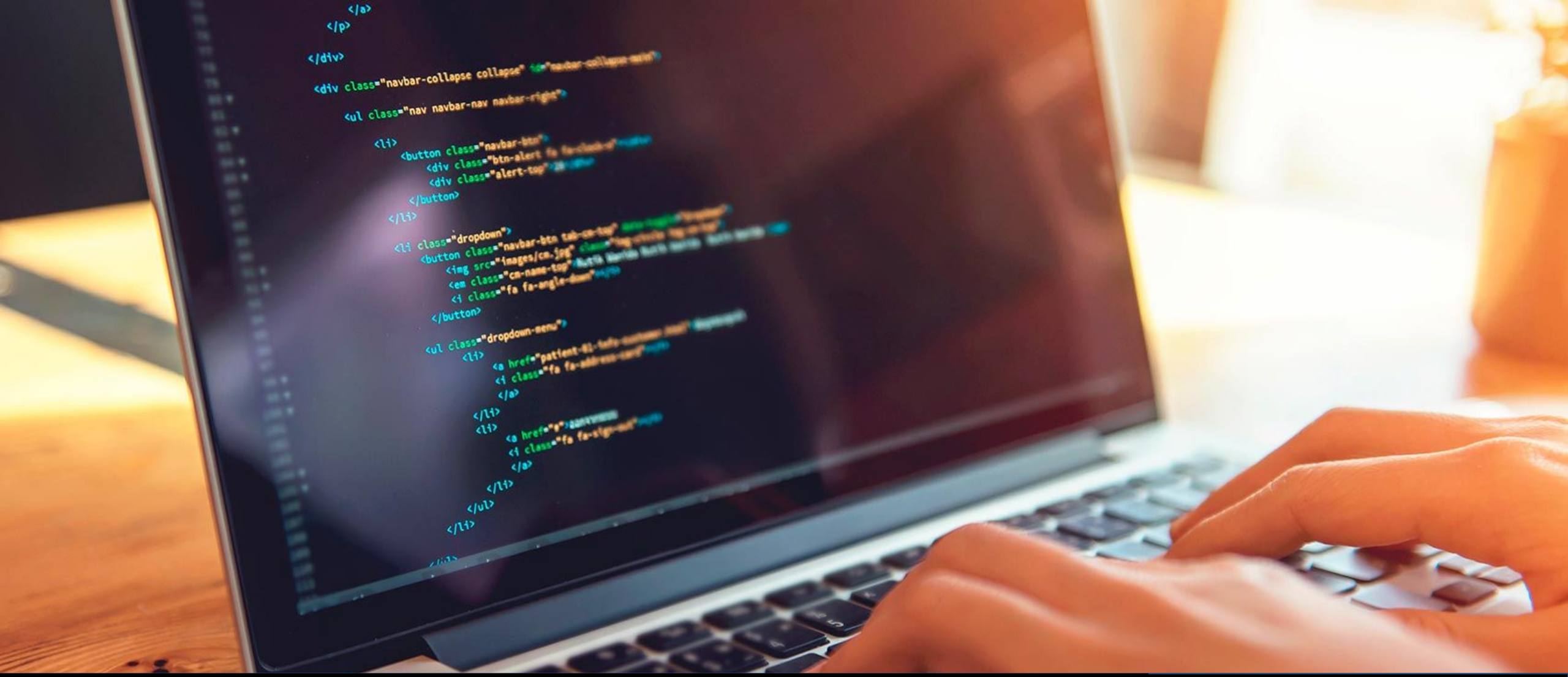
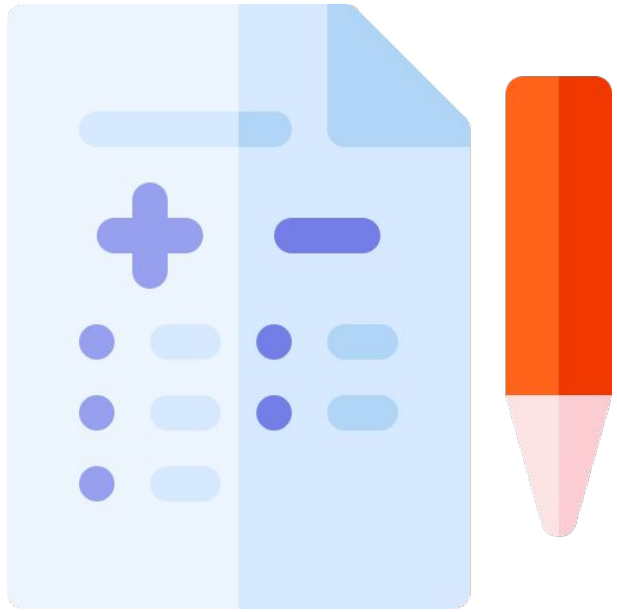


Introdução à Arquitetura de Microserviços

Jesiel Viana
@jesielviana
jesiel@ifpi.edu.br

Revisão

- Coesão
- Acoplamento
- Estilos arquiteturais
- Arquitetura em camadas
- Padrões de projeto
- Manutenibilidade



Roteiro

- Arquitetura monolítica
- Software como serviço (SAAS)
- Abordagens de implementação de serviços
- Arquitetura de microserviços

