

MBA
USP
ESALQ

MICROSERVIÇOS

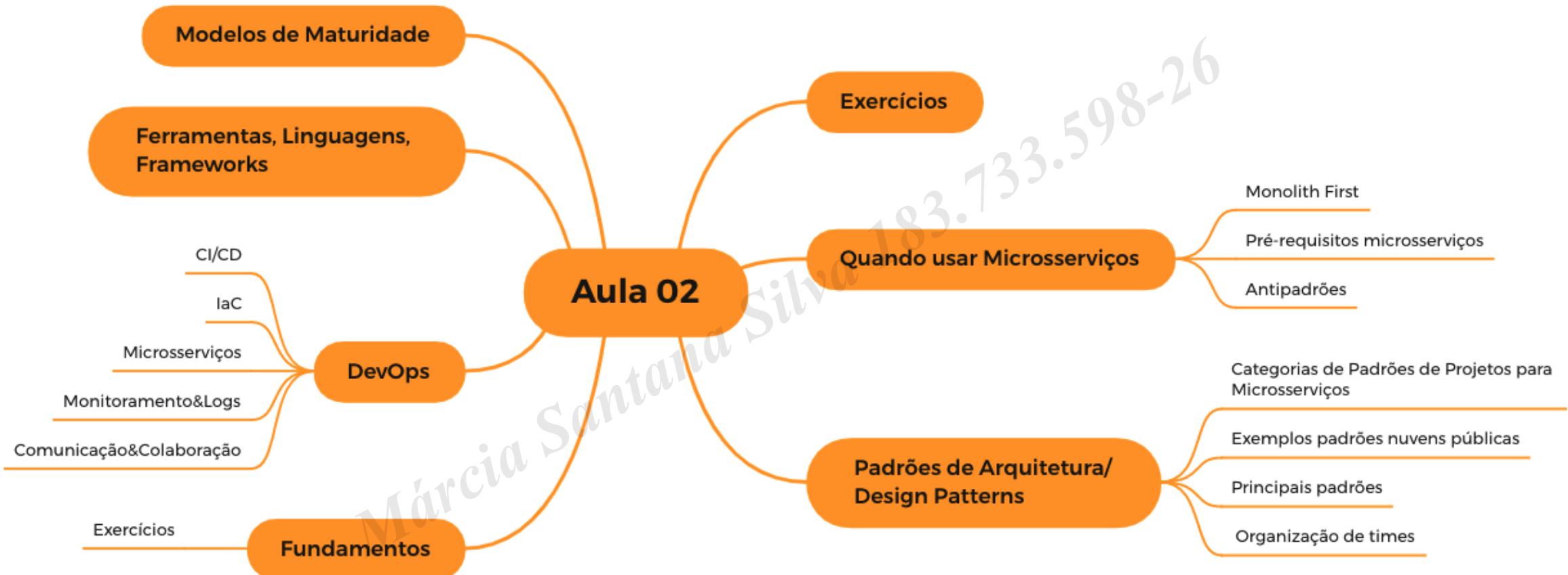
Prof. José Maria Cesário Jr.

*A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98

AGENDA MICROSERVIÇOS

Data	Conteúdo
08/02/2024	Fundamentos Quando usar microserviços Padrões de arquitetura/Design Patterns Métodos Ágeis/Devops Ferramentas, Linguagens, Frameworks Modelos de Maturidade Exercícios



ÍCONES

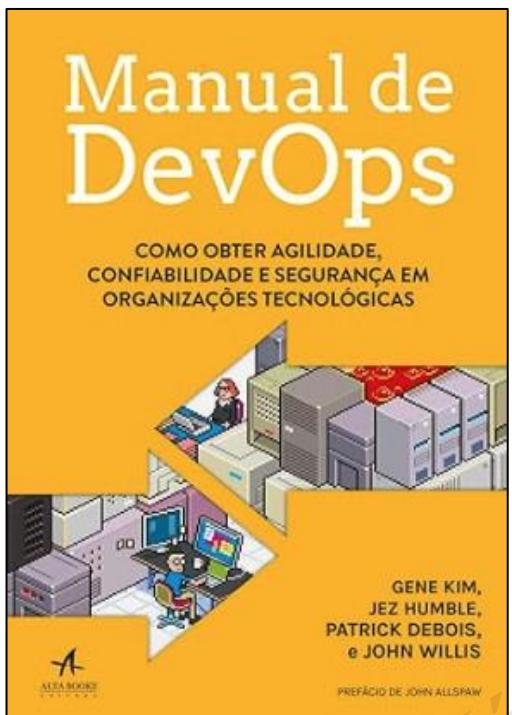


**DEFINIÇÃO OU
PONTO DE ATENÇÃO**

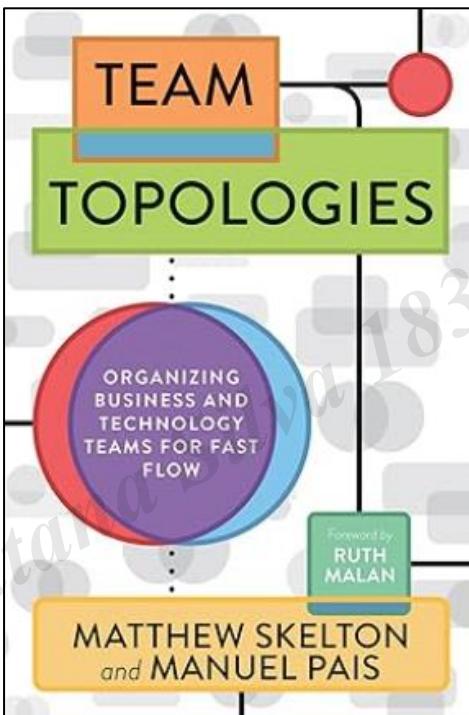


**EXERCÍCIO OU
ATIVIDADE PRÁTICA**

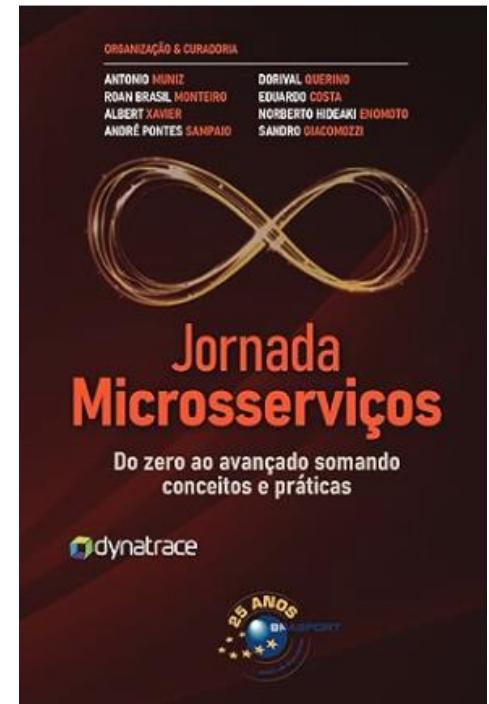
LIVROS



Manual de DevOps
Vários Autores
1ª Edição 2018



Team Topologies
Matthew Skelton
Manuel Pais
1ª Edição 2019



Jornada Microsserviços
Vários Autores
1ª Edição 2022

FUNDAMENTOS

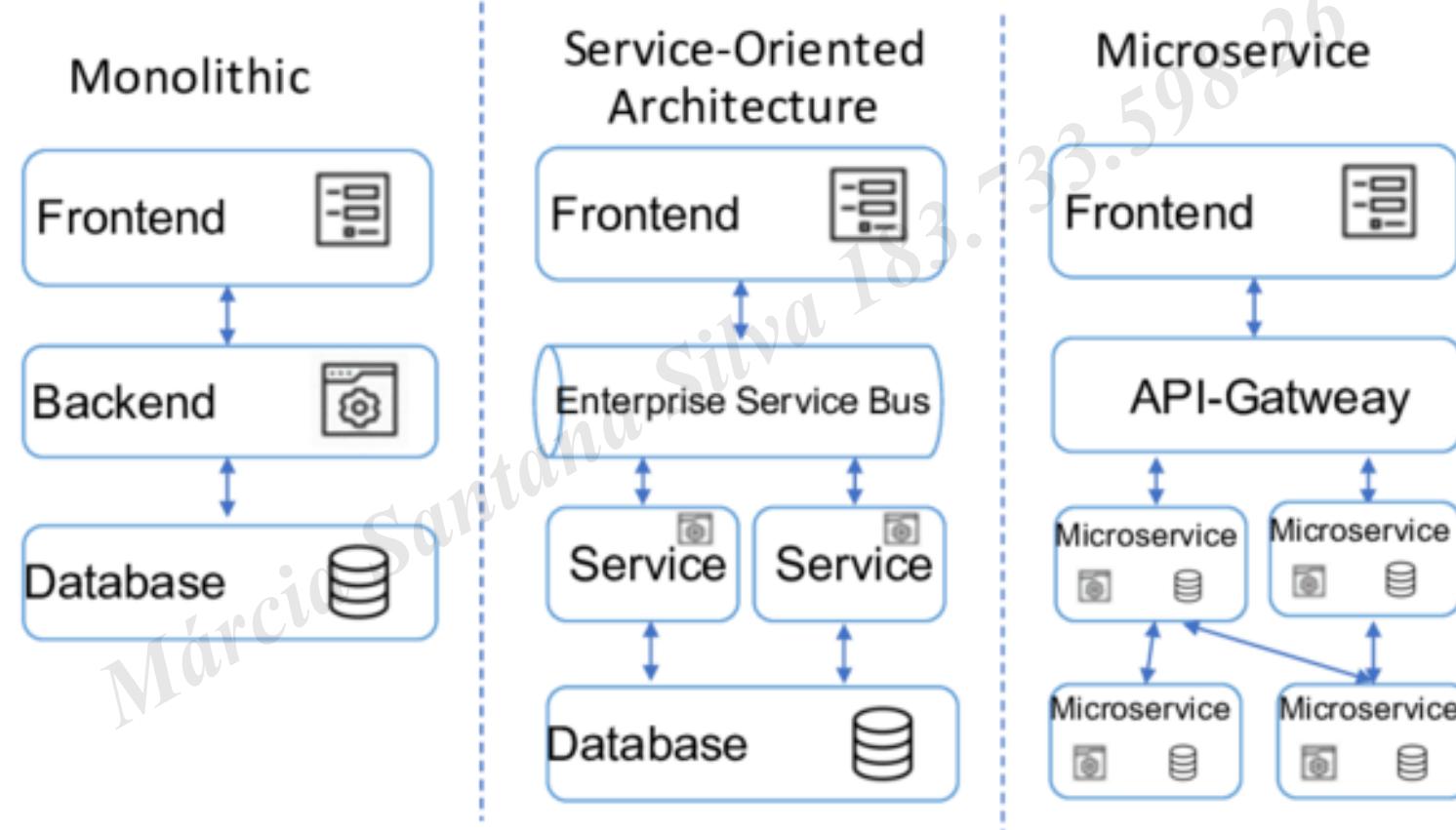
Parte 1

O que sabemos até agora

Microsserviços	Linguagens	REST API
<ul style="list-style-type: none">• O padrão de desenvolvimento de software de microsserviços fornece um método para criar um serviço unificado a partir de uma coleção de serviços menores.• Cada componente de microsserviço se concentra em um conjunto de ações fracamente acopladas em um conjunto de dados pequeno e bem definido.• Cada microsserviço inclui um sistema de armazenamento independente para que todos os seus dados estejam localizados em um único local.	<ul style="list-style-type: none">• Os microsserviços podem ser escritos em qualquer linguagem de programação e podem usar qualquer ambiente de banco de dados, hardware e software que faça mais sentido para a organização.• Uma interface de programação de aplicativos (API) fornece os únicos meios para os usuários e outros serviços acessarem os dados do microsserviço.	<ul style="list-style-type: none">• A API não precisa estar em nenhum formato específico, mas a transferência de estado representacional (REST) é popular, em parte porque sua legibilidade humana e natureza sem estado a tornam útil para interfaces da Web.• Outros formatos de API comuns incluem gRPC e GraphQL, cada um com abordagens e compensações diferentes para como os dados são acessados.

