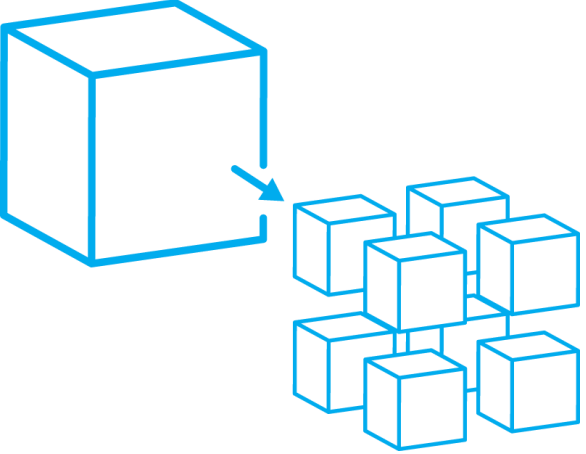




Arquitetura de Microsserviços

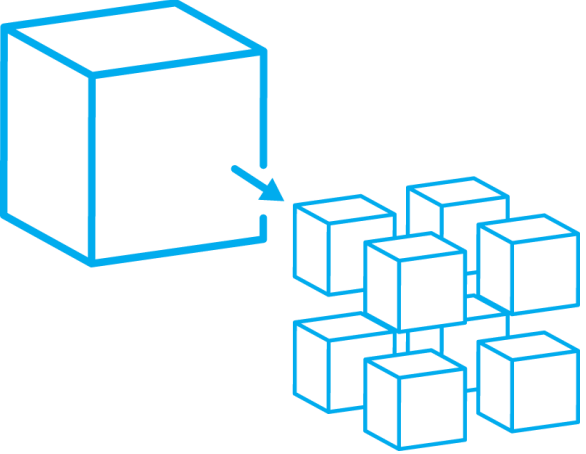
É uma abordagem para o desenvolvimento de software em que uma aplicação é desmembrada em componentes mínimos e independentes.

Diferentemente da abordagem tradicional monolítica em que toda a aplicação é criada como um único bloco, os microsserviços são componentes separados que trabalham juntos para realizar as mesmas tarefas. Cada um dos componentes ou processos é um microsserviço