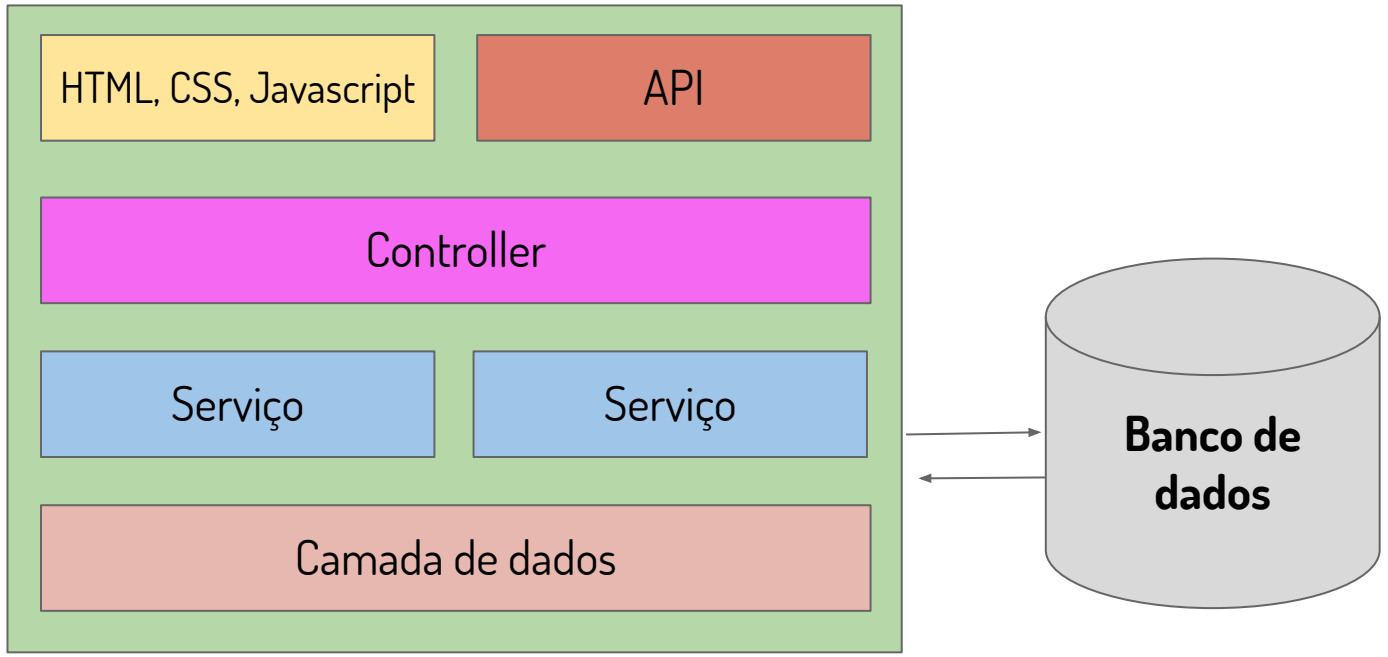
Arquitetura Monolítica

Arquitetura monolítica

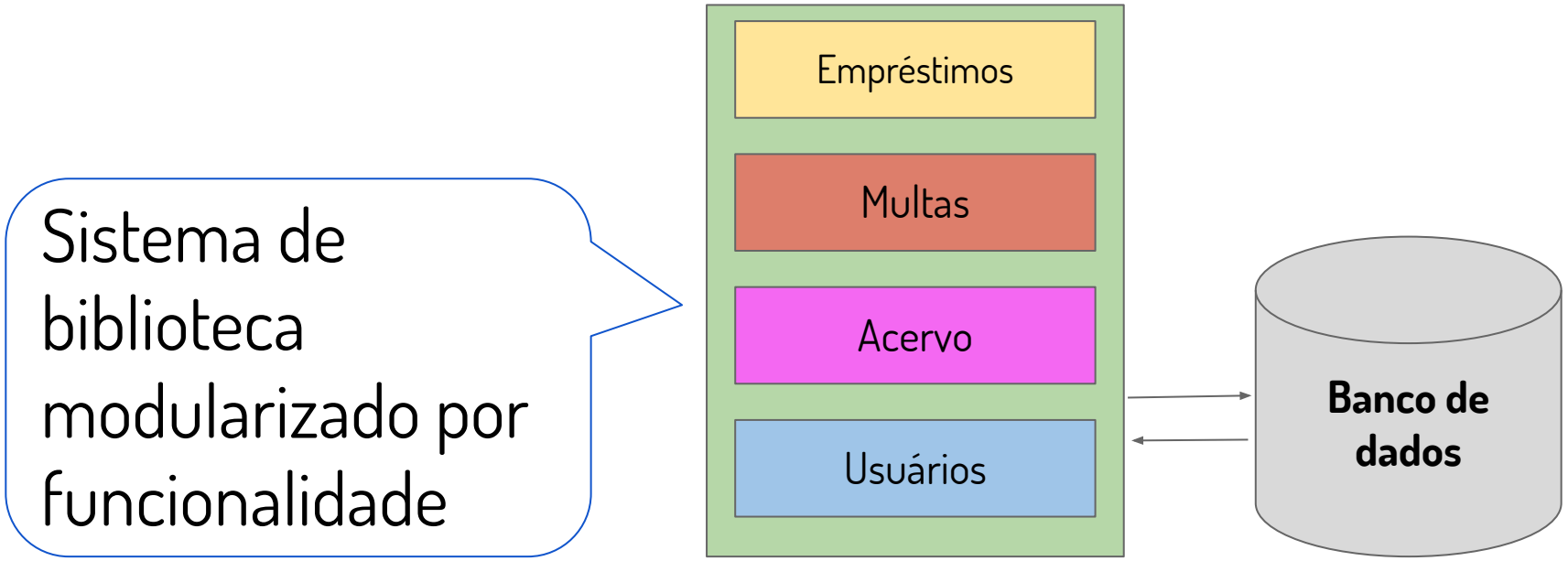
- Aplicações composta por um ou mais módulos que formam um único executável.
- A aplicação é executada em uma única máquina.
- Os módulos compartilham recursos de processamento, memória, bancos de dados e arquivos.
- A divisão da aplicação em módulos é importante para quebra de complexidade e reaproveitamento de código.

Arquitetura monolítica

Sistema de biblioteca modularizado por camadas



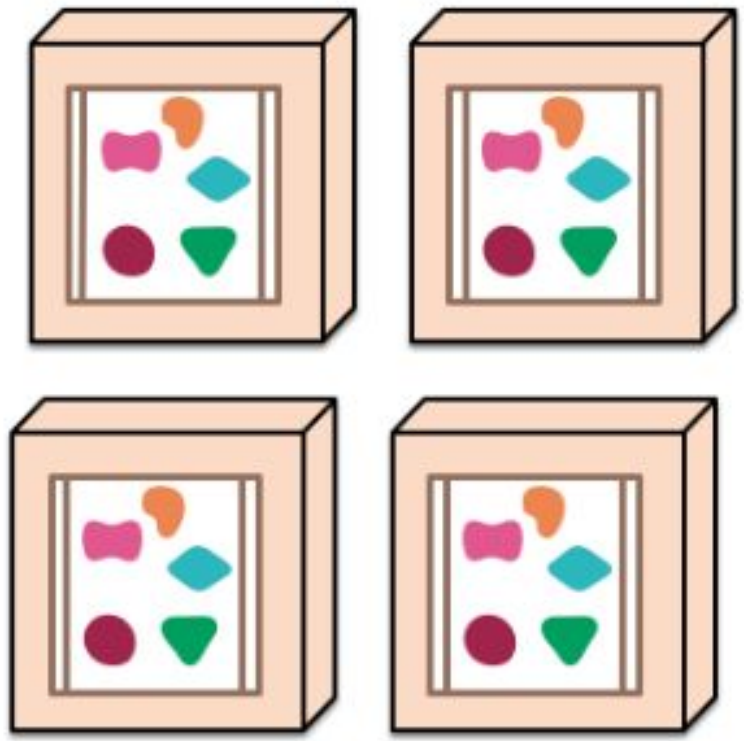
Arquitetura monolítica



Como escalar uma aplicação monolítica?

A monolithic application puts all its functionality into a single process...

... and scales by replicating the monolith on multiple servers



Fonte: [1]

Arquitetura monolítica

Vantagens

- Deploy mais simples
- Minimiza duplicidade de código

Desvantagens

- Ponto único de falha
- Base de código muito grande

Arquitetura monolítica

Com o crescimento do sistema sua complexidade vai aumentando, consumindo cada vez mais recursos.

Desafios para manutenção:

- Complexidade
- Dependência de componentes de código
- Escalabilidade do sistema é limitada
- Flexibilidade
- Integração
- Dificuldade para colocar alterações em produção

Surgimento da Arquitetura de Microsserviços...

Software como Serviço (SaaS)

Software como serviço

Software Tradicional: código binário instalado e rodando totalmente no dispositivo.

Software como Serviço (SaaS) é o fornecimento de software e dados como serviço através da rede, acessado via clientes 'leves' (ex. browser) sem a necessidade de instalar e configurar localmente um programa específico." (PATTERSON; FOX, 2012)

"SaaS foca em separar a posse e propriedade do software de seu uso." (TURNER et al., 2010)

Software como serviço

A Web é a principal plataforma habilitadora de Software como Serviço

Exemplos de SaaS

Exemplos:

BUREAU WORKS

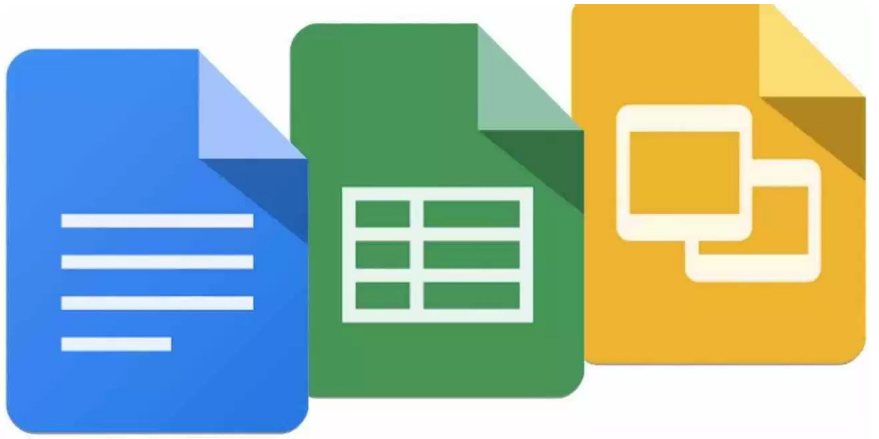
OpenWeatherMap

Tudo é API

Impactos pra quem usa SaaS

Mudança de Paradigma:

- Ferramentas de escritórios;
- Armazenamento;
- Vídeos e Músicas;
- Ambiente de desenvolvimento;



Razões para SaaS

Razões para SaaS

1. Sem preocupações com instalação e configuração de Software
2. Sem preocupações sobre perda de dados (backup no site remoto)
3. Fácil interação para grupos (dados compartilhados)
4. Compatibilidade com vários dispositivos
5. Simplificação das atualizações

Serviço

“Serviços são componentes de software reusáveis e de baixo acoplamento que encapsulam uma funcionalidade e podem estar distribuídos e serem acessados programaticamente. Web services, por exemplo, são serviços acessados através de protocolos baseados em XML e utilizando os padrões da internet.” (Sommerville)

Serviço by Jeff Bezos

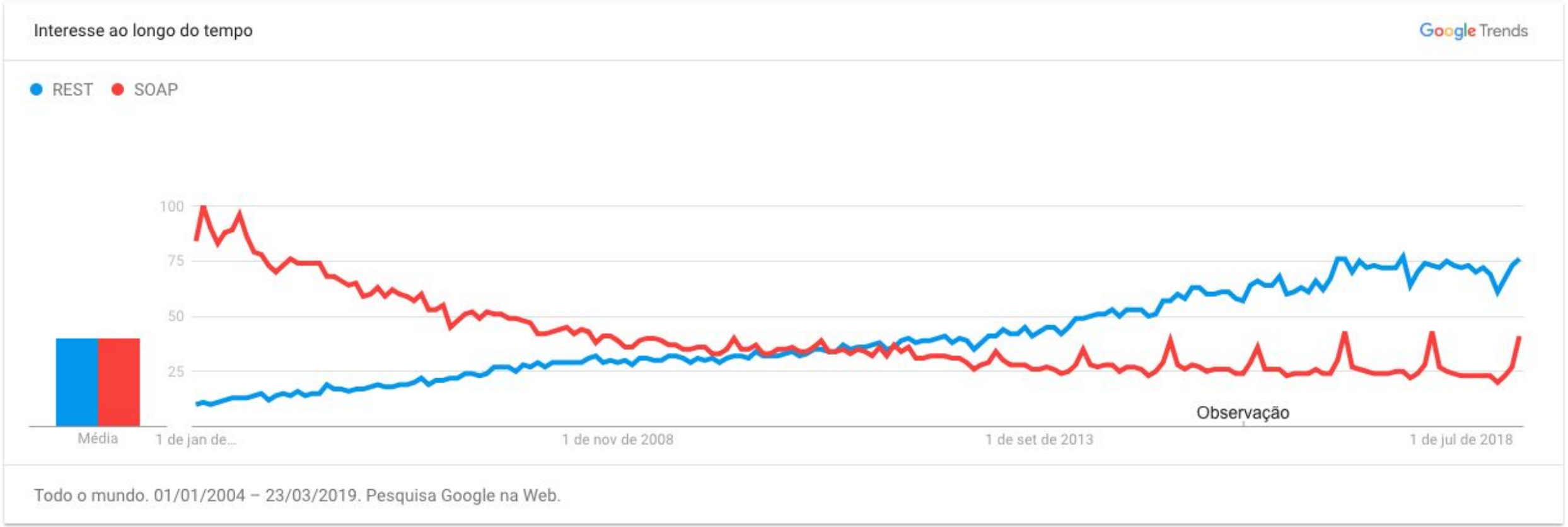
1. All teams will expose their data and functionality through service interfaces;
2. Teams **must** communicate with each other through these interfaces.
3. There will be no other form of interprocess communication allowed.
4. It doesn't matter what technology they use.
5. **All service interfaces,** without exception, must be designed from
6. the ground up to be externalizable. no exceptions.
7. Anyone who doesn't do this will be fired.
8. Thank You; have a nice day!