Fonte: Carnegie Mellon University, Software Engineering Institute → <https://insights.sei.cmu.edu/blog/8-steps-for-migrating-existing-applications-to-microservices/>

O que sabemos até agora

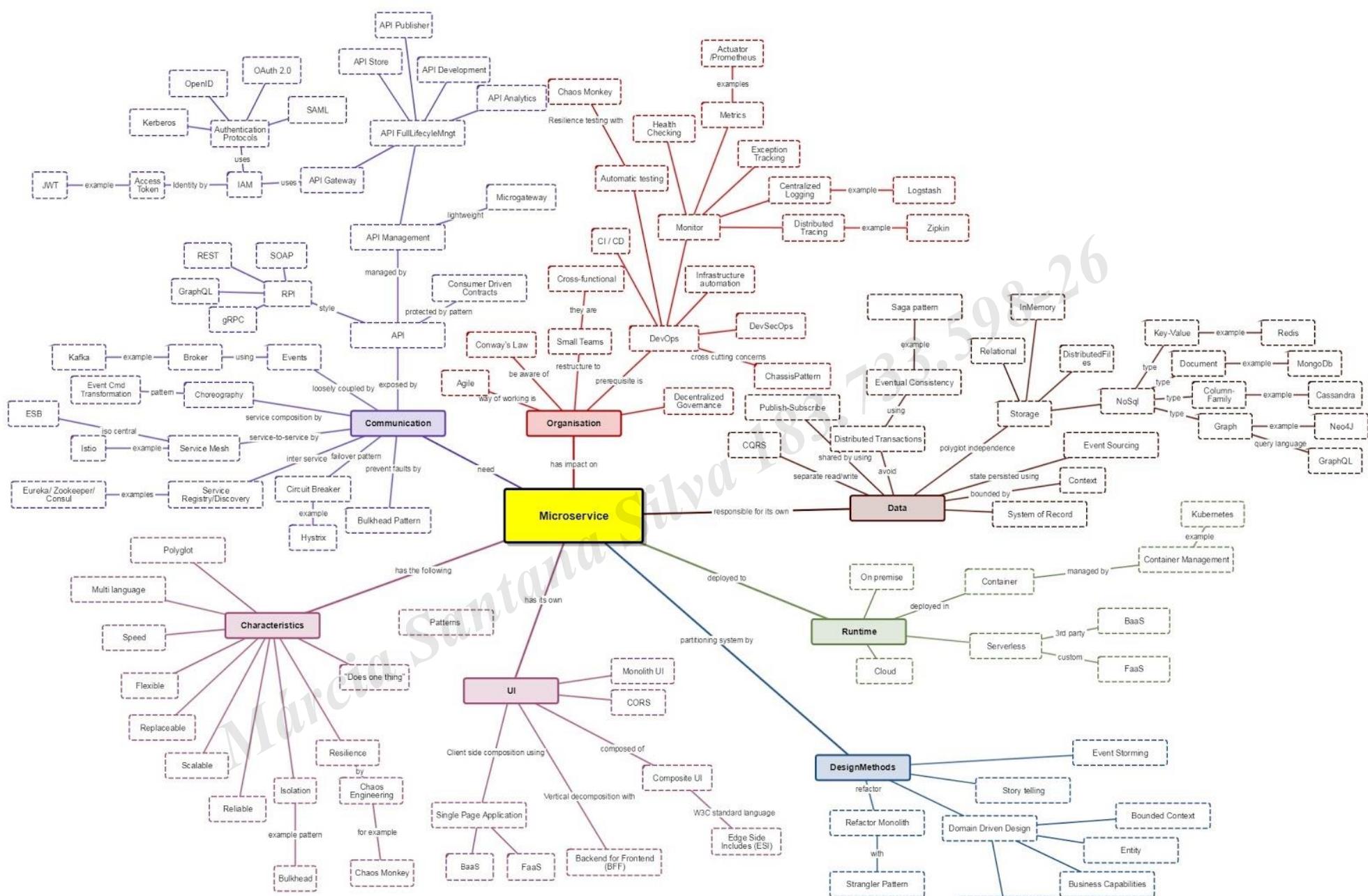


Fonte: https://www.researchgate.net/figure/The-Evolution-of-Software-Architectures_fig1_346675531

O que sabemos até agora

- Os princípios usados para projetar Microsserviços são os seguintes:
 1. Serviços Autônomos Independentes
 2. Escalabilidade
 3. Descentralização
 4. Serviços resilientes
 5. Balanceamento de carga em tempo real
 6. Disponibilidade
 7. Entrega contínua por meio da integração de DevOps
 8. Integração de API e monitoramento contínuo
 9. Isolamento de falhas
 10. Auto-Provisionamento

Fonte: <https://www.edureka.co/blog/microservices-design-patterns>



Fonte: <https://rogervdkimmenade.blogspot.com/2018/06/microservices-mindmap.html>

***A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.
Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98**

Quando usar microsserviços

Parte 2

Monolith First

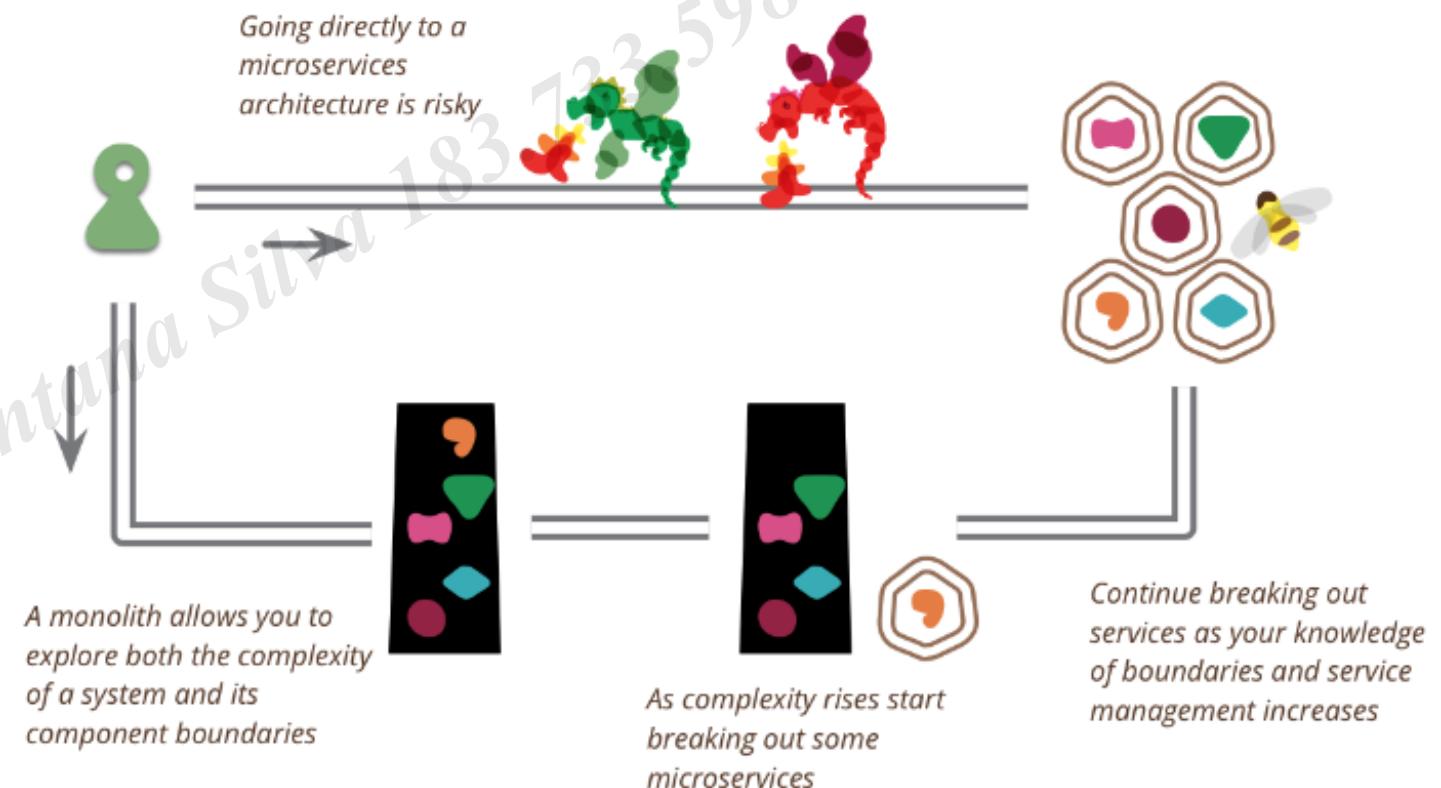


Martin Fowler

As I hear stories about teams using a [microservices architecture](#), I've noticed a common pattern.

1. Almost all the successful microservice stories have started with a monolith that got too big and was broken up
2. Almost all the cases where I've heard of a system that was built as a microservice system from scratch, it has ended up in serious trouble.

This pattern has led many of my colleagues to argue that **you shouldn't start a new project with microservices, even if you're sure your application will be big enough to make it worthwhile.**



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

Microservice Prerequisites



Microservice Prerequisites

28 August 2014



Martin Fowler

MICROSERVICES

As I talk to people about using a microservices architectural style I hear a lot of optimism. Developers enjoy working with smaller units and have expectations of better modularity than with monoliths. But as with any architectural decision there are trade-offs. In particular with microservices there are serious consequences for operations, who now have to handle an ecosystem of small services rather than a single, well-defined monolith. Consequently if you don't have certain baseline competencies, you shouldn't consider using the microservice style.

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

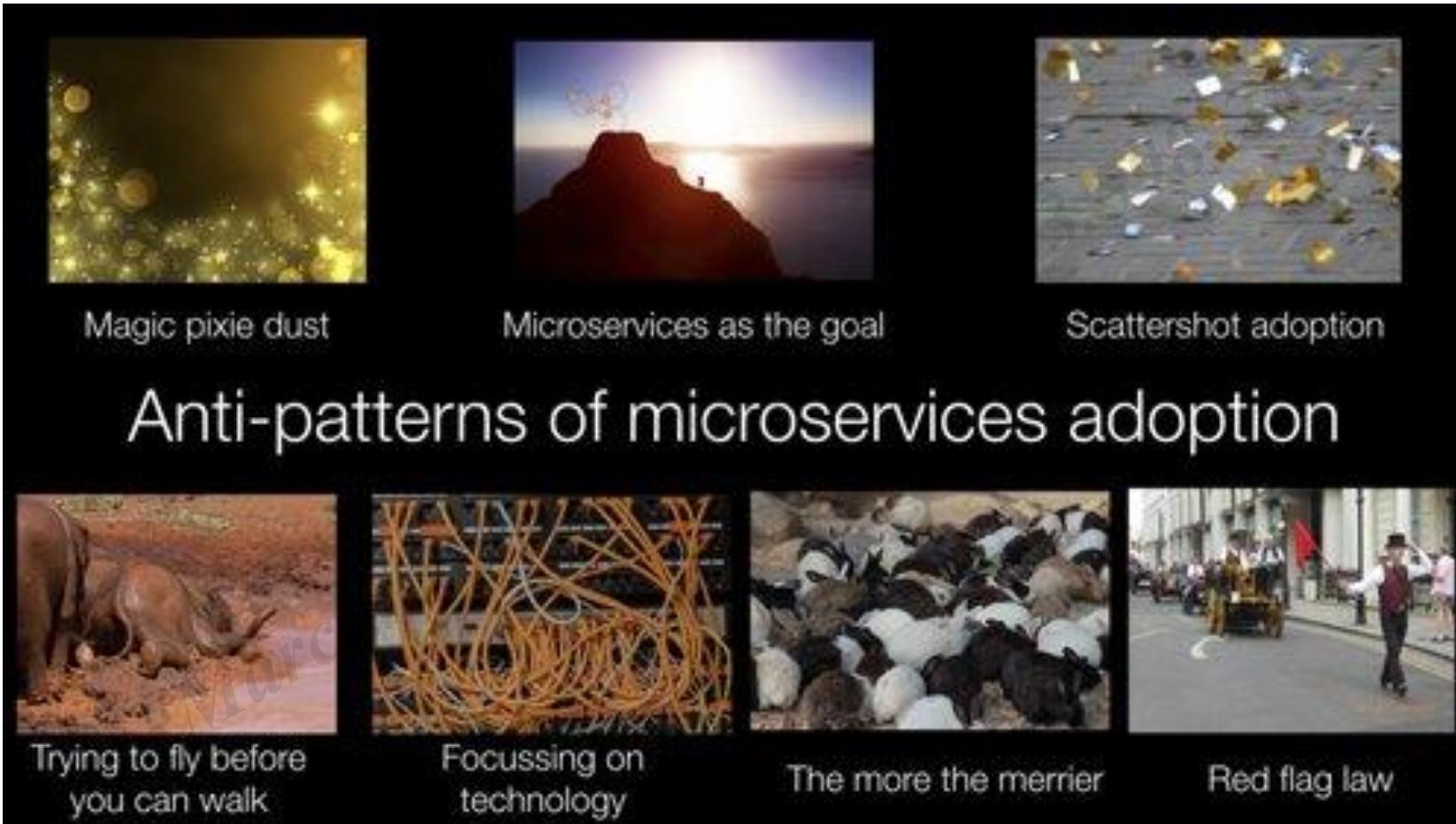
Microservice Prerequisites



1. Rapid provisioning
 1. Cloud
 2. Automation
2. Basic Monitoring
3. Rapid application deployment
4. Development Pipeline
5. DevOps Culture

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

Antipadrões (Antipatterns)



<https://microservices.io/microservices/antipatterns/-/the/series/2019/06/18/microservices-adoption-antipatterns.html>

Antipadrões (Antipatterns)

Pó mágico

- Acreditar que uma pitada de microsserviços resolverá todos os seus problemas de desenvolvimento

Microsserviços como meta

- Tornar a adoção de microsserviços a meta e medir o sucesso em termos do número de serviços escritos

Adoção dispersa

- Tentar adotar a arquitetura de microsserviços sem qualquer coordenação e por várias equipes de desenvolvimento

Tentar voar antes de andar

- Tentar adotar a arquitetura de microsserviços sem praticar técnicas básicas de desenvolvimento, como código limpo, bom design e testes automatizados

Foco em tecnologia

- Concentrar nos microsserviços, mais comumente na infraestrutura de implantação, e negligenciar questões-chave, como a decomposição do serviço

Quanto mais, melhor

- Criar intencionalmente uma arquitetura de microsserviços muito refinada (over architecting)

Red Flag Law

- Manter o mesmo processo de desenvolvimento e estrutura organizacional que foram usados no desenvolvimento de aplicativos monolíticos.

