FIAP ON

SERVICES ARCHITETURE,
API E MOBILE ARCHITETURE

CONCEITOS E DEFINIÇÃO DAS ARQUITETURAS SOA E MSA

CHRISTIANE DE PAULA REIS

1

LISTA DE FIGURAS

SUMÁRIO

1 MSA (MICROSERVICE ARQUITECTURE) E SOA (SERVICE-ORIENTED ARCHITECTURE)

Neste capítulo você vai entender o que são Microsserviços e o que é a MSA (*MicroService Arquitecture*), porém, você precisa saber previamente sobre Serviços e SOA (*Service-Oriented Architecture*), que são os predecessores.

1.1 SOA e Serviços: o que são?

Como o próprio nome diz, **SOA** – Arquitetura Orientada a Serviços, é um estilo de arquitetura de software que tem como princípio fundamental disponibilizar as funcionalidades de um sistema como um **serviço**. Essas funcionalidades podem ser compartilhadas e reutilizadas entre diversas aplicações.

IMPORTANTE: Serviços são **componentes de software** que são desenvolvidos de tal forma que possam ser facilmente vinculados a outros componentes de software.

Para elucidar alguns conceitos sobre serviços, Thomas Erl, em seu livro “SOA: Princípios do Design de Serviços” (ERL; Thomas, 2009), concretizou oito princípios para canalizar as diretrizes e estratégias que envolvem o desenvolvimento de sistemas em SOA:

1) Serviços são REUTILIZÁVEIS

O serviço deve ser o mais genérico possível para aumentar a probabilidade de reutilização. Um serviço não é propriedade de uma equipe de desenvolvimento, é um ativo da empresa.

2) Serviços compartilham um CONTRATO FORMAL

A comunicação entre requisitante e provedor funciona através de um “contrato” que especifica a funcionalidade do serviço e quais são os parâmetros de entrega e recebimento para uma comunicação bem-sucedida.

3) Serviços possuem BAIXO ACOPLAMENTO

Baixo acoplamento significa baixa dependência, ou seja, os consumidores do serviço não devem ser afetados em caso de descontinuidade, substituição, modificação ou evolução de um serviço específico.

4) Serviços ABSTRAEM A LÓGICA

Serviços devem ser como “caixas pretas”, no sentido de encapsulamento da lógica de negócio, o que simplifica o contrato formal, pois permite futuras evoluções na lógica sem necessidade de alteração do contrato.

5) Serviços são CAPAZES DE SE COMPOR

Compor é uma maneira de reutilizar, ou seja, é possível ter uma solução composta por diversas partes que trabalham em conjunto, minimizando a necessidade de reescrever funcionalidades em diversas partes do sistema.

6) Serviços são AUTÔNOMOS

Um serviço autônomo é capaz de se autogerir e é independente de elementos externos para executar sua lógica.

7) Serviços evitam ALOCAÇÃO DE RECURSOS por longos períodos

Para garantir a disponibilidade do serviço, é preciso evitar informação de Estado. Essa medida também possibilita a reutilização do serviço pelo fato de não onerar a infraestrutura com diversas instâncias.

8) Serviços devem possuir a CAPACIDADE DE SEREM DESCOBERTOS

Serviços devem ser descobertos através de mecanismos que a arquitetura deve prover como Diretórios e Registros, que usam tecnologias UDDI (*Universal Description, Discovery and Integration*) e ebXML (*electronic business XML*). A padronização do contrato formal evita redundância de serviços.

Para melhor atender às necessidades do mercado, SOA surgiu basicamente com o objetivo de ser mais flexível, mais ágil e para prover redução de custos na reutilização de serviços.