Abordagens de implementação de Serviços

Implementação de serviços

- SOAP Web Services
- REST Web Services

Implementação de serviços



Leitura e discussão

- SOAP vs. REST: A Look at Two Different API Styles - https://bit.ly/2DC6dzi

Arquitetura de Microserviços

Arquitetura de microserviços

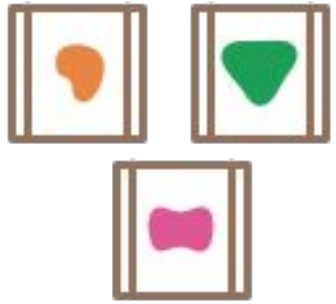
A arquitetura de microserviços é utilizada para desenvolver uma aplicação como um conjunto de pequenos serviços, cada um funcionando em seu próprio processo. Cada serviço é desenvolvido em torno de um conjunto de regras de negócio específico, e é implementado de forma independente.

Arquitetura de microserviços

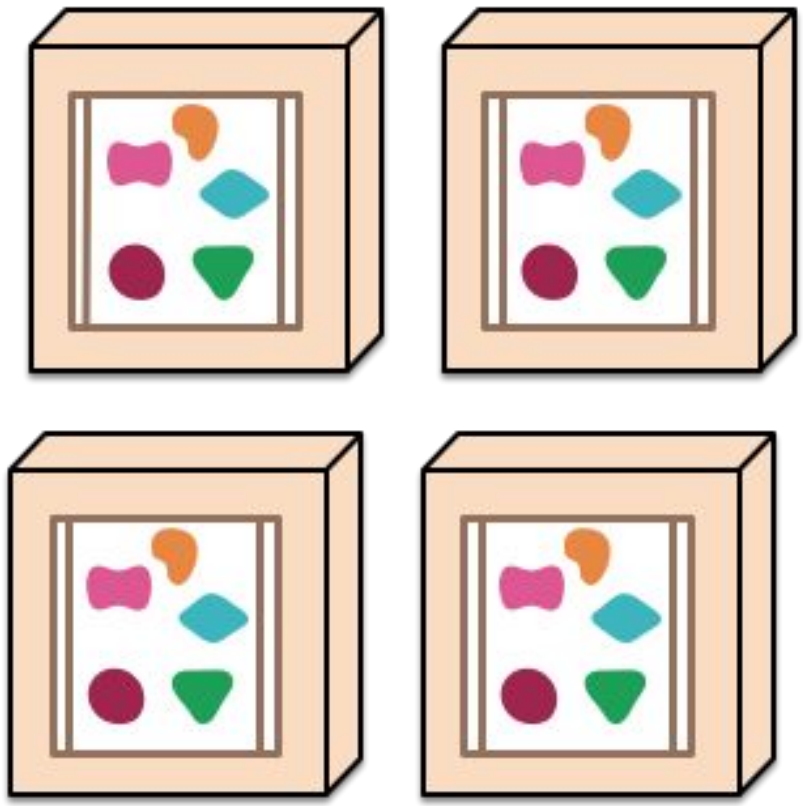
Microserviço é uma abordagem para desenvolver uma única aplicação como uma suíte de serviços, cada um rodando em seu próprio processo e se comunicando através de mecanismos leves, geralmente através de uma API HTTP. Estes serviços são construídos através de pequenas responsabilidades e publicados em produção de maneira independente através de processos de deploys automatizados. Existe um gerenciamento centralizado mínimo destes serviços, que podem serem escritos em diferentes linguagens e usarem diferentes tecnologias para armazenamento de dados.[1]

A monolithic application puts all its functionality into a single process...

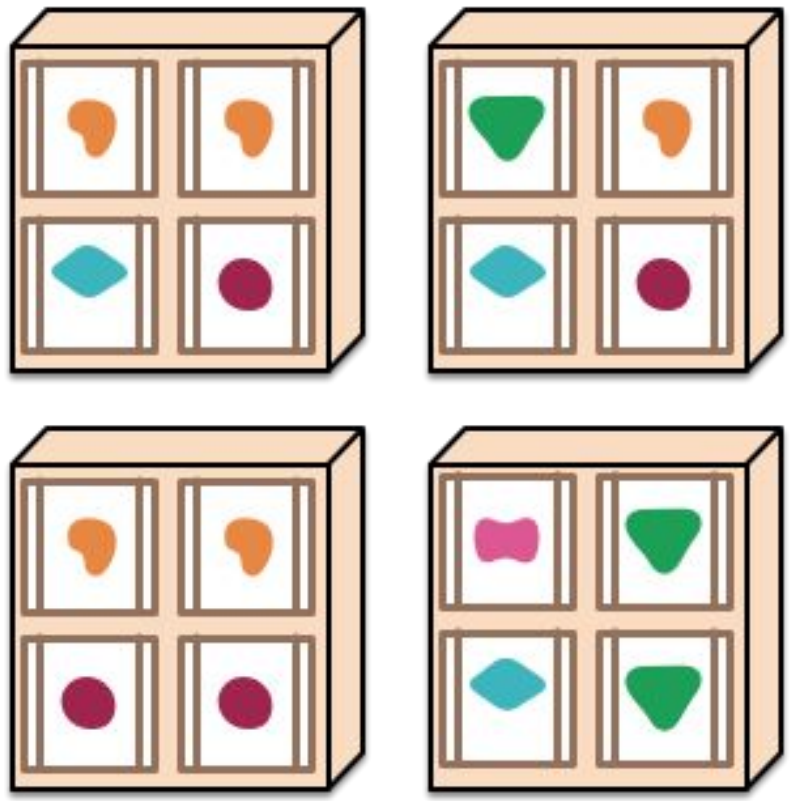
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



... and scales by distributing these services across servers, replicating as needed.



Fonte: [1]

Quando falamos de Arquitetura de Microserviços uma questão comum é compará-la com Arquitetura Orientada a Serviços (SOA).

SOA

- SOA é um estilo de arquitetura de software cujo princípio fundamental prega que as funcionalidades implementadas pelas aplicações devem ser disponibilizadas na forma de serviços. Frequentemente estes serviços são conectados através de um "barramento de serviços" (enterprise service bus, em inglês) que disponibiliza interfaces, ou contratos, acessíveis através de web services ou outra forma de comunicação entre aplicações [2] .
- "SOA é uma abordagem arquitetural corporativa que permite a criação de serviços de negócio interoperáveis que podem facilmente ser reutilizados e compartilhados entre aplicações e empresas." [2].

Conceitos

- **API - Application Programming Interface**: é um conjunto de instruções e padrões de programação para acesso a um aplicativo de software.
- **Biblioteca**: é uma implementação real das regras de uma API.
- **Framework**: é um conjunto de bibliotecas para conseguir executar uma operação maior.
- **Endpoint**: em termos simples, um endpoint de serviço web é um endereço da web (URL) no qual os clientes de um serviço específico podem ter acesso a serviço. Ao referenciar essa URL, os clientes podem obter as operações fornecidas pelo serviço.

Características de uma arquitetura em microserviços

Componentização via serviços

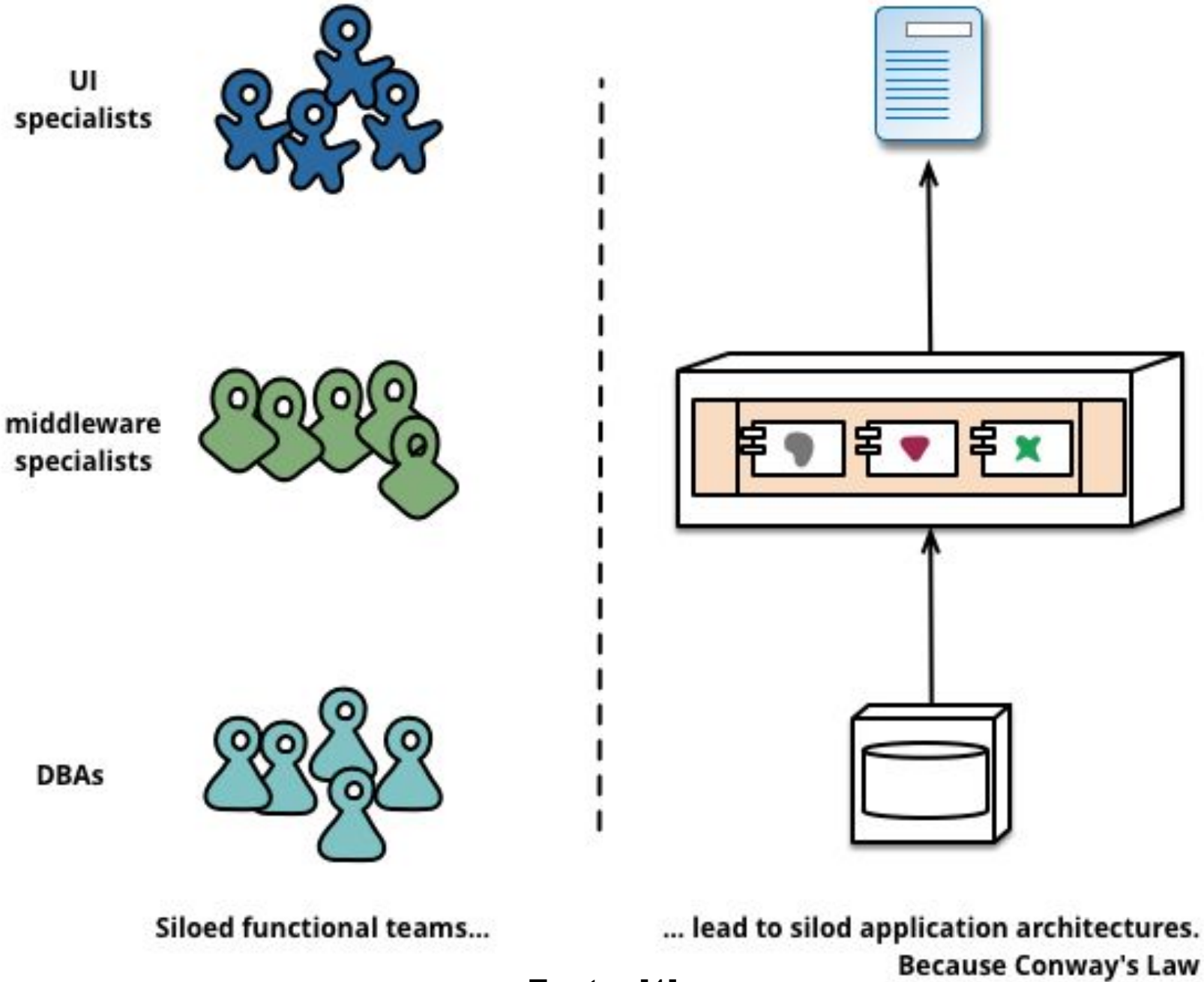
- Componente é uma unidade de software que é substituída ou atualizada de maneira independente.
- Arquiteturas em microsserviços usarão bibliotecas.
- Bibliotecas são componentes que são usados em um programa através de chamadas de função diretamente em memória, enquanto serviços são componentes em processos diferentes que se comunicam através de mecanismos tais como requisições via web services ou chamadas de código remotas.