<https://microservices.io/microservices/antipatterns/-/the/series/2019/06/18/microservices-adoption-antipatterns.html>

When Microservices Are a Bad Idea

semaphoreci.com/blog/bad-microservices

Product Customers Resources Blog Podcast Login Sign up

Blog write with us Search over 600 articles

1 Aug 2022 · Software Engineering

When Microservices Are a Bad Idea

Written by: Tomas Fernandez Reviewed by: Dan Ackerson

9 min read Share this f t in

Contents

- Microservices are only viable for mature products
- Microservices aren't the best for on-premise
- Your monolith might have life left
- If it's working, don't fix it
- Brooke's Law and developer productivity
- Are you ready to transition?

On paper, microservices sound wonderful. They are modular, scalable, and fault tolerant. A lot of companies have had great success using this model, so microservices might naturally seem to be the superior architecture and the best way to start new applications.

However, most firms that have succeeded with microservices did not begin with them. Consider the examples of Airbnb and Twitter, which went the microservice route after outgrowing their monoliths and are now [battling its complexities](#). Even successful companies that use microservices appear to still be figuring out the best way to make them work. It is evident that microservices come with their share of tradeoffs.

Migrating from a monolith to microservices is also not a simple task, and creating an untested product as a new microservice is even more complicated. Microservices should only be seriously considered after evaluating the alternative paths.

Learn CI/CD Level up your developer skills to use CI/CD at its max. Start Learning



<https://semaphoreci.com/blog/bad-microservices>

Como responder a essas perguntas ?

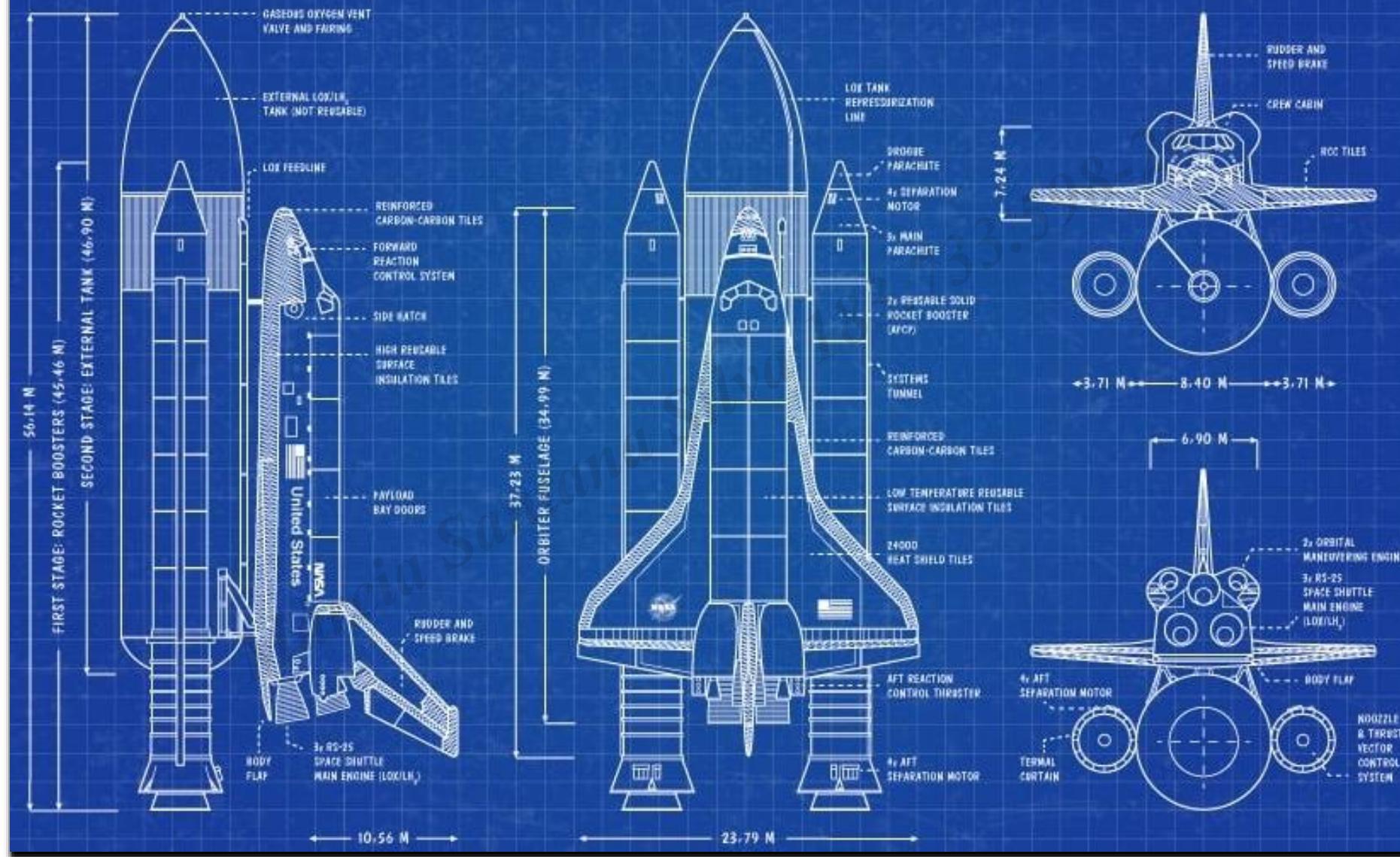
- Como lidar com operações que são de longa duração, processadas em diversos microsserviços diferentes, em inúmeras bases de dados diferentes e ainda assim, garantir integridade?
- Se os microsserviços possuem cada um a sua própria base de dados, como manter a integridade referencial? Como manter os relacionamentos?
- Se o processamento das atividades ocorrerá em inúmeros microsserviços de maneira distribuída, como informar ao usuário que o processamento ocorreu com sucesso ou falha?
- O que acontece quando um microsserviço está fora do ar? Todos os demais também ficarão? Se não, como se dará uma transação quando um microsserviço do ecossistema estiver indisponível?
- Quando um microsserviço for publicado, quais outros deverão ser atualizados? Ou não deveria haver essa dependência?
- Sendo as bases de dados distribuídas, como manter os dados atualizados entre todas elas?
- Posso ter uma base de dados para cada microsserviço, no mesmo servidor e dados?
- Como faria para publicar 20 microsserviços?

Fonte: Adaptado de <https://pt.linkedin.com/pulse/microsservi%C3%A7os-voc%C3%AA-precisa-mesmo-lucas-massena>

Padrões de Arquitetura/ Design Patterns

Parte 3

NASA SPACE SHUTTLE (1981 - 2011)



Padrões de projetos (Design Patterns)

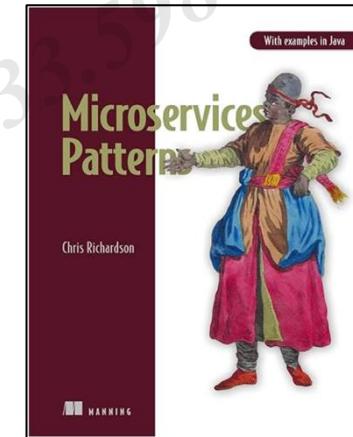


- São similares a esboços para uma construção, que podem ser ajustados para resolver um problema, baseado em microserviços.
- Não é código, mas um conceito que irá influenciar em uma solução.
- Para implementar um padrão de projeto, é necessário escolher e seguir o conceito do padrão escolhido e adapta-lo ao problema a ser resolvido.

Padrões de projetos (Design Patterns)

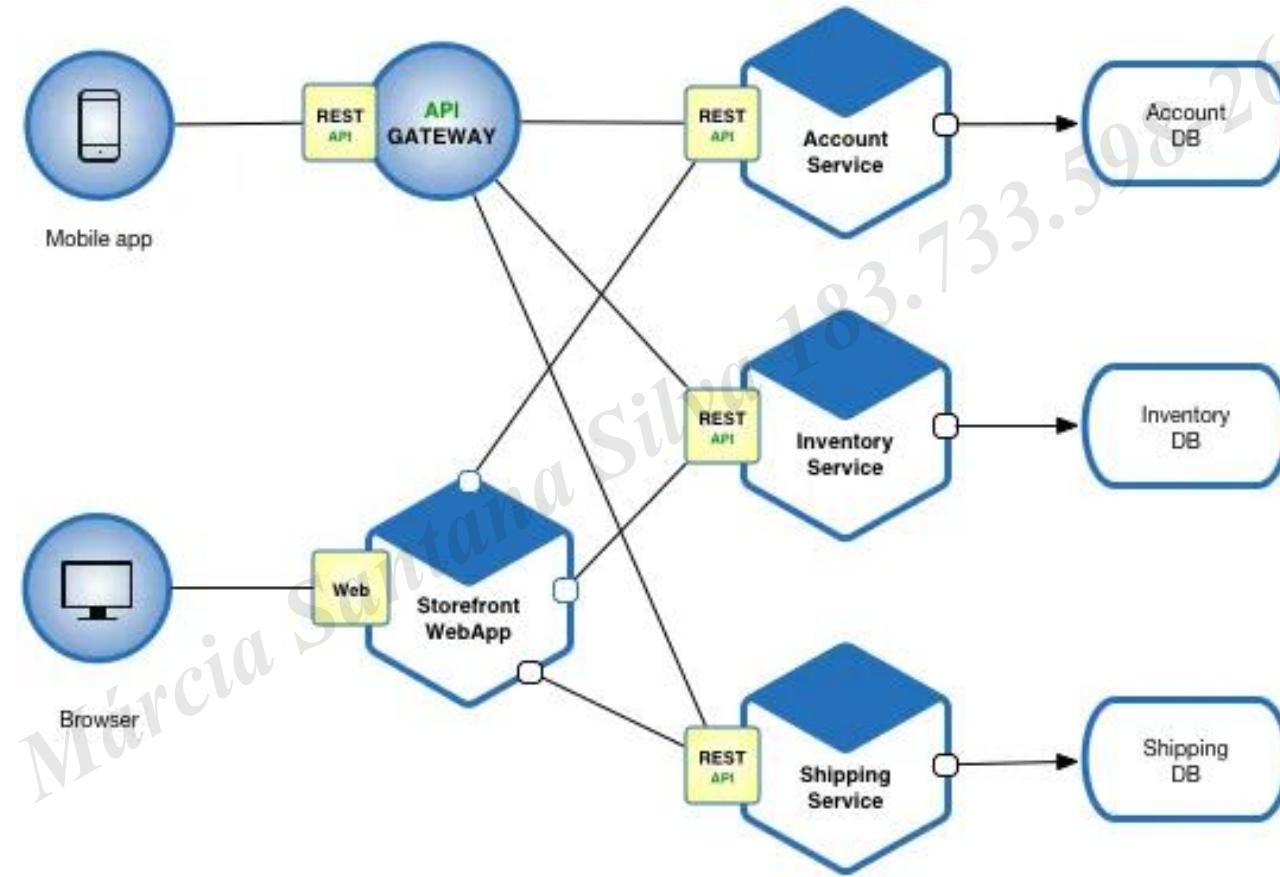
Referência de livro e site

- Chris Richardson
- Site <https://microservices.io/>
- Livro Microservices Patterns



**Microservices Patterns:
With Examples in Java**
Chris Richardson
1ª Edição 2018

Padrões de projetos (Design Patterns)



<https://microservices.io/patterns/microservices.html>

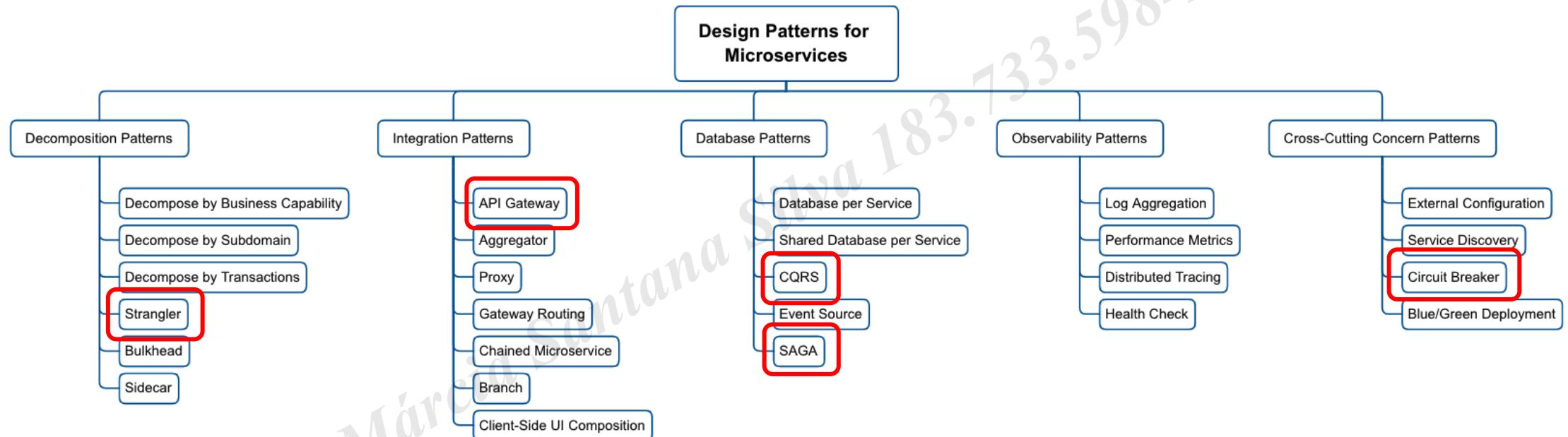
Categorias de Padrões de Projetos para Microsserviços