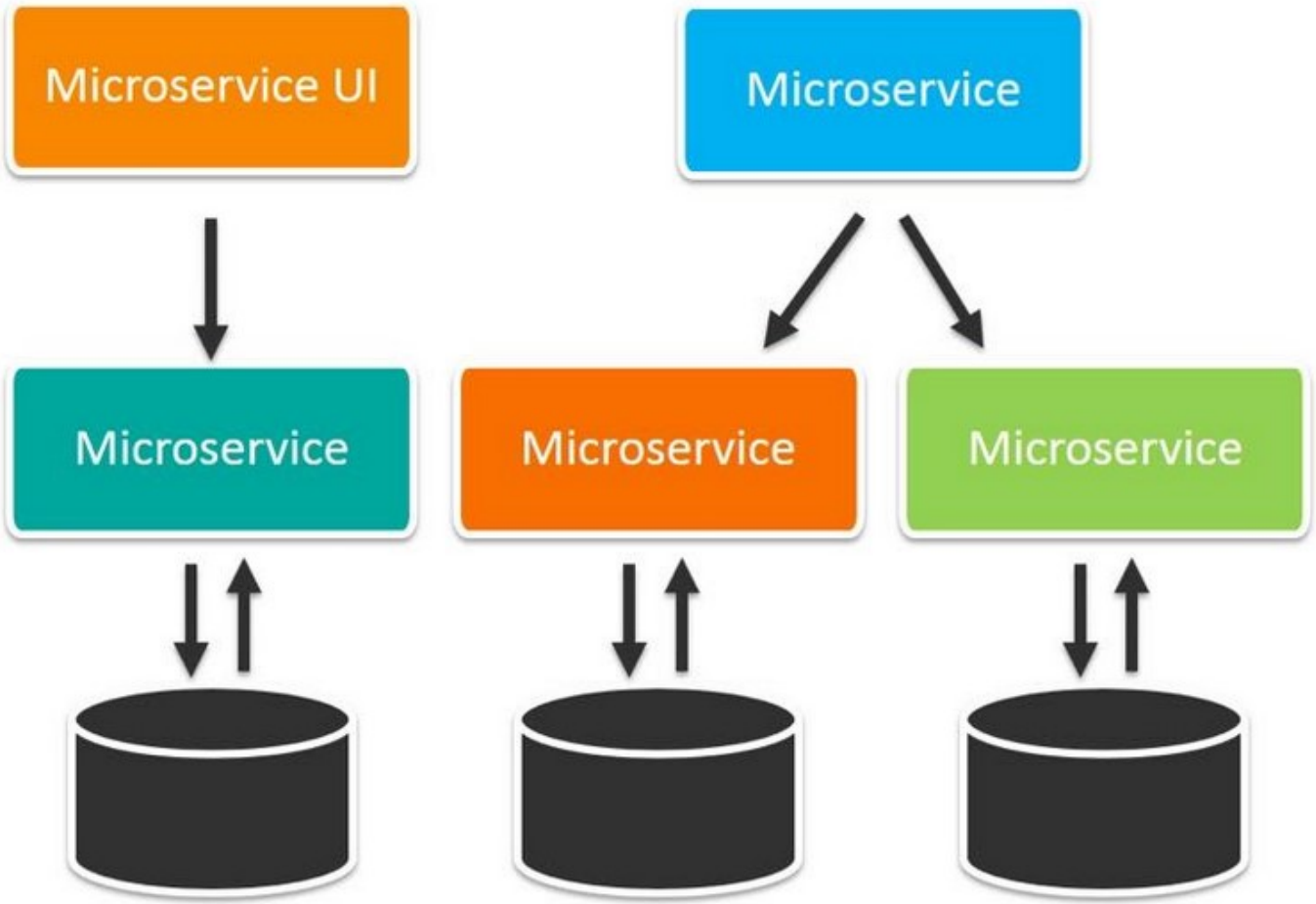


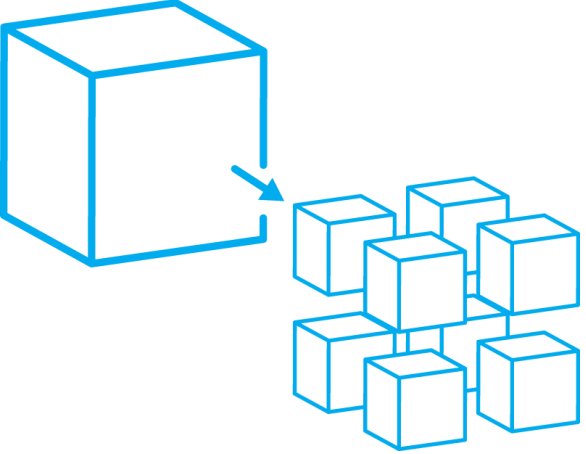
Arquitetura de Microsserviços

- Geralmente são desenvolvidos em formatos de API REST.
- Quando necessário, comunicam-se com outros microsserviços de maneira síncrona e assíncrona.
- São módulos independentes.
- Em uma única solução, pode-se ter diversas tecnologias e linguagens de programação distintas, desde que consigam se comunicar.
- Tem escalabilidade.
- Uma boa prática é que cada microsserviço possua sua própria base de dados para acesso.
- Quando há necessidade de deploy, a aplicação não é totalmente impedida de operar, apenas a funcionalidade ao qual o microsserviço responsável estará temporariamente indisponível.
- Fácil de interagir com diversos sistemas e plataformas.