Componentização via serviços

- ✓ Serviços são publicados de forma independentes, bibliotecas não.
- ✓ Bibliotecas aumentam o acoplamento.
- ✓ Chamadas remotas a serviços são mais "caras".
- ✓ Serviços precisam ser granulares, exige mais maturidade do desenvolvedor.

Organização de Times

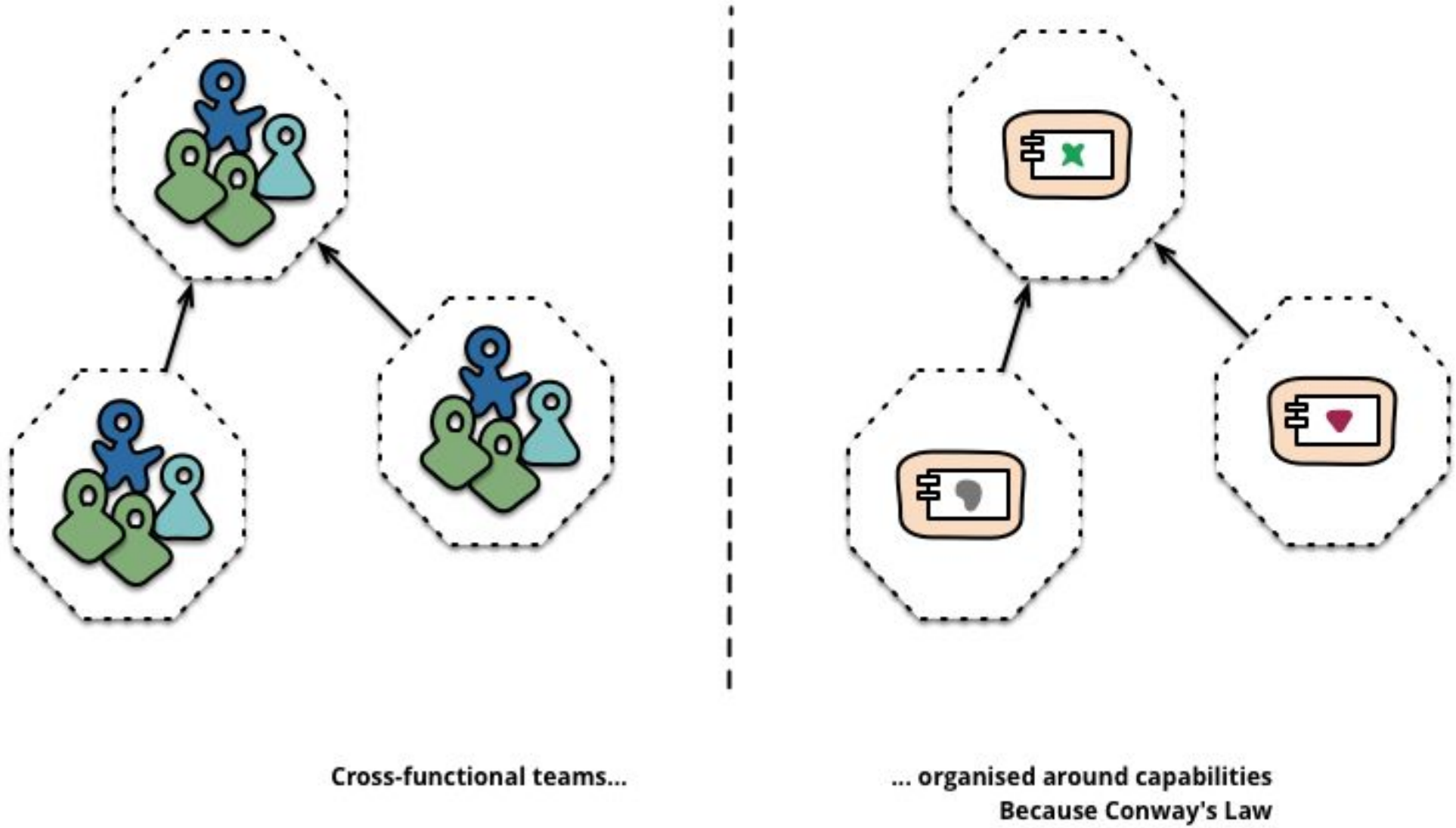
- Geralmente quando desenvolvemos grandes aplicações fazemos divisões por camadas, levando as equipes a serem divididos por especialidades como: front-end, back-end, bd, etc.


UI
specialists
middleware
specialists
DBAs
Siloed functional teams...
... lead to silod application architectures.
Because Conway's Law

Fonte: [1]

Organizado por das áreas do negócio

- A abordagem proposta pelos microserviços:
  - ✓ organizar os times ao redor das áreas do negócio.
  - ✓ Times multifuncionais.


Cross-functional teams...
... organised around capabilities
Because Conway's Law

Fonte: [1]

Self-Contained Systems

A abordagem do Sistema Autônomo (SCS) é uma arquitetura que se concentra na separação da funcionalidade em muitos sistemas independentes, em que a colaboração de muitos sistemas de software menores forma um sistema lógico completo. Isso evita o problema de grandes monólitos que crescem constantemente e acabam se tornando inatingíveis.

A ideia é dividir um grande sistema em vários sistemas independentes menores, ou SCSs, que seguem certas regras.

Veja mais em: https://scs-architecture.org

Endpoints inteligentes

- ✓ Endpoints inteligentes e fluxos de comunicação simples.
- ✓ Aplicações construídas a partir de microsserviços devem ser tão desacopladas e coesas quanto possível
- ✓ recebe uma requisição, aplica a lógica apropriada e produz uma resposta.
- ✓ As formas mais usadas são:
  - Protocolo HTTP (REST)
  - Barramento de mensagens simples (RabbitMQ)

Governança descentralizada

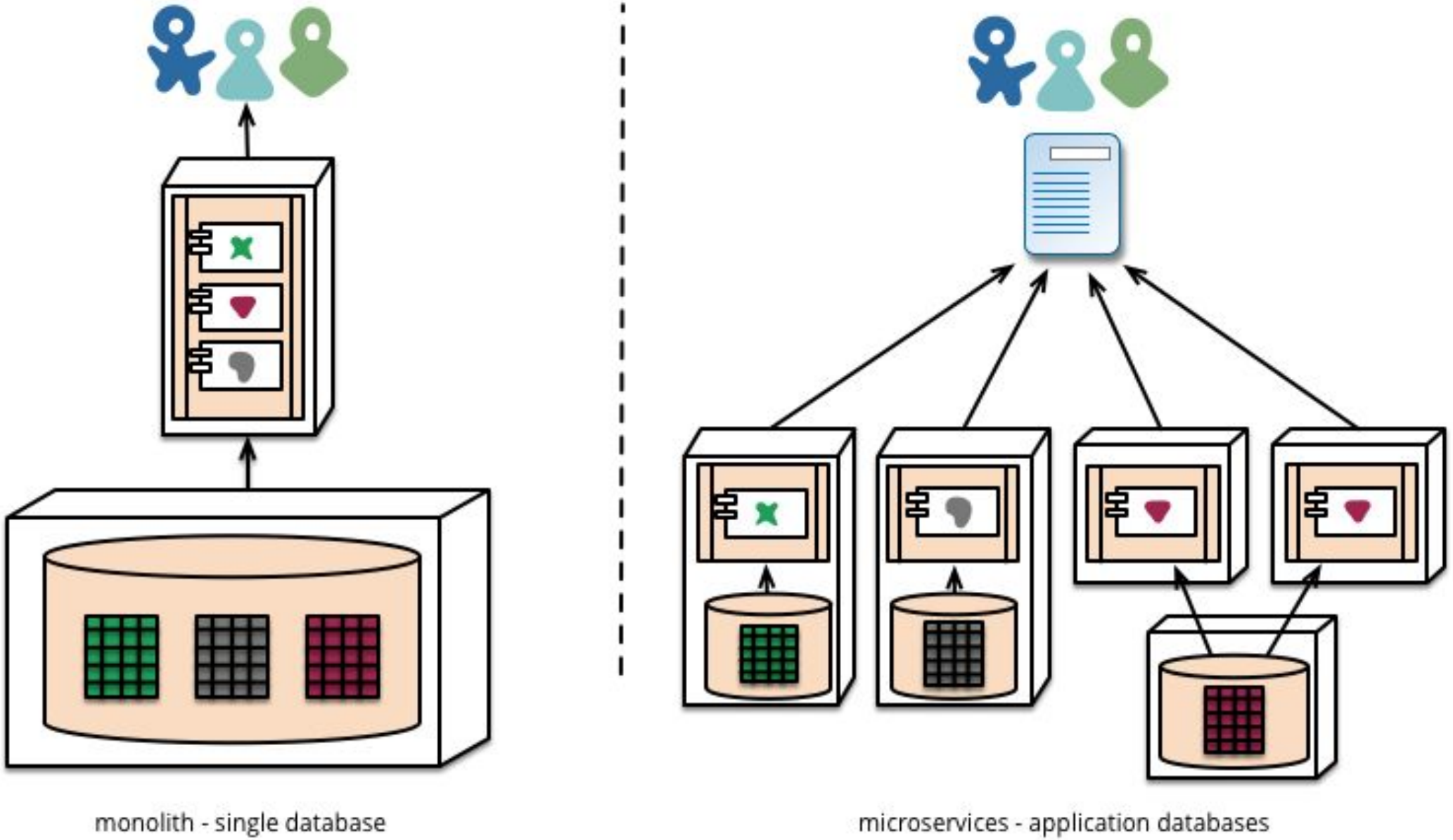
- Uma das conseqüências de governanças centralizadas é a tendência de padronizar tudo em uma única plataforma tecnológica.
- Ao quebrar componentes monolíticos em serviços temos a escolha de como construir cada um deles de maneira diferente, com tecnologias (linguagem, banco de dados) diferentes.
- Estratégias de deploy e recursos computacionais diferentes.