Padrões de decomposição	Padrões de integração	Padrões de banco de dados	Padrões de observabilidade	Padrões de preocupações transversais (Cross-Cutting Concerns)
<ul style="list-style-type: none">• Esses padrões nos dizem como decompor aplicações, por exemplo uma aplicação monolítica para microsserviços	<ul style="list-style-type: none">• Cobrem principalmente aspectos de comunicação de microsserviços• Recomendam como os microsserviços podem se comunicar entre si e com outros sistemas e como agregar resultados de vários serviços antes de enviá-los de volta ao usuário	<ul style="list-style-type: none">• Recomendam estratégias de armazenamento de dados para microsserviços• Ajudam a resolver problemas de dados como sincronização de dados, atomicidade, ACID e compartilhamento de dados• Endereçam a integridade dos dados, como lidar com transações entre microsserviços e transações distribuídas	<ul style="list-style-type: none">• Tem objetivo de rastrear e monitorar a integridade e o desempenho de microsserviços• Um exemplo é enviar uma notificação se houver uma falha no microsserviço	<ul style="list-style-type: none">• Esses padrões são outras preocupações que auxiliam na manutenção dos microsserviços• Não são ou fazem partes da lógica de negócios, mas orientam práticas recomendadas para criar, manter e proteger microsserviços

Fonte: Adaptado de <https://www.pluralsight.com/courses/microservices-playbook-architectural-design-patterns> e <https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

Categorias de Padrões de Projetos para Microsserviços



Fonte: Adaptado de <https://www.pluralsight.com/courses/microservices-playbook-architectural-design-patterns> e <https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>

Design Patterns em nuvem pública: AWS

Recomendações da AWS

Habilitar a persistência de dados em
microsserviços

Introdução

- ▶ Padrões para permitir a persistência de dados

Perguntas frequentes

Recursos

Histórico do documento

Glossário

https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-data-persistence/welcome.html

Recomendações da AWS

Decompor monólitos em microsserviços

Introdução

- ▶ Padrões para decomposição de monólitos

Perguntas frequentes

Recursos

Histórico do documento

Glossário

https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-decomposing-monoliths/welcome.html

Recomendações da AWS

Integração de microsserviços usando
serviços da tecnologia sem servidor.

Introdução

- ▶ Padrões para integrar microsserviços

Perguntas frequentes

Recursos

Histórico do documento

Glossário

https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-integrating-microservices/welcome.html

AWS Orientação prescritiva

Padrões, arquiteturas e implementações de
design de nuvem

Introdução

Padrão de camada anticorrupção

- ▶ Padrões de roteamento de API

Padrão de disjuntor

Padrões de orquestração e coreografia

Tente novamente com o padrão de recuo

Orquestração da saga

Padrão de figo Strangler

Padrão de caixa de saída transacional

Recursos

Histórico do documento

https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/cloud-design-patterns/introduction.html

Design Patterns em nuvem pública: Azure

Learn / Azure / Architecture Center /

Cloud Design Patterns

Article • 04/13/2023 • 24 contributors

Feedback

In this article

[Challenges in cloud development](#)

[Catalog of patterns](#)

These design patterns are useful for building reliable, scalable, secure applications in the cloud.

Each pattern describes the problem that the pattern addresses, considerations for applying the pattern, and an example based on Microsoft Azure. Most patterns include code samples or snippets that show how to implement the pattern on Azure. However, most patterns are relevant to any distributed system, whether hosted on Azure or other cloud platforms.

Microsoft Azure Architecture Center – Cloud Design Patterns

<https://learn.microsoft.com/en-us/azure/architecture/patterns/>

Conteúdo sobre Design Patterns

- <https://medium.com/@madhukaudantha/microservice-architecture-and-design-patterns-for-microservices-e0e5013fd58a>
- <https://medium.com/@mail.ruchitayal/mind-mapping-microservices-design-patterns-dd1cfdba8dae>

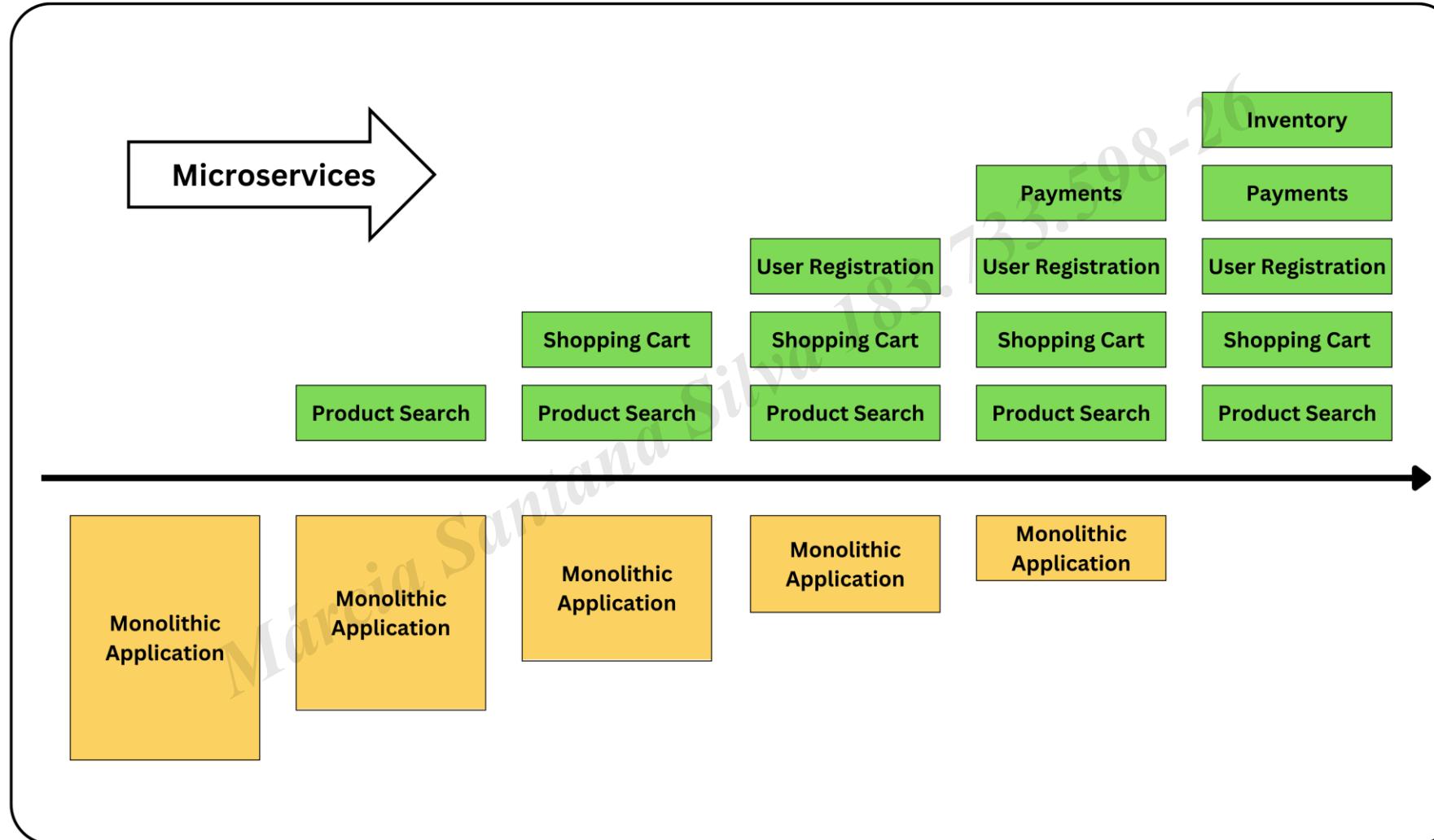
Padrão de Decomposição: Strangler



- Esse padrão é comumente usado para transformar incrementalmente um aplicativo monolítico em microserviços, substituindo uma funcionalidade específica por um novo serviço.
- O objetivo é que o legado e as versões novas e modernizadas coexistam
- O novo sistema é inicialmente suportado e encapsulado pelo sistema existente. Esse suporte dá ao novo sistema tempo para crescer e potencialmente substituir totalmente o sistema antigo.
- O padrão possui 3 etapas: transformar, coexistir e eliminar:
 - Transformar (transform): Crie uma nova aplicação paralela com abordagens modernas
 - Coexistir (coexist): Deixe a aplicação existente onde está por um tempo. Redirecione partes da a aplicação existente para a nova aplicação e implemente novas funcionalidades incrementalmente.
 - Eliminar (eliminate): Remova as funcionalidades antigas aplicação existente.

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-decomposing-monoliths/strangler-fig.html

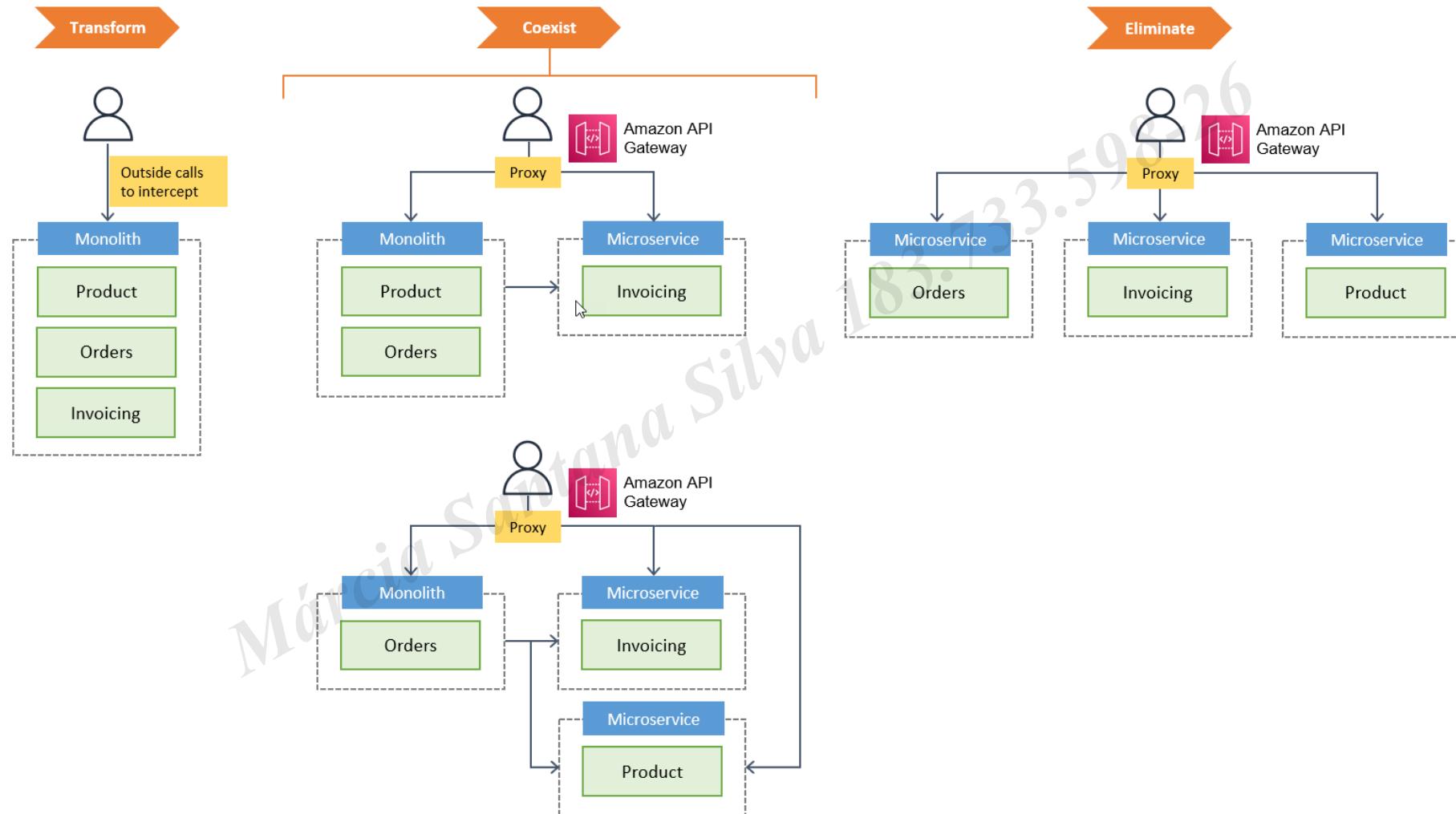
Padrão de Decomposição: Strangler



Fonte: <https://amplication.com/blog/monoliths-to-microservices-using-the-strangler-pattern>



Padrão de Decomposição: Strangler



Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-decomposing-monoliths/strangler-fig.html

Padrão de Decomposição: Strangler



Vantagens	Desvantagens
Permite uma migração simples de um serviço para um ou mais serviços	Não é adequado para sistemas pequenos em que a complexidade é baixa e o tamanho é pequeno
Mantém os serviços antigos em funcionamento enquanto refatora para versões atualizadas	Não pode ser usado em sistemas em que as solicitações para o sistema back-end não podem ser interceptadas e roteadas
Oferece a capacidade de adicionar novos serviços e funcionalidades enquanto refatora serviços antigos	A camada de proxy ou fachada (façade) pode se tornar um ponto único de falha ou um gargalo de desempenho se não for projetada adequadamente
O padrão pode ser usado para controle de versão de APIs	Requer um plano de reversão para cada serviço refatorado para voltar à maneira antiga de fazer as coisas com rapidez e segurança se as coisas derem errado
O padrão pode ser usado para interações antigas para soluções que não são ou não serão atualizadas	

Fonte: Adaptado de https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-decomposing-monoliths/strangler-fig.html

How to break a Monolith into Microservices



How to break a Monolith into Microservices

What to decouple and when

As monolithic systems become too large to deal with, many enterprises are drawn to breaking them down into the microservices architectural style. It is a worthwhile journey, but not an easy one. We've learned that to do this well, we need to start with a simple service, but then draw out services that are based on vertical capabilities that are important to the business and subject to frequent change. These services should be large at first and preferably not dependent upon the remaining monolith. We should ensure that each step of migration represents an atomic improvement to the overall architecture.