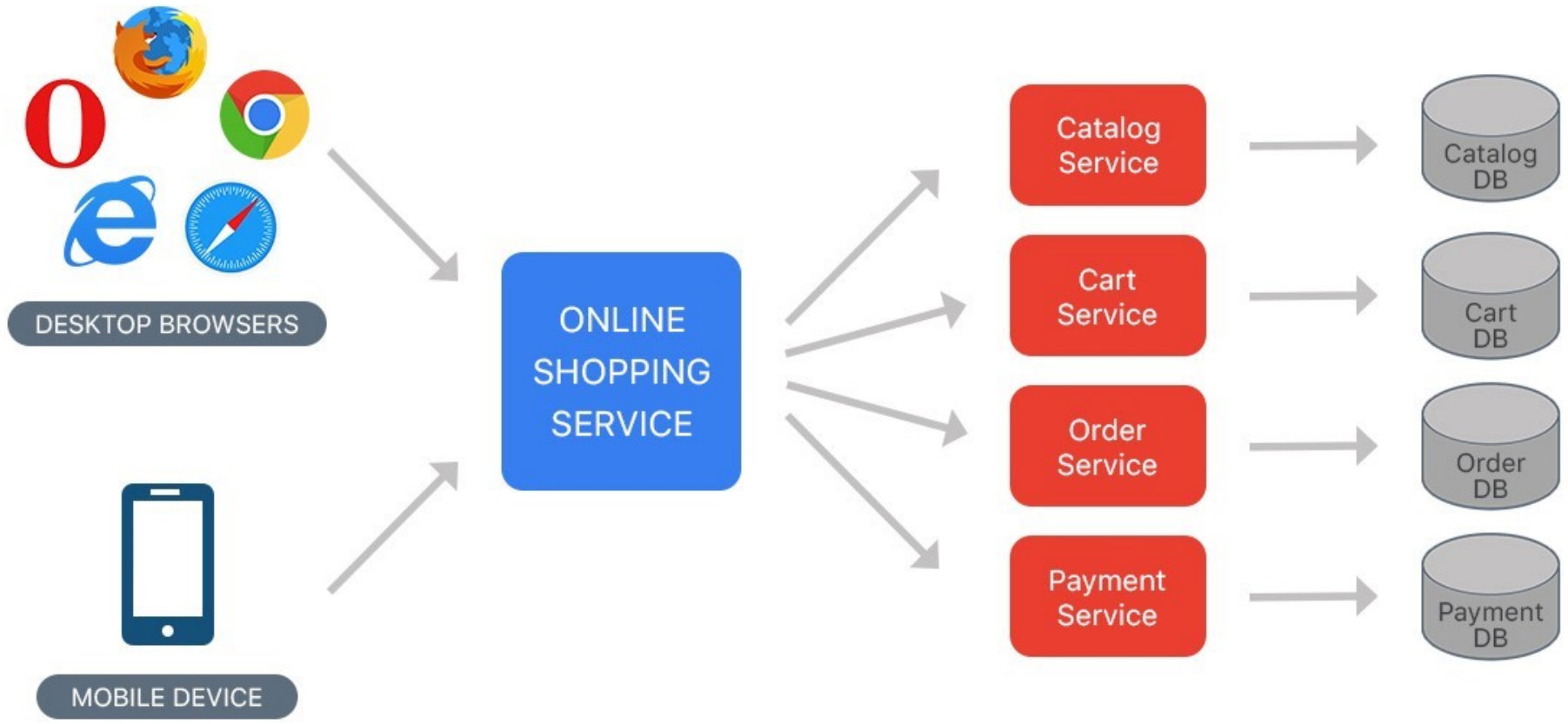


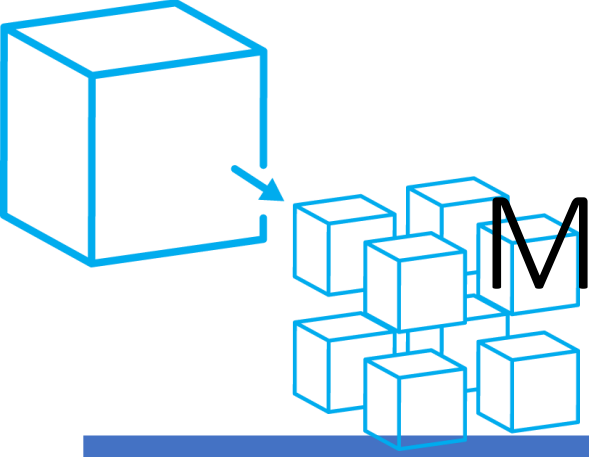
Arquitetura de Microsserviços





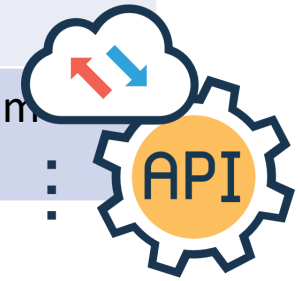
Arquitetura de Microsserviços

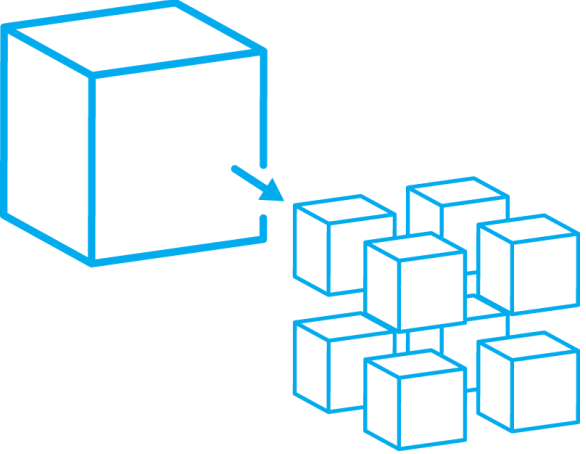




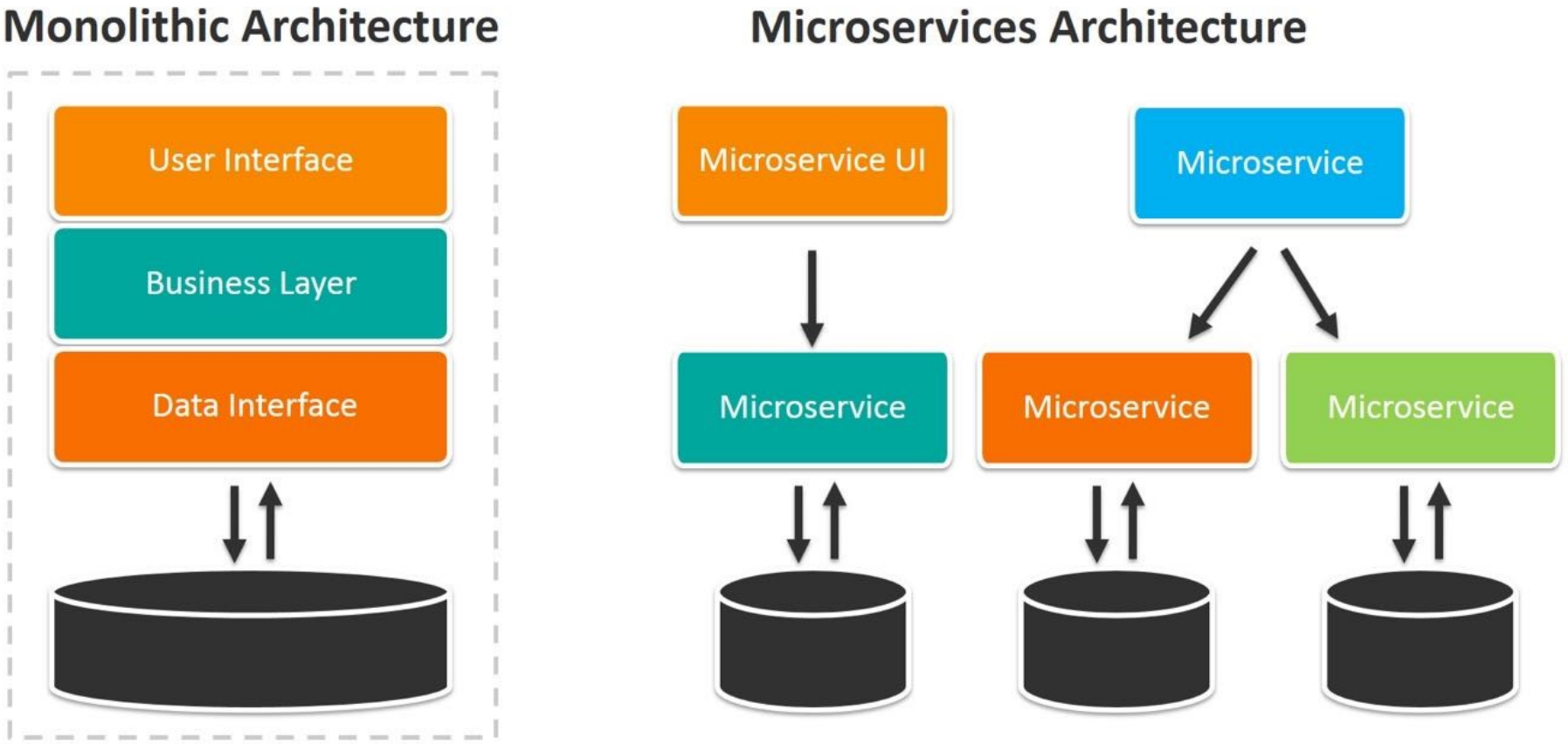
Monolítico vs Microsserviços

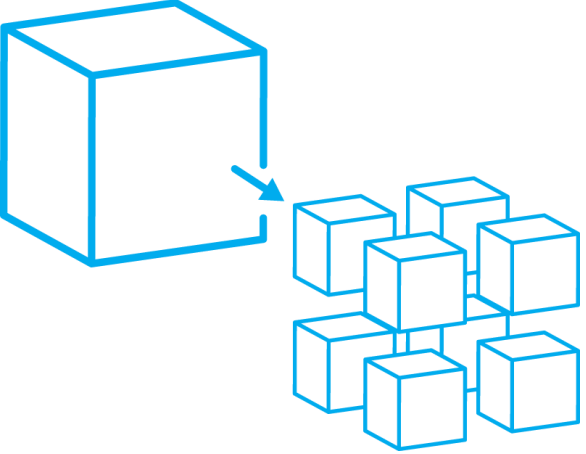
Monolítico	Microsserviços
Desenvolvimento mais rápido para implementar novas funcionalidades.	Conforme cresce o número de serviços, fica mais complexo para implementar novas funcionalidades.
Dificuldade com escalabilidade e necessidade de manutenções.	Escalável e possibilita manutenções diárias sem grandes impactos.
Toda a aplicação é desenvolvida em apenas um bloco contendo front-end, lógica de back-end e banco de dados.	O front-end geralmente é uma aplicação web própria que consome os dados dos microsserviços. Back-ends separados.
Grande acoplamento entre as dependências da aplicação.	Baixo acoplamento, apenas realizando comunicações síncronas e assíncronas.
Geralmente, os testes são da aplicação toda, e conforme cresce, fica cada vez mais difícil manter.	Facilidade em teste e monitoramento. Testes rodam muito mais rápido e com maior desempenho.





Monolítico vs Microsserviços





Referências e links utilizados

- RedHat – O que é arquitetura de microsserviços?
- Microsoft – Comunicação em uma arquitetura de microsserviço
- DZone - Patterns for Microservices — Sync vs. Async