Administração descentralizada de dados

- Resolve o problema em que alguns setores de uma organização tem visões diferentes sobre determinadas entidades
- A visão que o setor de vendas tem de um cliente pode ser diferente do ponto de vista do suporte. Objetos podem ter diferentes atributos e (pior) atributos em comum, com diferentes significados.
- Enquanto aplicações monolíticas preferem uma única base de dados lógica para a persistência de dados. Microsserviços preferem permitir que cada serviço gerencie sua própria base de dados, quer através de diferentes instâncias usando a mesma tecnologia de banco de dados, ou até mesmo usando diferentes sistemas de banco de dados.

Administração descentralizada de dados

- A responsabilidade descentralizada para os dados através dos microsserviços tem implicações para a administração de atualizações. A abordagem comum para lidar com atualizações tem sido usar transações para garantir a atualização de múltiplos recursos.
- O uso de transações como estas ajuda com a consistência, mas impõem um acoplamento temporário significante,
- A forma de gerenciar estas inconsistências é um novo desa o para muitos times de desenvolvimento.
- Geralmente as empresas lidam com um certo grau de inconsistência com o objetivo de responder rapidamente a sua demanda, possuindo algum processo para lidar com erros. Esta troca é vantajosa enquanto o custo de corrigir os erros for menor que o custo da perda gerada por ter uma consistência maior.

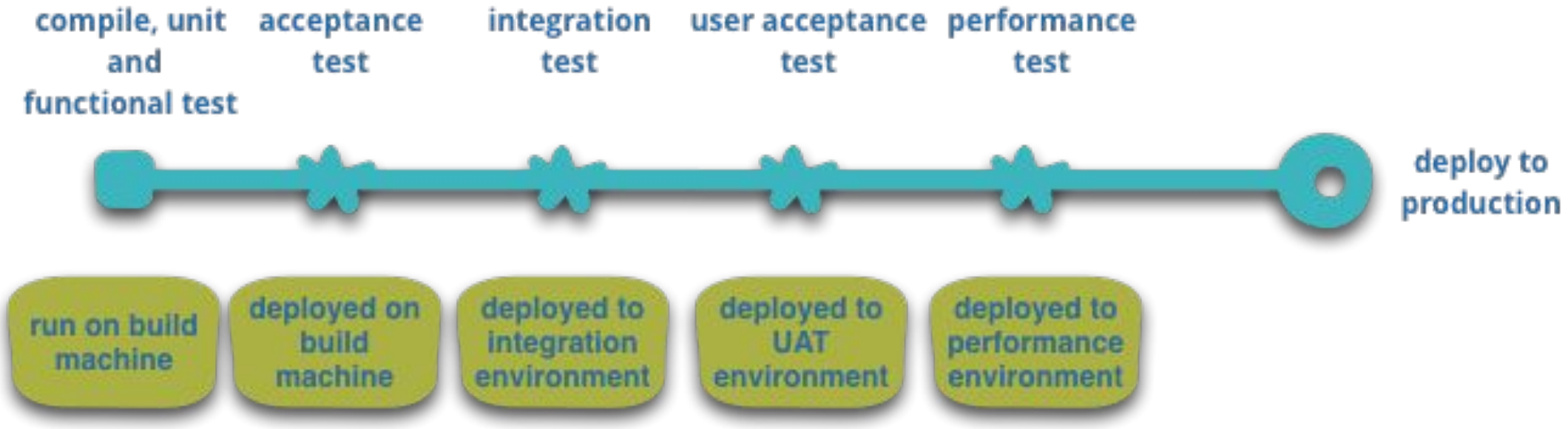

monolith - single database
microservices - application databases

Fonte: [1]

Automação da Infraestrutura

Testes automatizados

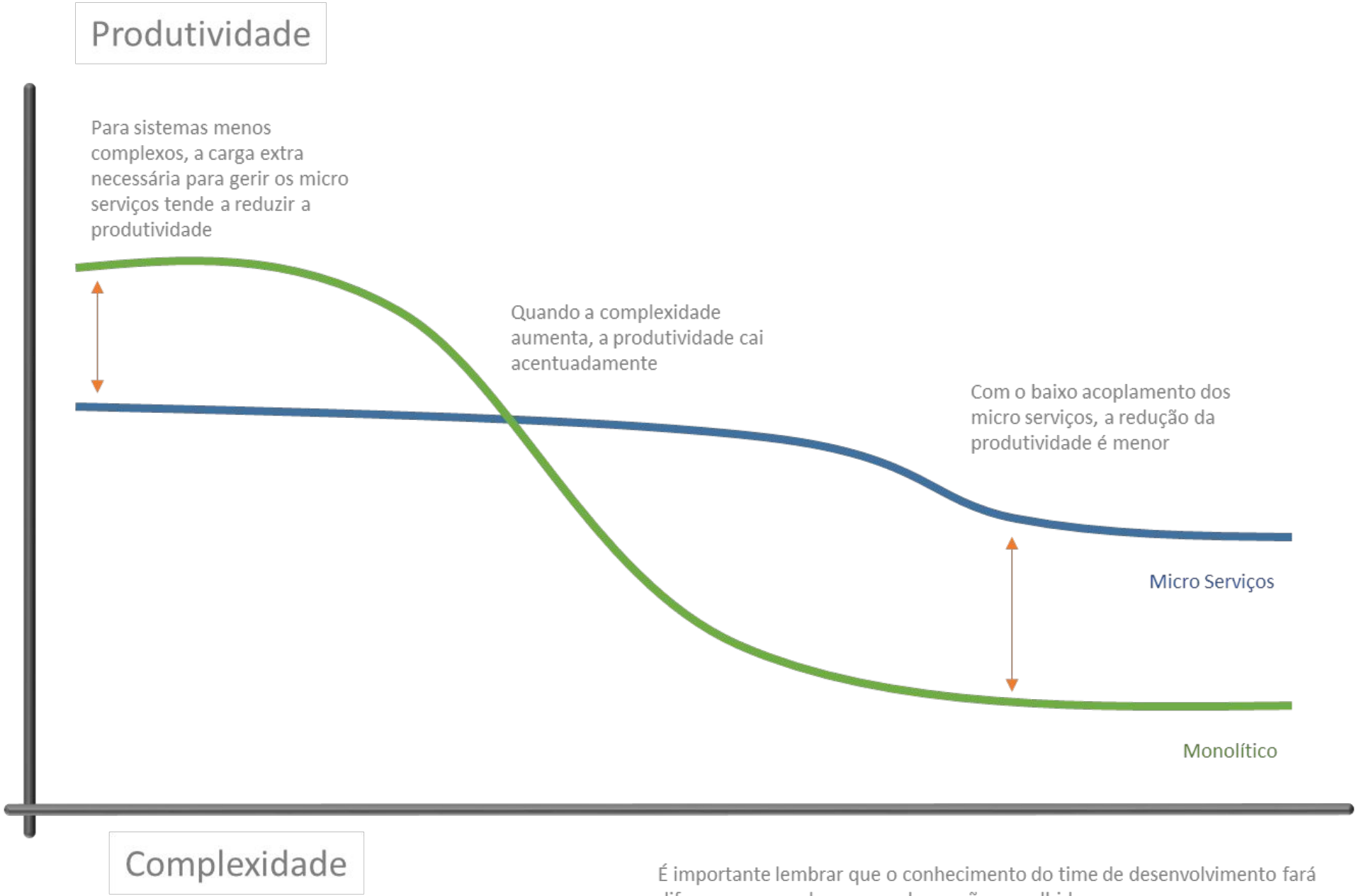
Integração contínua



Projetado para a falha

- Uma consequência do uso de serviços como componentes é que as aplicações precisam ser desenhadas de maneira que possam tolerar a falha dos serviços.
- Qualquer chamada de serviço poderá falhar devido a uma indisponibilidade do mesmo, e o cliente precisa responder a isso da maneira mais tranquila possível.
- Uma vez que os serviços podem falhar a qualquer momento, é importante ser capaz de detectar falhas rapidamente e, se possível, restaurar o serviço automaticamente.
  - ✓ Teste do ambiente de produção de maneira automática.
  - ✓ Monitoramento em tempo real.