24 April 2018

<https://martinfowler.com/articles/break-monolith-into-microservices.html>

Padrão de Integração: API Gateway



- O padrão de gateway de API é recomendado para projetar e criar aplicativos complexos ou grandes baseados em microsserviços com vários aplicativos clientes. O padrão é semelhante ao padrão de fachada (façade) do design orientado a objetos, mas faz parte de um proxy reverso de sistema distribuído ou roteamento de gateway e usa um modelo de comunicação síncrona.
- O padrão fornece um proxy reverso para redirecionar ou rotear solicitações para seus endpoints de microsserviços internos. Um gateway de API fornece um único endpoint ou URL para os aplicativos cliente e mapeia internamente as solicitações para microsserviços internos. Uma camada de abstração é fornecida ocultando certos detalhes da implementação e funcionalidades adicionais também podem ser incluídas ao serviço de back-end, como transformações de resposta e solicitação, autorização de acesso ao endpoint ou rastreamento.
- Você deve considerar o uso do padrão de gateway da API se:
 - O número de dependências de um microsserviço é gerenciável e não cresce com o tempo
 - O sistema de chamada exige uma resposta síncrona do microsserviço
 - Existe o requisito de baixa latência
 - É preciso expor uma API para coletar dados de vários microsserviços

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-integrating-microservices/api-gateway-pattern.html

Padrão de Integração: API Gateway

Caso de Uso

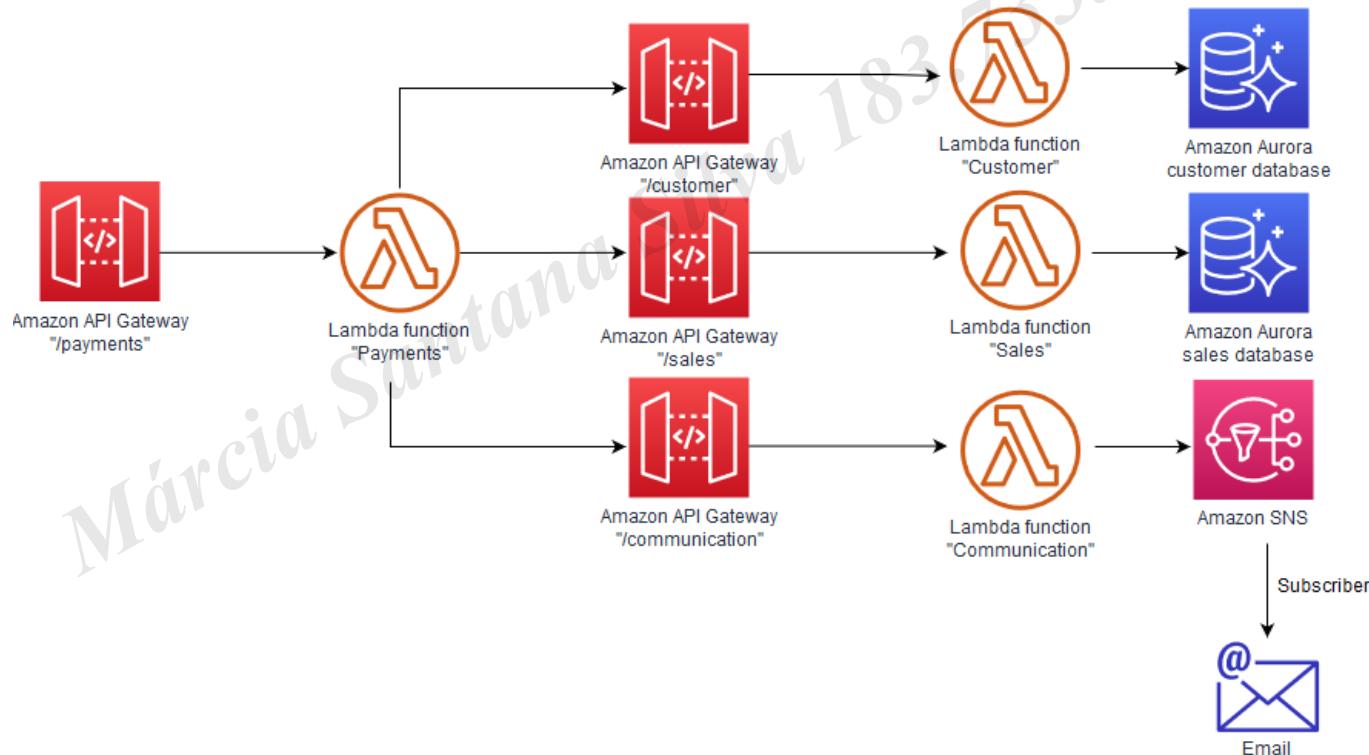
- Um cliente faz pagamentos mensais regulares em um sistema de seguro que consiste em quatro microsserviços (“Cliente”, “Comunicação”, “Pagamentos” e “Vendas”). O microsserviço “Cliente” atualiza o banco de dados do cliente com os detalhes do pagamento mensal. O microsserviço “Vendas” atualiza o banco de dados de vendas com informações relevantes que ajudam a equipe de vendas a acompanhar o cliente em busca de oportunidades de venda cruzada. O microsserviço “Comunicação” envia um e-mail de confirmação ao cliente após o pagamento ser processado com sucesso. Finalmente, o microsserviço “Pagamentos” é o sistema geral que o cliente usa para fazer seu pagamento mensal. O padrão usa serviços da web para integrar os subsistemas “Cliente”, “Vendas” e “Comunicação” com o microsserviço “Pagamentos”.
- Há três desafios em usar esse padrão para esse caso de uso:
 1. As chamadas síncronas são feitas para sistemas downstream, o que significa que qualquer latência causada por esses subsistemas afeta o tempo geral de resposta.
 2. Os custos operacionais são mais altos porque o sistema de “Pagamentos” aguarda as respostas dos outros microsserviços antes de responder ao sistema de chamadas. O tempo total de execução é, portanto, relativamente maior em comparação com um sistema assíncrono.
 3. O tratamento de erros e a repetição são tratados separadamente para cada microsserviço dentro do sistema de “Pagamentos”, não pelos microsserviços individuais.

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-integrating-microservices/api-gateway-pattern.html

Padrão de Integração: API Gateway

Caso de Uso

- Cenário 1: Cada microserviço tem seu próprio gateway de API. O microserviço “Pagamentos” chama sistemas individuais e implementa o padrão de gateway da API.

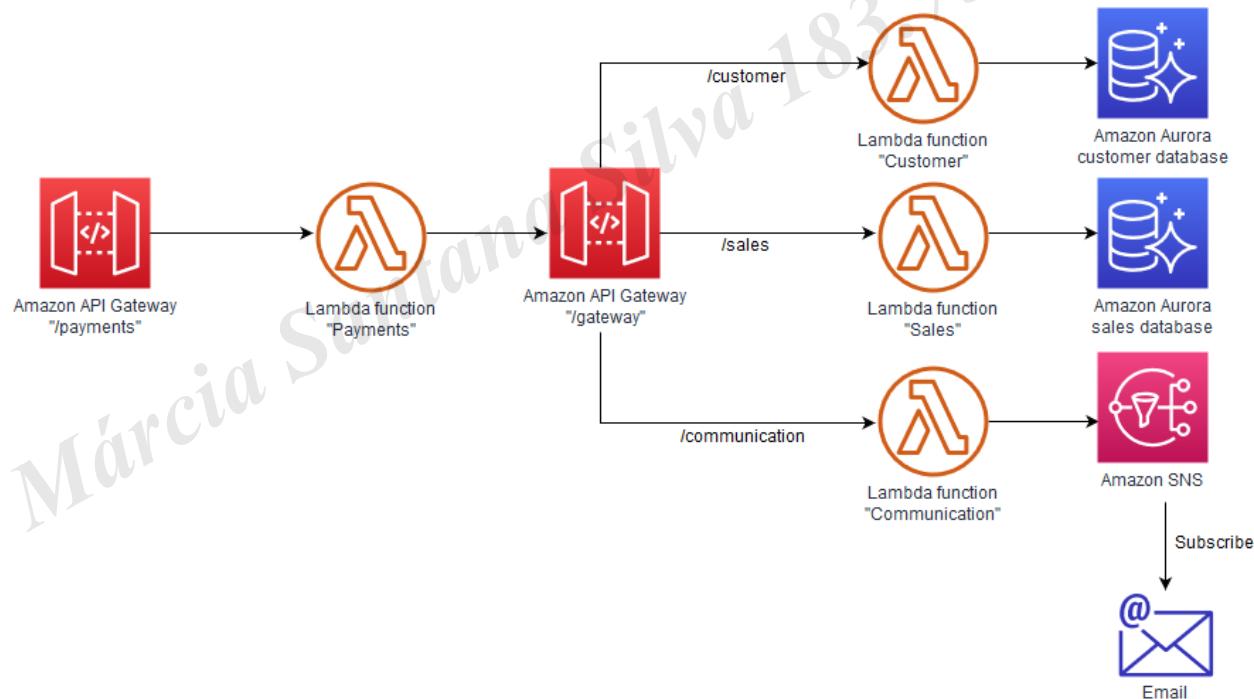


Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-integrating-microservices/api-gateway-pattern.html

Padrão de Integração: API Gateway

Caso de Uso

- Cenário 2: Cada microserviço é implantado como uma função (Serverless), mas todos os microserviços são conectados pelo mesmo gateway de API.



Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-integrating-microservices/api-gateway-pattern.html

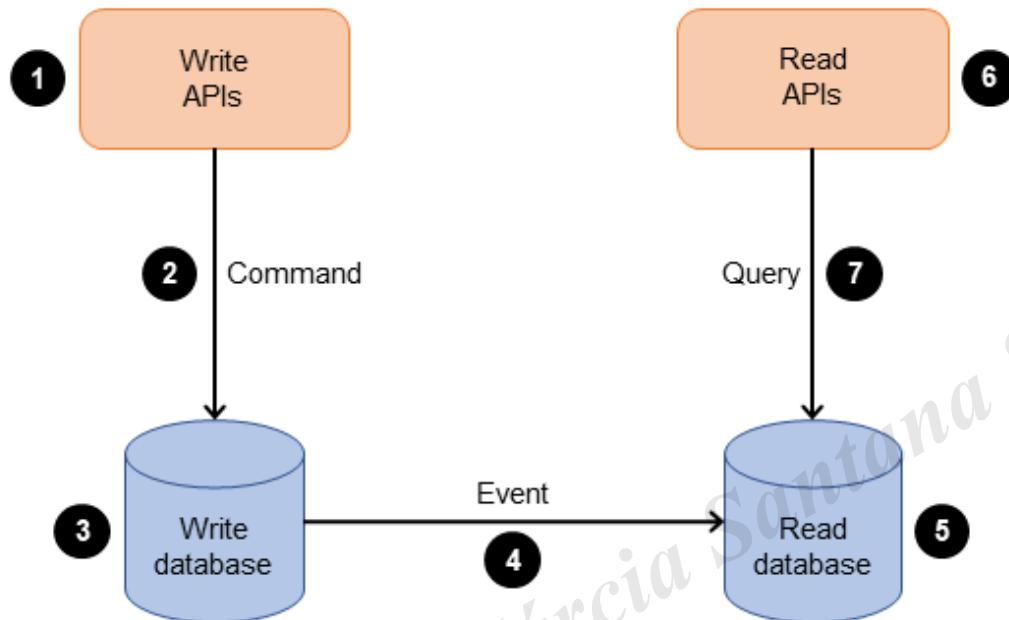
Padrão de Banco de Dados: CQRS



- O padrão de segmentação de responsabilidade de consulta de comando (CQRS - Command Query Responsibility Segregation) separa a mutação de dados, ou a parte do comando de um sistema, da parte de consulta.
- Você pode usar o padrão CQRS para separar atualizações e consultas se elas tiverem requisitos diferentes de taxa de throughput, latência ou consistência.
- O padrão CQRS divide o aplicativo em duas partes: o lado do comando e o lado da consulta.
 - O lado do comando trata das solicitações create, update e delete
 - O lado da consulta executa a parte query usando as réplicas de leitura.

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html

Padrão de Banco de Dados: CQRS



O diagrama mostra o seguinte processo:

1. A empresa interage com o aplicativo enviando comandos por meio de uma API.
2. Comandos são ações como criar, atualizar ou excluir dados.
3. O aplicativo processa o comando de entrada no lado do comando. Isso envolve validar, autorizar e executar a operação.
4. O aplicativo persiste os dados do comando no banco de dados de gravação (comando).
5. Depois que o comando é armazenado no banco de dados de gravação, os eventos são disparados para atualizar os dados no banco de dados de leitura (consulta).
6. O banco de dados de leitura (consulta) processa e persiste os dados. Os bancos de dados de leitura são projetados para serem otimizados para requisitos de consulta específicos.
7. A empresa interage com APIs de leitura para enviar consultas para o lado da consulta do aplicativo.
8. O aplicativo processa a consulta de entrada no lado da consulta e recupera os dados do banco de dados lido.

Fonte: <https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/cqrs-pattern.html>

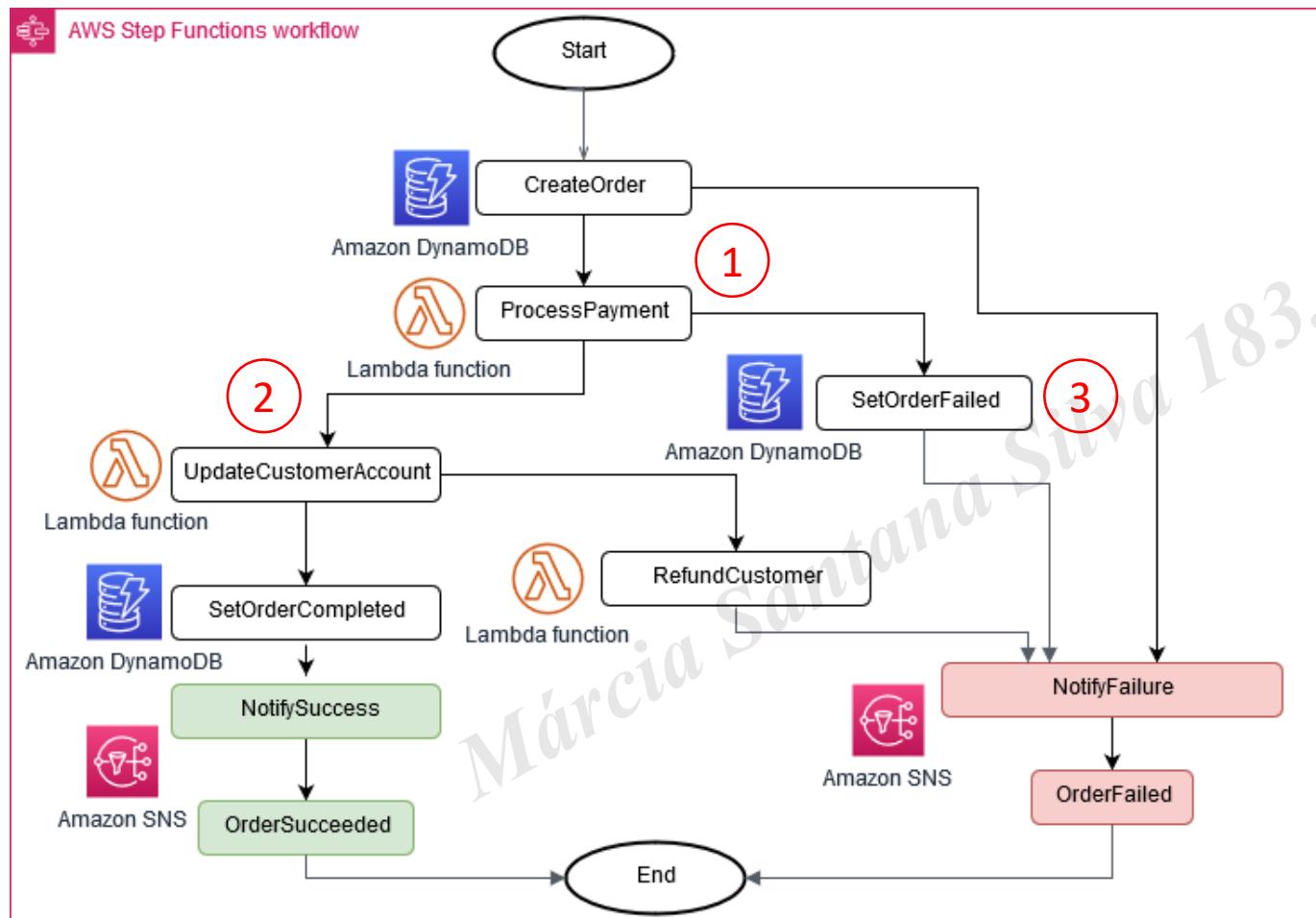
Padrão de Banco de Dados: SAGA



- O padrão saga é um padrão de gerenciamento de falhas que ajuda a estabelecer a consistência em aplicativos distribuídos e coordena as transações entre vários microsserviços para manter a consistência de dados.
- Um microsserviço publica um evento para cada transação e a próxima transação é iniciada com base no resultado do evento. Ele pode seguir dois caminhos diferentes, dependendo do êxito ou da falha das transações.

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html

Padrão de Banco de Dados: SAGA



A ilustração a seguir mostra como o padrão SAGA implementa um sistema de processamento de pedidos em um provedor de nuvem pública:

1. Cada etapa (por exemplo, "ProcessPayment") tem etapas separadas para lidar com o sucesso ou Falha do processo
2. Sucesso "UpdateCustomerAccount"
3. Falha "SetOrderFailure"

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html



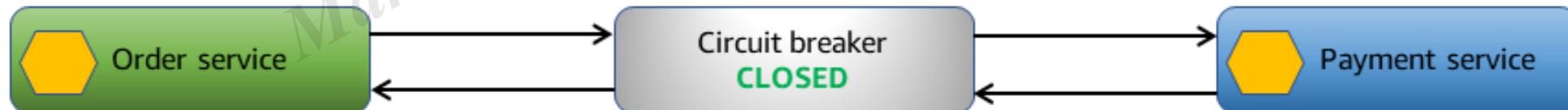
Padrão de preocupações transversais: Disjuntor ou Circuit Breaker

- O padrão Disjuntor ou Circuit Breaker ajuda a evitar a ocorrência de falhas em cascata, quando microserviços estiverem indisponíveis ou com alta latência.
- O padrão do disjuntor pode impedir que um microserviço repita uma chamada para outro microserviço quando a chamada já causou interrupções ou falhas repetidas.
- O padrão também é usado para detectar quando o serviço de chamada está funcionando novamente.

Fonte: https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/modernization-data-persistence/saga-pattern.html

Padrão de preocupações transversais: Disjuntor ou Circuit Breaker

- No exemplo a seguir, temos dois microserviços:
 1. O microserviço “Order” (Pedidos) que inicia a chamada
 2. O microserviço “Payment” (Pagamentos) que recebe a chamada
- Quando não há falhas, o serviço de Pedidos encaminha todas as chamadas para o serviço de Pagamentos pelo Circuit Breaker (Disjuntor), como mostra o diagrama a seguir:



Fonte: <https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/circuit-breaker.html>

Padrão de preocupações transversais: Disjuntor ou Circuit Breaker

- Se o serviço de pagamento atingir o tempo limite, o Disjuntor poderá detectar o tempo limite e rastrear a falha:



- Se os tempos limite excederem um limite especificado, o aplicativo abrirá o circuito. Quando o circuito está aberto, o Disjuntor não encaminha as chamadas para o microserviço de Pagamentos. Ele retorna uma falha imediata quando o microserviço de Pedidos liga para o serviço de pagamento.



Fonte: <https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/circuit-breaker.html>

Padrão de preocupações transversais: Disjuntor ou Circuit Breaker

- O objeto disjuntor tenta verificar periodicamente se as chamadas para o serviço de pagamento foram bem-sucedidas:



Circuit breaker periodically retries payment service

- Quando a chamada para o serviço de pagamento é bem-sucedida, o circuito é fechado e todas as outras chamadas são encaminhadas para o serviço de pagamento novamente:



Circuit breaker with working payment service

Fonte: <https://docs.aws.amazon.com/prescriptive-guidance/latest/cloud-design-patterns/circuit-breaker.html>



Um exemplo de etapas para construção de uma aplicação baseada em microsserviços

1. Definição e Planejamento	2. Desenvolvimento Individual dos Microsserviços	3. Comunicação Entre Microsserviços	4. Gerenciamento de Dados	5. Testes e Integração
<ul style="list-style-type: none">Identifique as funcionalidades do seu aplicativo para dividi-las em microsserviçosDefina as interfaces e a forma de comunicação (APIs REST, mensagens em fila, eventos assíncronos, etc.)Planeje a estrutura de cada microsserviço	<ul style="list-style-type: none">Desenvolva cada microsserviço como um projeto independenteUse um framework para criar APIs	<ul style="list-style-type: none">Configure a forma como os microsserviços irão se comunicar;Use bibliotecas para implementar a comunicação	<ul style="list-style-type: none">Decida como os microsserviços irão armazenar e acessar dadosPode-se compartilhar um banco de dados centralizado ou usar bancos de dados separados para cada microsserviçoLide com consistência de dados entre os microsserviços	<ul style="list-style-type: none">Teste cada microsserviço de forma independenteImplemente testes de integração para garantir que os microsserviços interajam corretamente.

<https://www.homehost.com.br/blog/python/django/microsservicos-python/>

Um exemplo de etapas para construção de uma aplicação baseada em microsserviços



6. Implantação	7. Monitoramento e Observabilidade	8. Escalabilidade	9. Segurança	10. Documentação
<ul style="list-style-type: none">• Use contêineres Docker para empacotar cada microsserviço com suas dependências• Use ferramentas como Kubernetes ou Docker Compose para implantar e, também, orquestrar os microsserviços.	<ul style="list-style-type: none">• Implemente monitoramento para cada microsserviço e para a comunicação entre eles• Use ferramentas como Prometheus, Grafana, ELK (Elasticsearch, Logstash, Kibana) para monitorar a saúde e o desempenho	<ul style="list-style-type: none">• Implemente escalabilidade automática para lidar com picos de tráfego• Use ferramentas de orquestração para adicionar ou remover instâncias de microsserviços conforme necessário	<ul style="list-style-type: none">• Proteja as APIs com autenticação e autorização adequadas• Mantenha as dependências atualizadas para evitar vulnerabilidades conhecidas	<ul style="list-style-type: none">• Documente cada microsserviço, suas APIs e como eles se comunicam• Forneça uma documentação clara para ajudar no desenvolvimento, manutenção e integração com outros sistemas

<https://www.homehost.com.br/blog/python/django/microsservicos-python/>

Identificando limites de microsserviços

- Cada serviço tem uma única responsabilidade.
- Não há nenhuma chamada de conversa entre os serviços. Se a divisão da funcionalidade em dois serviços gerar excesso de conversas, isso poderá ser um sintoma de que essas funções pertencem ao mesmo serviço.
- Cada serviço é pequeno o suficiente para ser criado por uma pequena equipe trabalhando de forma independente.
- Não há nenhuma interdependência que exige que dois ou mais serviços sejam implantados em sincronia. Deve ser sempre possível implantar um serviço sem redistribuir outros serviços.
- Os serviços não estão acoplados de forma firme e podem evoluir de forma independente.
- Os limites do serviço não criarão problemas de consistência ou integridade de dados. Às vezes, é importante manter a consistência dos dados colocando a funcionalidade em um único microsserviço. Dito isto, considere se você realmente precisa de consistência forte. Existem estratégias para abordar a eventual consistência em um sistema distribuído, e os benefícios dos serviços de decomposição geralmente superam os desafios de gerenciar a consistência eventual.