Na análise de Martin Fowler, um sistema complexo que utiliza micro serviços tem um custo de manutenção menor do que o de aplicações com arquitetura monolítica, como exemplificado no gráfico a seguir:



Fonte: [3]

É importante lembrar que o conhecimento do time de desenvolvimento fará diferença em qualquer uma das opções escolhidas

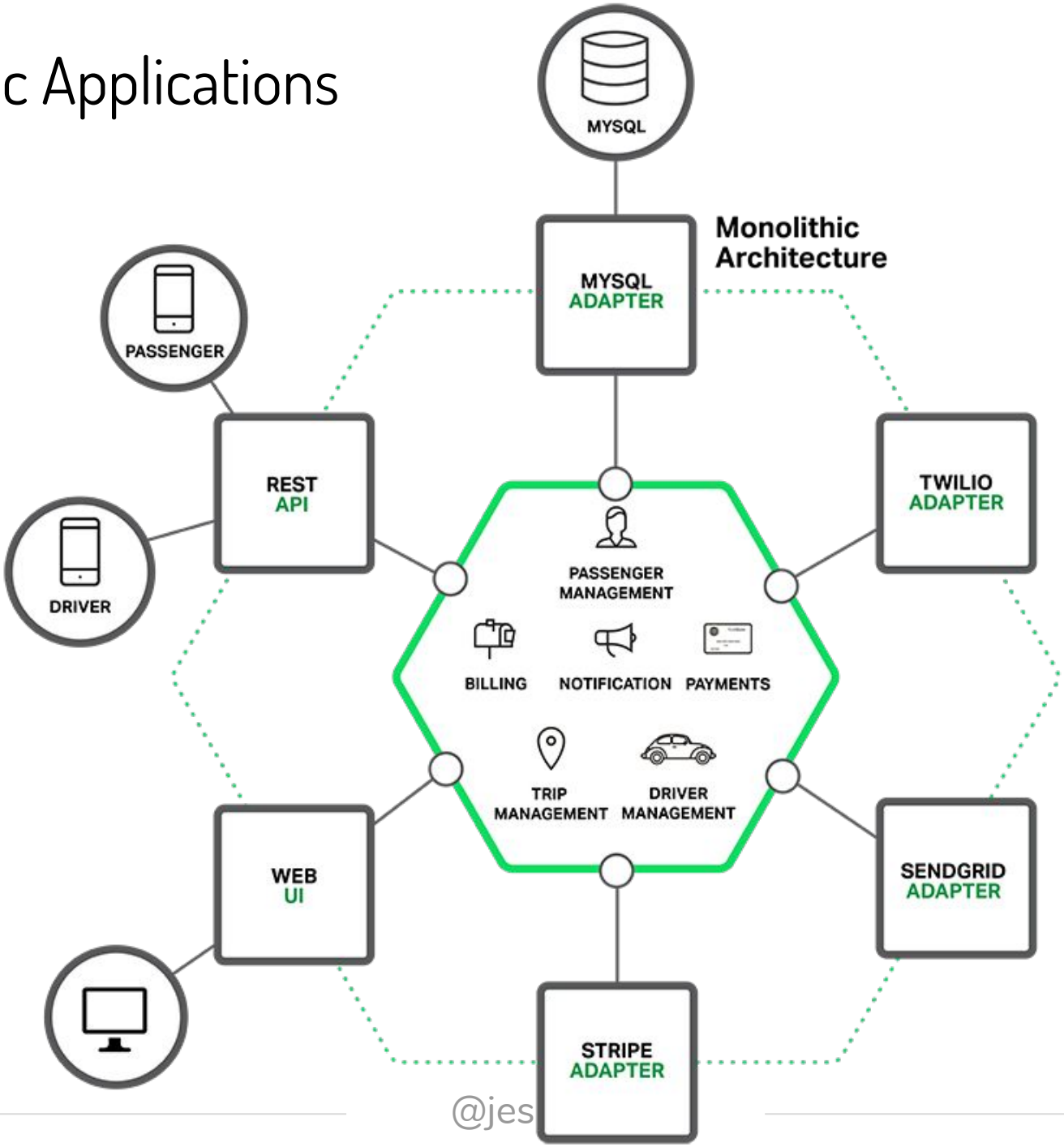
Microserviços será o futuro?

Quem usa Microserviços?

- Uber
- Netflix
- Amazon
- Ebay
- Sound Cloud
- Serviço Digital Governamental do Reino Unido,

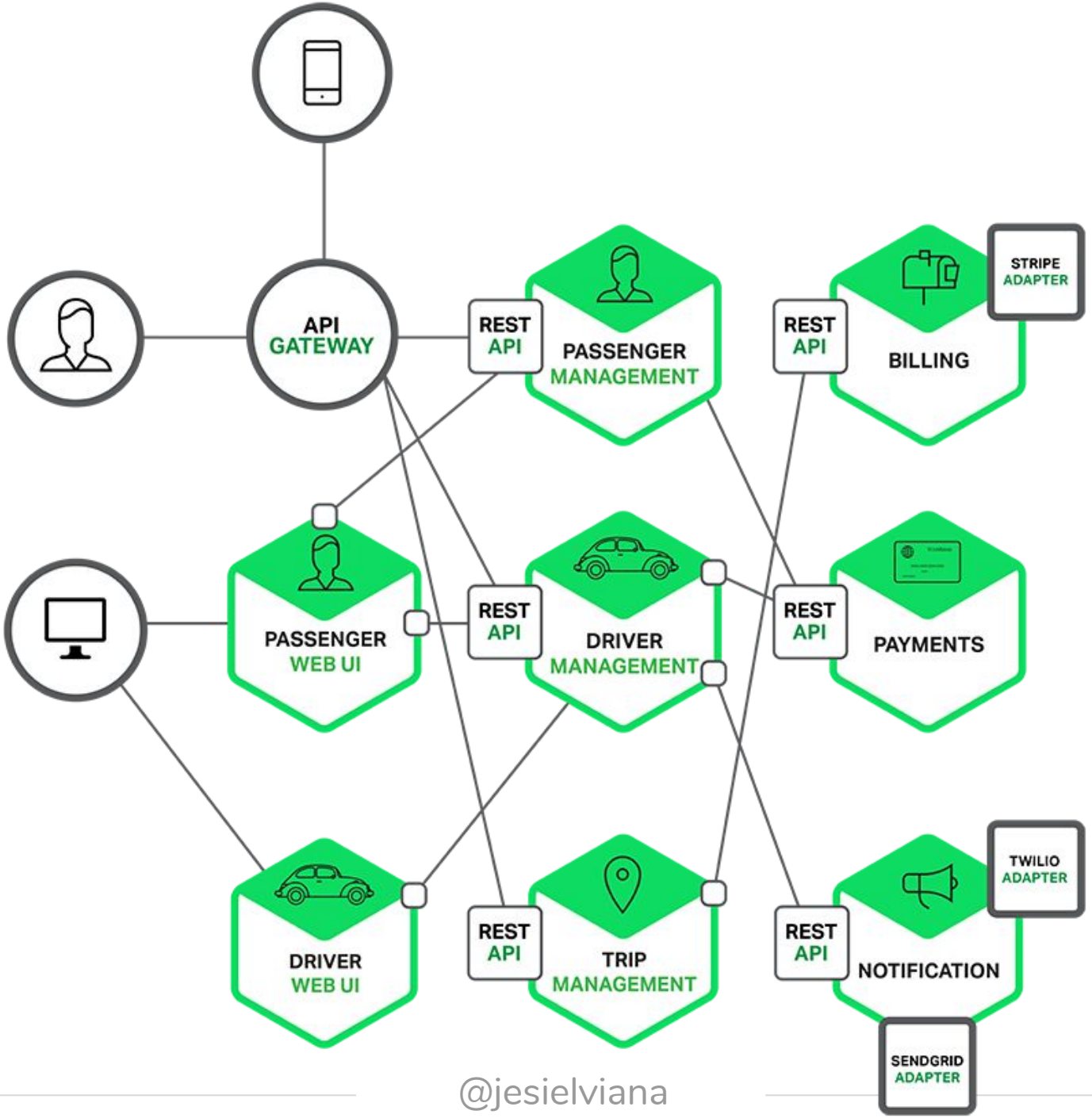
Apêndice A - Arquitetura Monolítica e Microserviços