<https://learn.microsoft.com/pt-br/azure/architecture/microservices/model/microservice-boundaries>

Identificando limites de microsserviços

- “Right Size” Microservices
- <https://www.softplan.com.br/en/tech-writers/microsservicos-do-tamanho-certo-parte-i/>

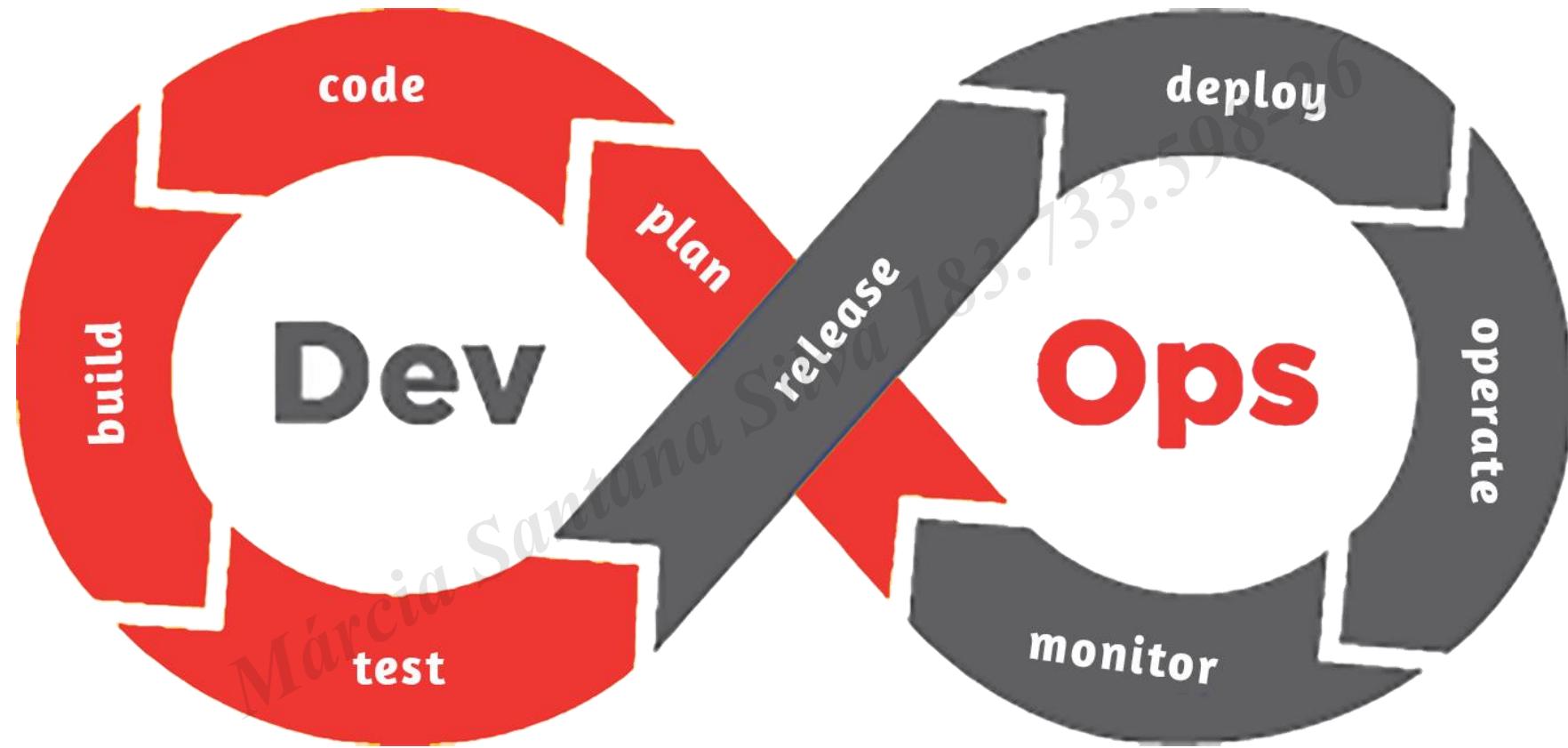
Métodos Ágeis DevOps

Parte 4

Métodos Ágeis

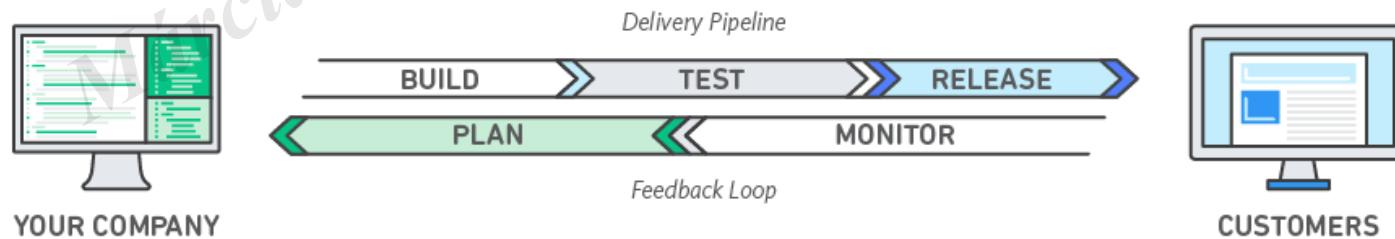


- Lean
- Scrum
- Kanban
- Extreme Programming (XP)
- Design Sprint



O que é DevOps?

- O DevOps é a combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura.
- Essa velocidade permite que as empresas atendam melhor aos seus clientes e consigam competir de modo mais eficaz no mercado.

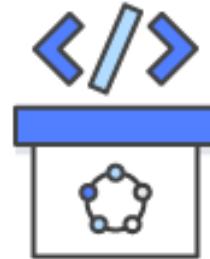


Fonte: <https://aws.amazon.com/pt/devops/what-is-devops/>

Práticas de DevOps



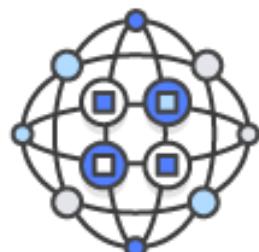
Integração Contínua
(CI: Continuous Integration)



Entrega contínua
(CD: Continuous Delivery)



Infraestrutura como Código
(IaC: Infrastructure as a Code)



Microsserviços



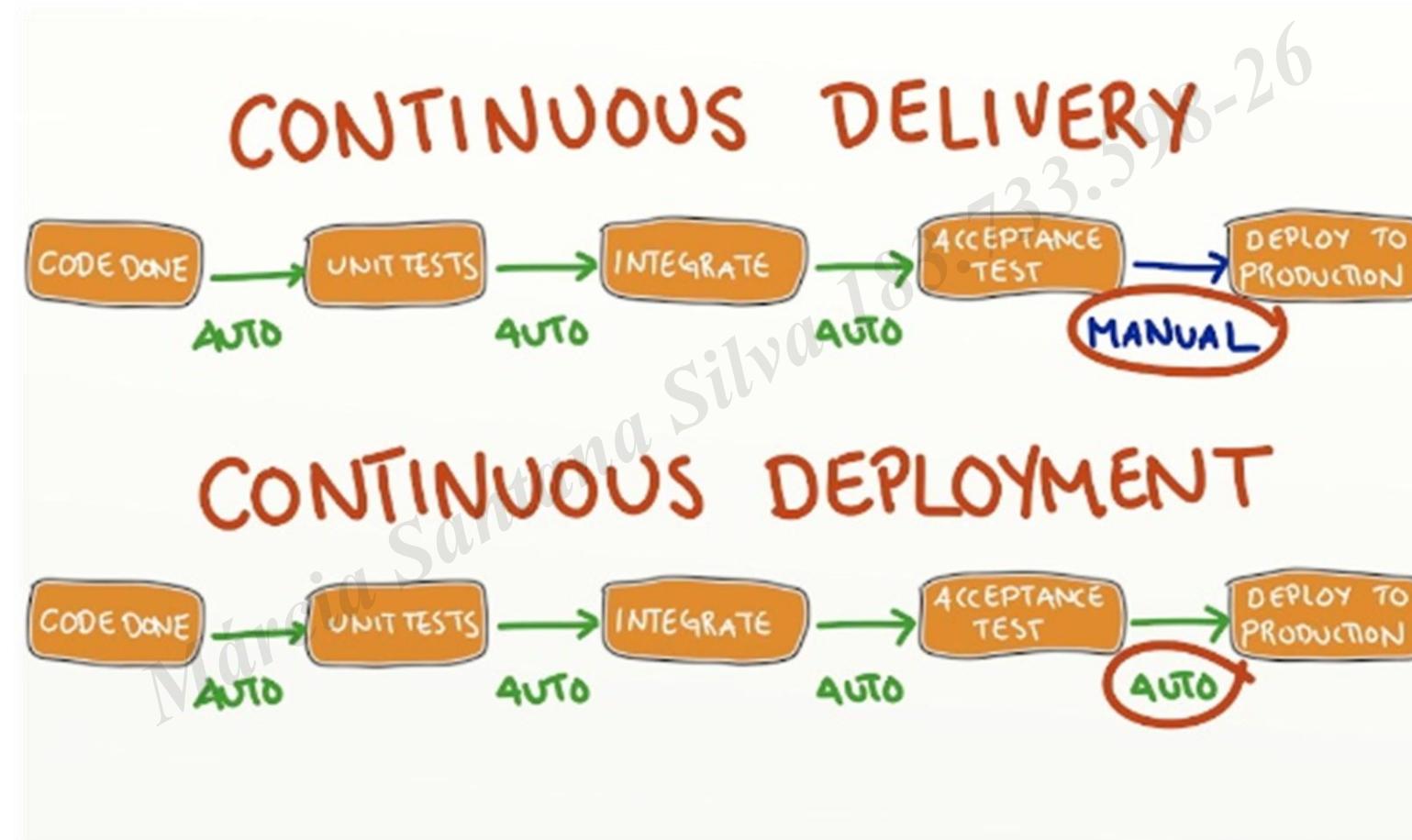
Monitoramento e
Logs



Comunicação e
colaboração

Fonte: Adaptado de <https://aws.amazon.com/pt/devops/what-is-devops/>

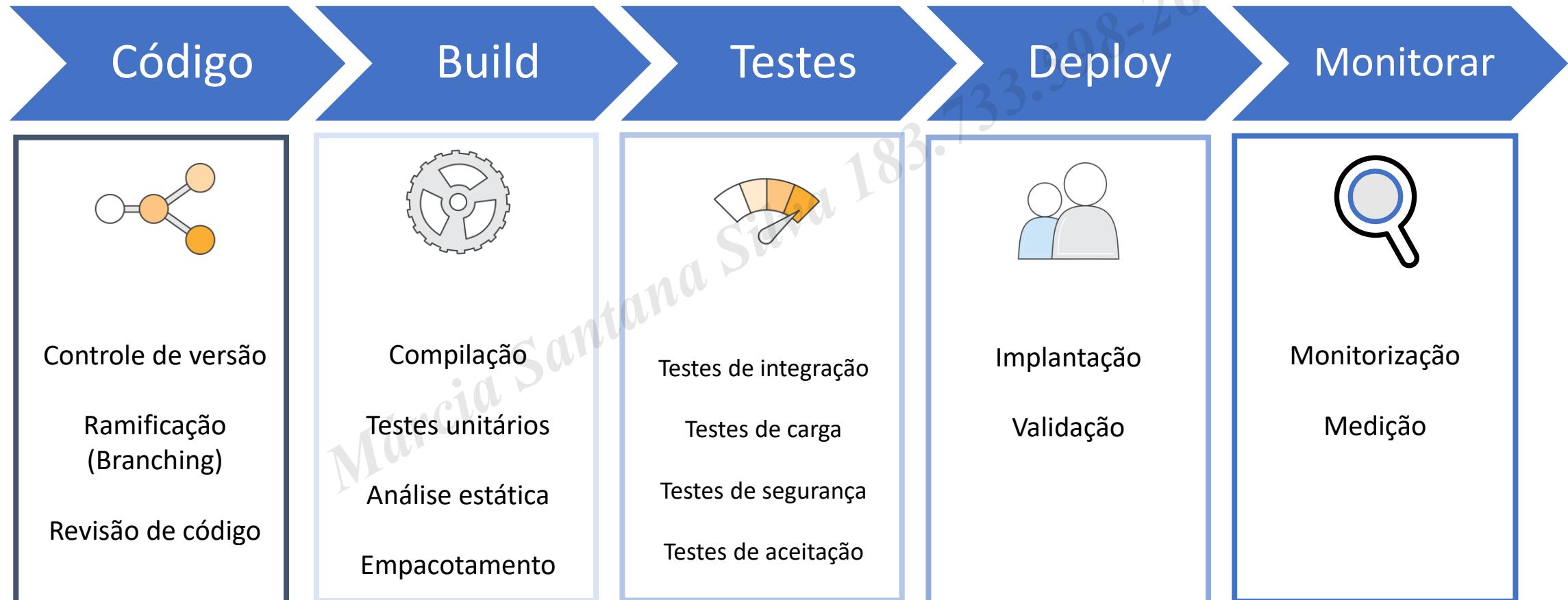
Continuous Delivery + Continuous Deployment



Fonte: <https://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>



Continuous Integration & Continuous Delivery (CI/CD)