Building Monolithic Applications



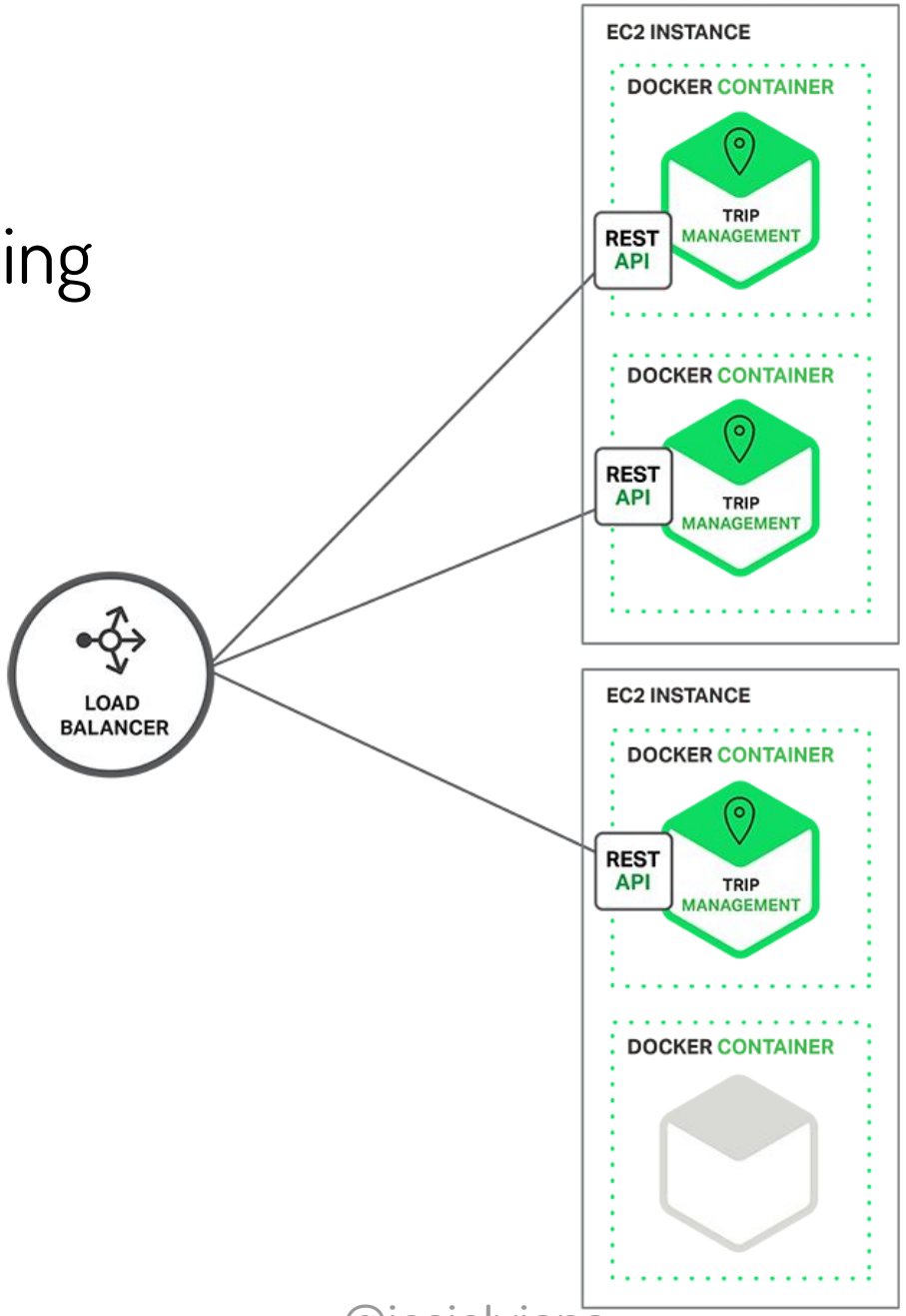
Fonte: [5]

Microservices



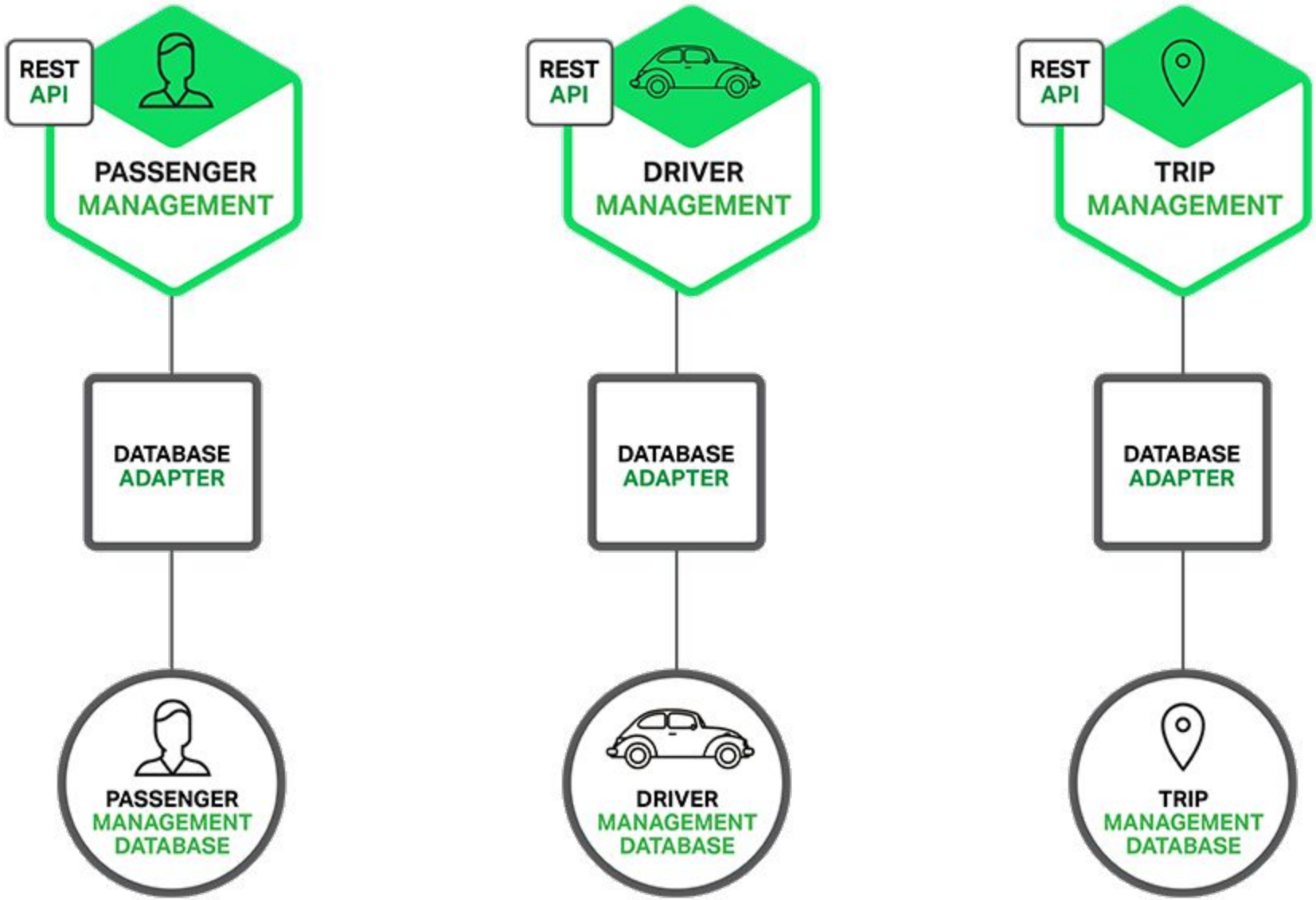
Fonte: [5]

Microservices scaling



Fonte: [5]

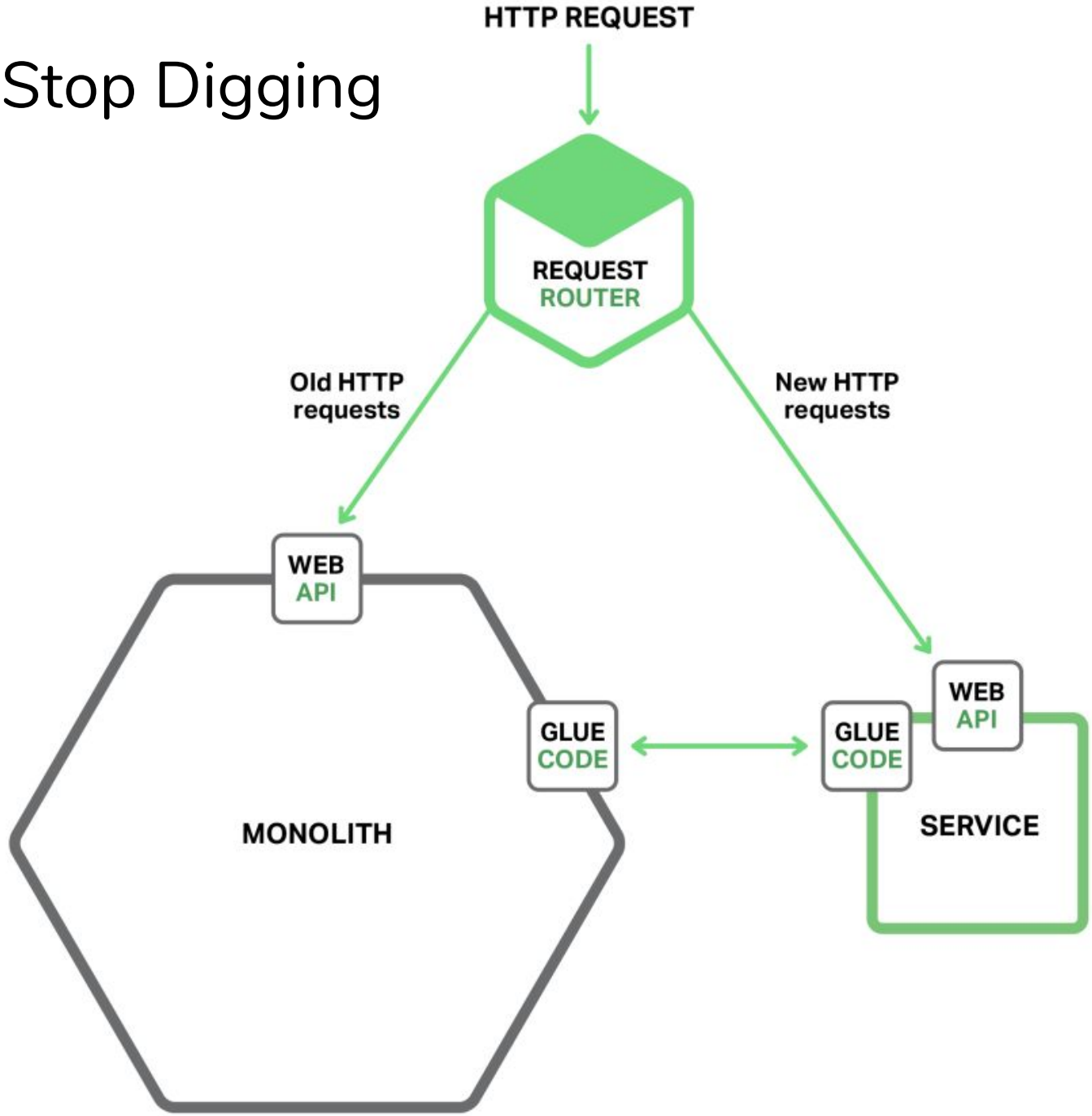
Microservices Databases



Fonte: [5]

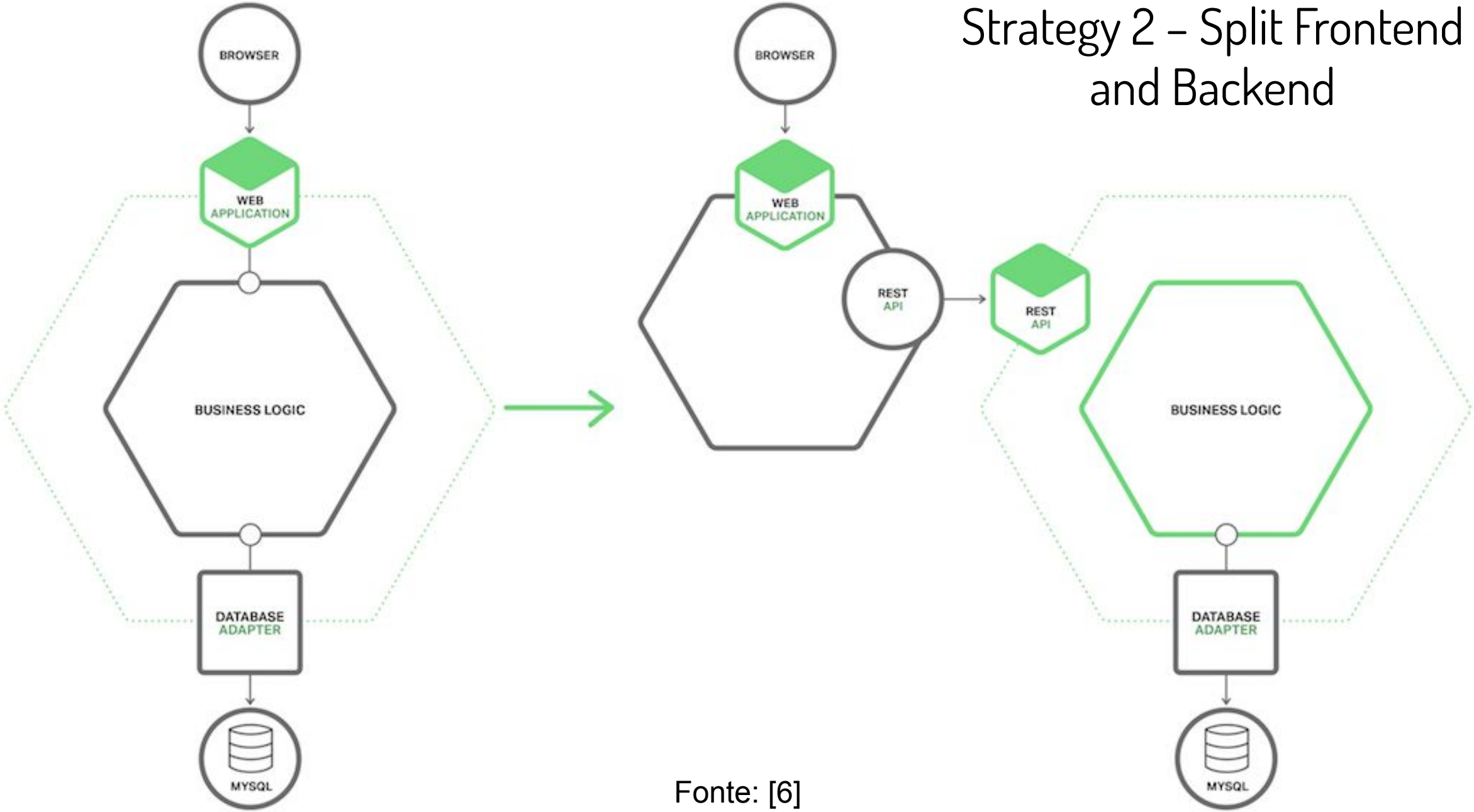
Apêndice B - Refactoring: Arquitetura Monolítica em Microserviços

Strategy 1 – Stop Digging



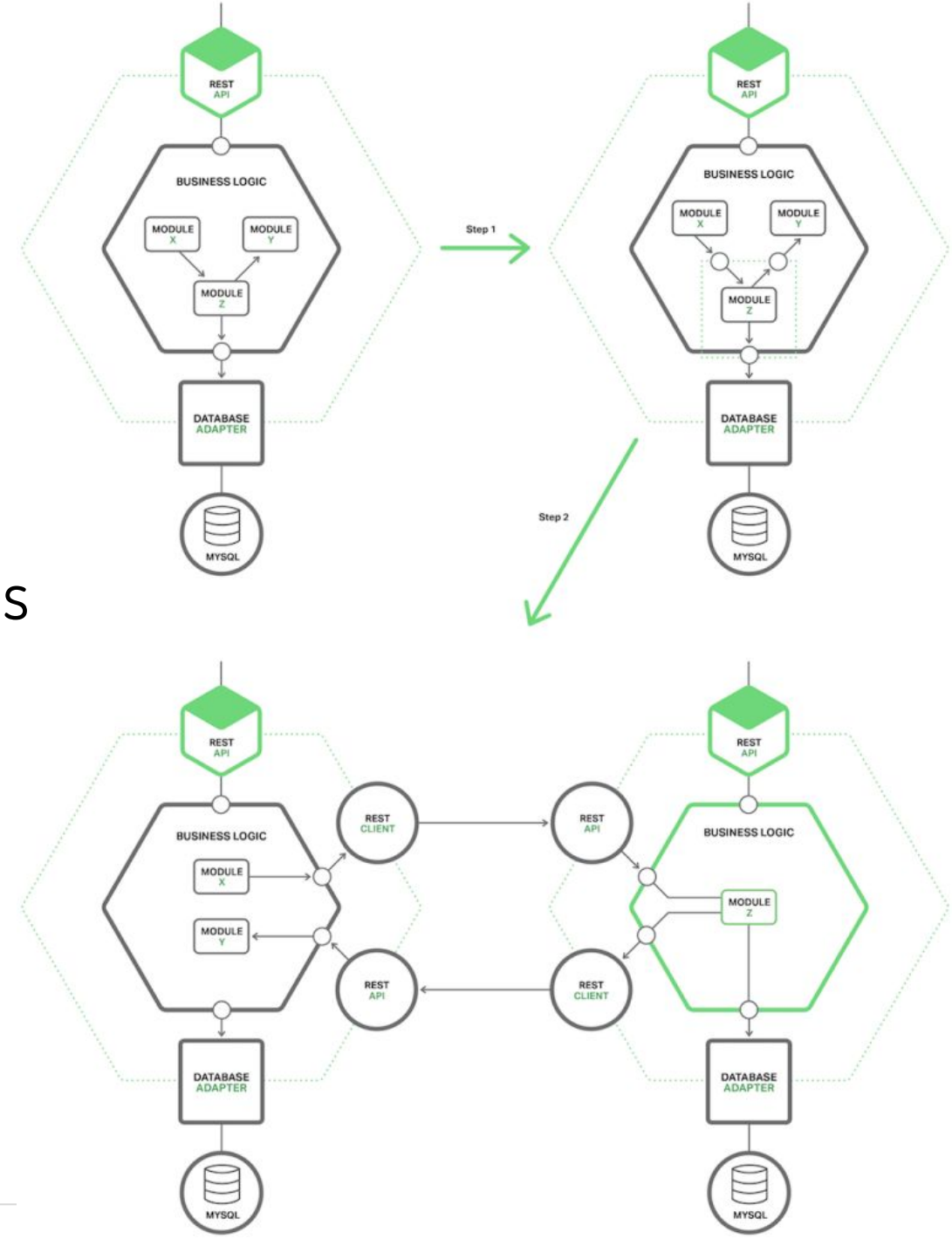
Fonte: [6]

Strategy 2 – Split Frontend and Backend



Fonte: [6]

Strategy 3 – Extract Services



Fonte: [6]

Bibliografia

- [1] https://martinfowler.com/articles/microservices.html
- [2]https://pt.wikipedia.org/wiki/Service-oriented_architecture
- [3]https://martinfowler.com/bliki/MicroservicePremium.html
- [4]http://redmonk.com/sograd y/2017/07/20/soa-microservices/
- [5]https://www.nginx.com/blog/introduction-to-microservices/
- [6]https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/
- http://www.pedromendes.com.br/2016/01/02/microservicos/
- http://blog.caelum.com.br/arquitetura-de-microservicos-ou-monolitica/
- https://www.opus-software.com.br/microservicos-diferenca-arquietura-monoliticas/
- https://www.nginx.com/resources/library/designing-deploying-microservices/
- NEWMAN, Sam. Building microservices: designing fine-grained systems. " O'Reilly Media, Inc.", 2015.

Obrigado!!!

jesiel@ifpi.edu.br @jesielviana @prof-jesielviana