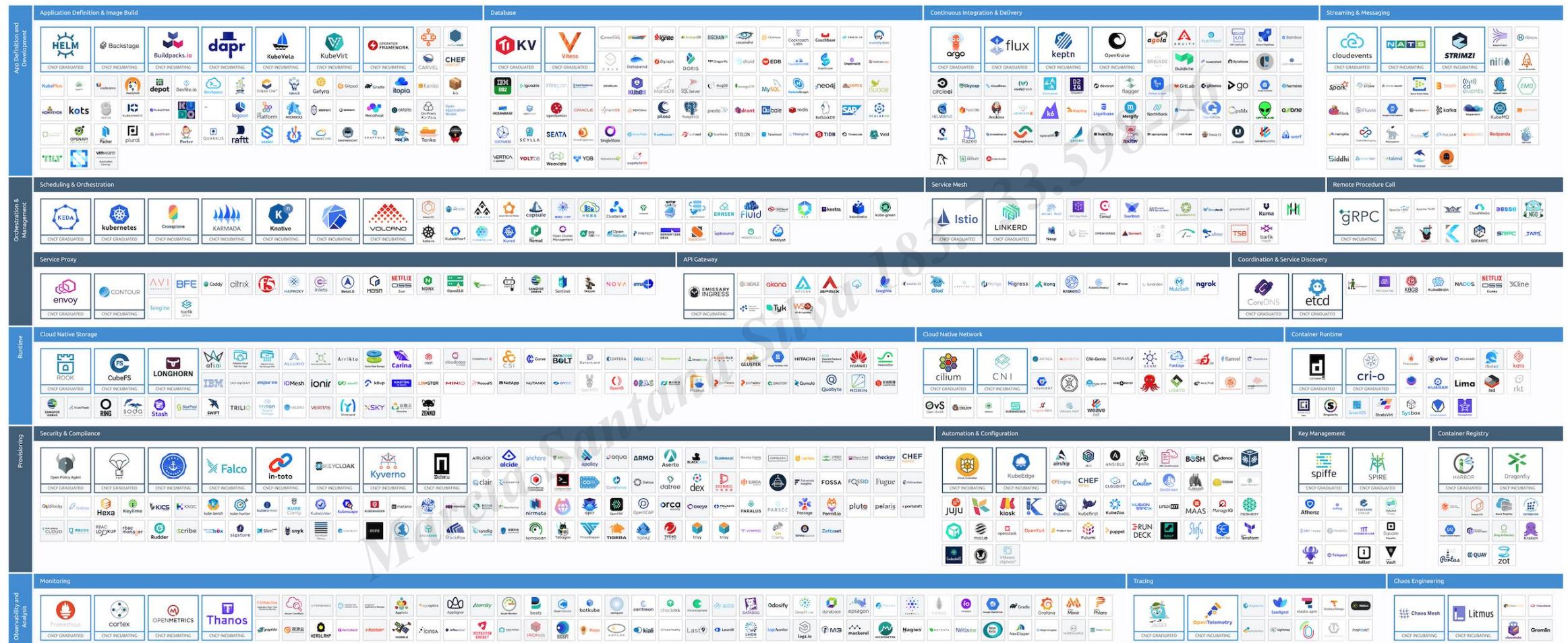


Fonte: Adaptado de <https://aws.amazon.com/pt/devops/what-is-devops/>

Ferramentas, Linguagens, Frameworks

Parte 5

Cloud Native Landscape



Fonte: <https://landscape.cncf.io/>

***A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.
Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98**

Algumas ferramentas

Containerization

- Docker & Docker Compose
- Kubernetes

Service mesh

- Istio
- Linkerd
- Consul Connect

API gateway

- Ambassador
- Kong
- Apiman
- WSO2 API Manager
- Apigee

Monitoring tools

- New Relic
- CloudWatch
- Datadog
- Prometheus
- Grafana

Log consolidation tools

- Fluentd
- Graylog
- Splunk
- ELK (Elasticsearch, Logstash, Kibana)

Tracing tools

- Zipkin
- Jaeger
- New relic
- Splunk

CI/CD tools

- Jenkins
- GitLab CI/CD
- CircleCI

API Documentation

- OpenAPI

Algumas linguagens



Fonte: Imagens abertas fabricantes

Alguns frameworks



Go kit



Fonte: Imagens abertas fabricantes



The State of Developer Ecosystem 2023

JetBrains



<https://www.jetbrains.com/lp/devecosystem-2023/>

Modelos de Maturidade

Parte 6

Microservices 2022

JetBrains

Do you develop microservices?



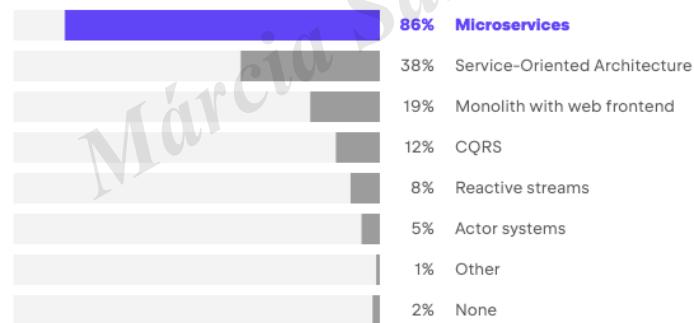
No • 63%
Yes • 37%

This question was shown to a general survey audience and was used as a qualification question to the Microservices section.

37%

of all respondents develop microservices, which is just two percentage points higher in comparison to 2021.

What approaches do you use in your system design?

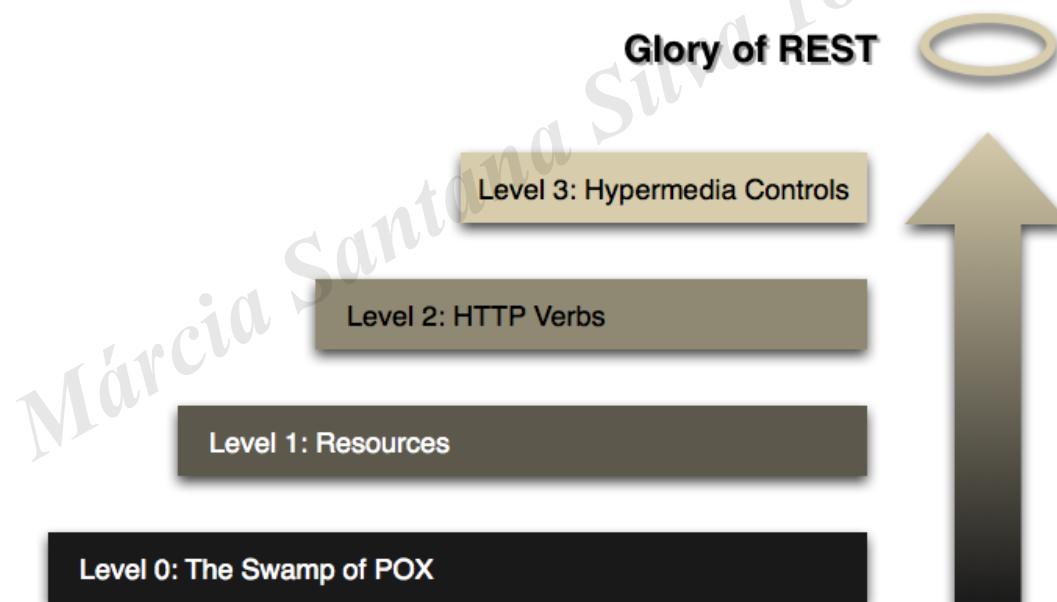


There is a significant decrease in the usage of "Monolith with web frontend" approach, down from 25% in 2021 to 19% in 2022.

<https://www.jetbrains.com/lp/devcosystem-2022/microservices/>

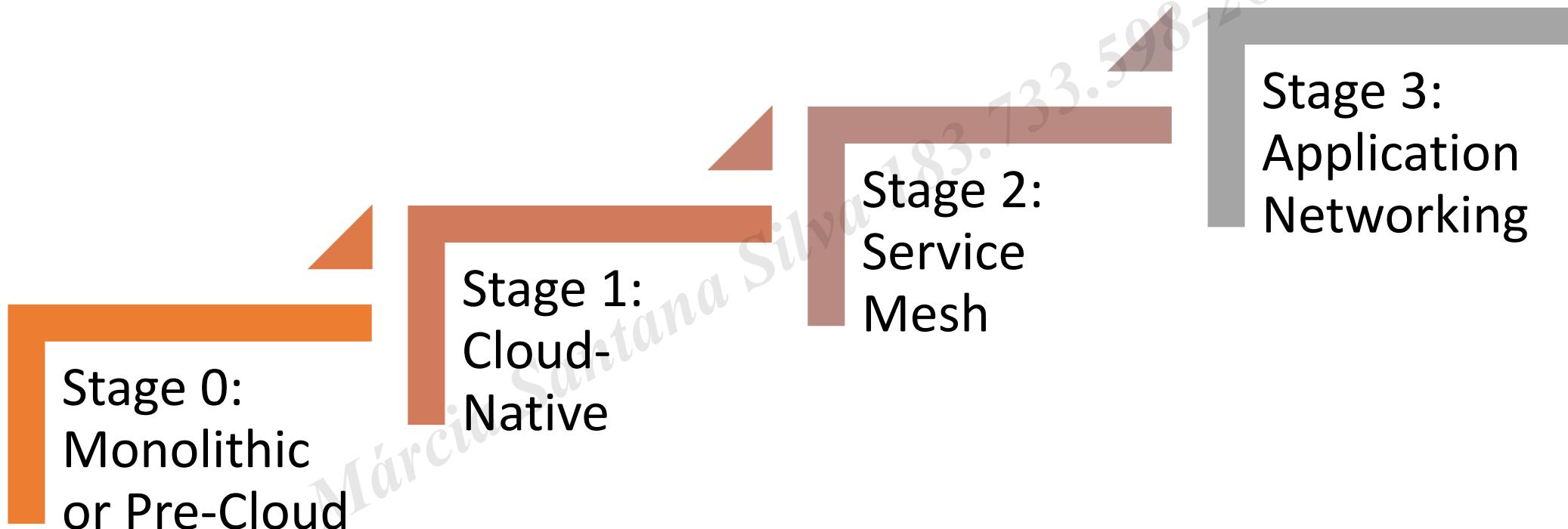
Richardson Maturity Model

- Um modelo, desenvolvido por Leonard Richardson, que divide os principais elementos de uma abordagem REST em três etapas. Eles introduzem recursos, verbos http e controles de hiper mídia.



Fonte: <https://martinfowler.com/articles/richardsonMaturityModel.html>

Exemplo de Modelo de Maturidade para Microsserviços



Fonte: <https://www.greymatter.io/blog/the-microservices-maturity-model/>

Exemplo de Modelo de Maturidade para Microsserviços

	Stage 0: Monolithic	Stage 1: Cloud-Native	Stage 2: Service Mesh	Stage 3: Application Networking
Team Structure	Siloed Individuals or Teams	Architecture Review and Integration	Cross-functional Business Teams	Enterprise Platform Engineering Team
Application	Monolithic	A monolith and microservice hybrid	Cloud-native, loosely-coupled small services	Extremely small, single-purpose services
Database	Multiple, external, RDBMS, NOSQL, search indexes	Monolith, virtualized, and data-lake data sources	Multiple, integrated with individual services	Federated fabric contracted as a data mesh from any source
Infrastructure	Bare metal in physical data center	Virtualized Data Center	Public/Private Cloud	On-Premise, Multi-Cloud and Hybrid-Cloud
Development	Waterfall, Yearly/Quarterly	Agile & Continuous Integration, Monthly/Bi-weekly	CI/CD, Weekly	Continuous Delivery, Daily, Hourly
Security	Traditional Perimeter-Based Application Network Security	Authentication and Authorization	mTLS Encryption, Role-based Access Control	Zero-Trust Security with Micro-segmentation
Scalability	Vertical	Combination of monolith and microservice	Vertical and Horizontal Auto-scaling	On-Demand horizontal cloning / clustering of services
Visibility	OS and Hardware Monitoring	Telemetry, Statistics, and Counters	Observability Metrics into Mesh-Specific Services	Single Pane of Glass Heuristics Across All Environments

Fonte: <https://www.greymatter.io/blog/the-microservices-maturity-model/>

Exemplo: Azure Microservices assessment and readiness

Critério	Critério
Entender as prioridades de negócios	Avaliar o tratamento de transações
Registrar decisões de arquitetura	Avalie seu modelo de desenvolvimento de serviços
Avaliar a composição da equipe	Avaliar sua abordagem de implantação
Utilizar a metodologia dos Doze Fatores	Avalie sua plataforma de hospedagem
Entenda a abordagem de decomposição	Avaliar o monitoramento de serviços
Avaliar a prontidão do DevOps	Avaliar a atribuição de token de correlação
Identificar áreas de negócios que mudam com frequência	Avaliar a necessidade de uma estrutura de microsserviços
Avaliar a prontidão da infraestrutura	Avalie sua abordagem ao teste de aplicativos
Avaliar os ciclos de lançamento	Avaliar a segurança dos microsserviços
Avaliar a comunicação entre serviços	Avaliar como os serviços são expostos aos clientes

Fonte: Adaptado de <https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/microservices-assessment>

EXERCÍCIOS E PESQUISAS

EXEMPLO: Especificação REST API



README Apache-2.0 license

PayPal REST API Specifications

This repository contains the specification files for [PayPal REST APIs](#).

You can try our REST APIs in Postman without a PayPal Developer account. Learn more in our [Postman guide](#).

▶ Run in Postman

<https://github.com/paypal/paypal-rest-api-specifications>

ARTIGO

- How do microservices evolve? An empirical analysis of changes in open-source microservice repositories



Journal of Systems and Software

Volume 204, October 2023, 111788



How do microservices evolve? An empirical
analysis of changes in open-source
microservice repositories

Wesley K.G. Assunção^{a b}    , Jacob Krüger^c   , Sébastien Mosser^{d e}   ,
Sofiane Selaoui^f 

<https://www.sciencedirect.com/science/article/pii/S0164121223001838>

OBRIGADO!

<https://www.linkedin.com/in/josemariacesariojunior/>