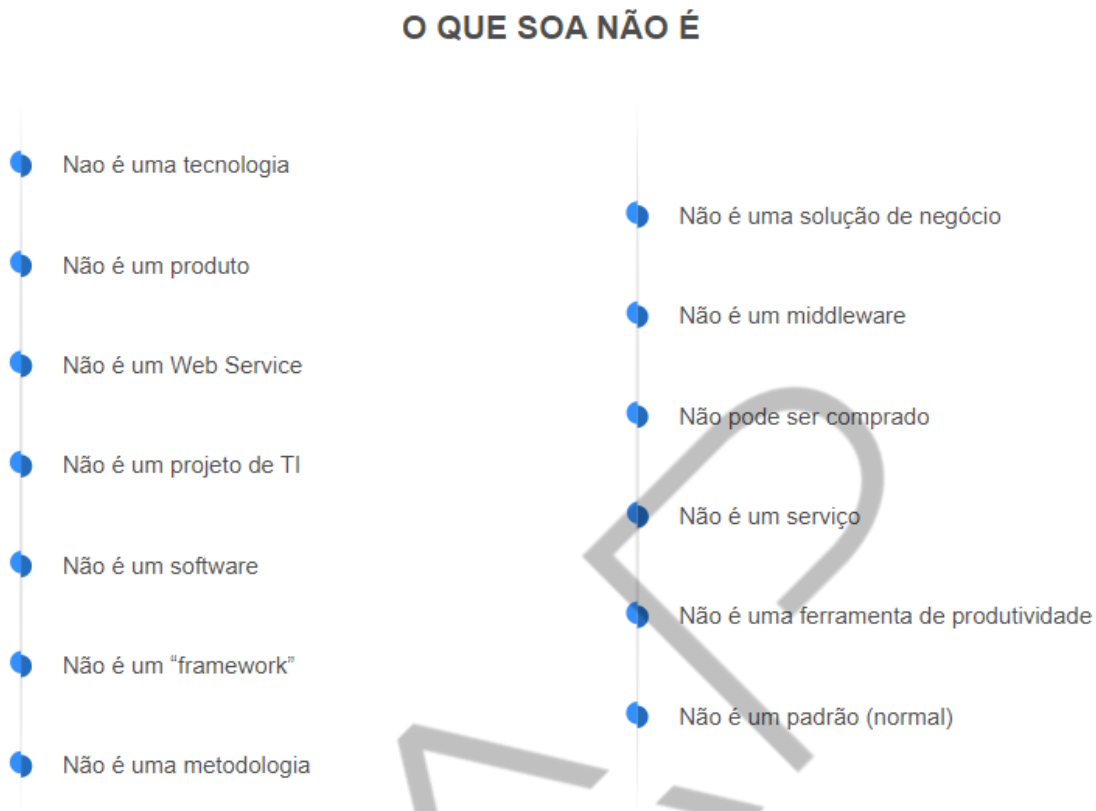

Figura 1.1 – O que NÃO é SOA (parte 1).
Fonte: Elaborado pela autora (2019)

Figura 1.2 – O que NÃO é SOA (parte 2)
Fonte: Elaborado pela autora (2019)

SOA possui um belo pacote de características, não é mesmo? Então, agora que você já sabe as principais características da arquitetura, vamos entender como funciona.

1.1.1 Funcionamento da SOA

Os serviços do sistema são disponibilizados para usuários e outras aplicações, através de um dos componentes mais importantes do SOA, o *Enterprise Service Bus* (ESB). Estamos falando de um barramento de serviços corporativos, que acelera os processos de integração e tem como principais funcionalidades:

- ✓ expor e invocar serviços dos sistemas integrados

- ✓ implementar funções genéricas como autenticação, autorização e *logging*

O ESB ajuda a monitorar, manter e garantir a disponibilidade das aplicações, oferecendo como vantagens:

- ✓ Integração
- ✓ Monitoramento
- ✓ Conversão de protocolos
- ✓ Segurança
- ✓ Roteamento e mediação de mensagens

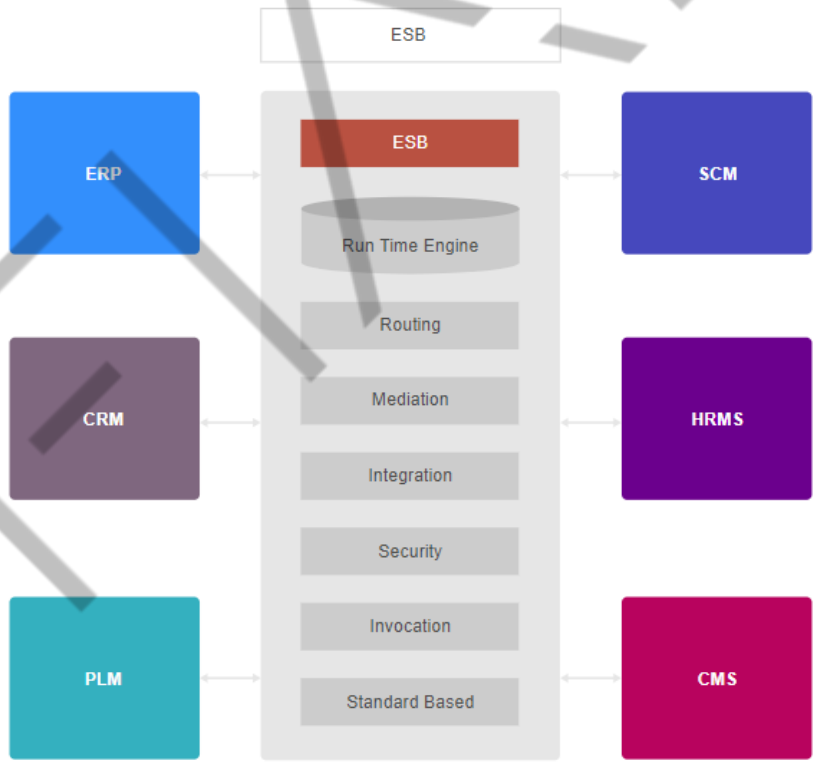

Figura 1.3 – Responsabilidades do ESB em SOA.
Fonte: Adaptado para Fiap (2020)

Porém, a SOA é muito maior que o ESB. Envolve vários componentes que devem ser capazes de se relacionar para prover o melhor resultado para o negócio.

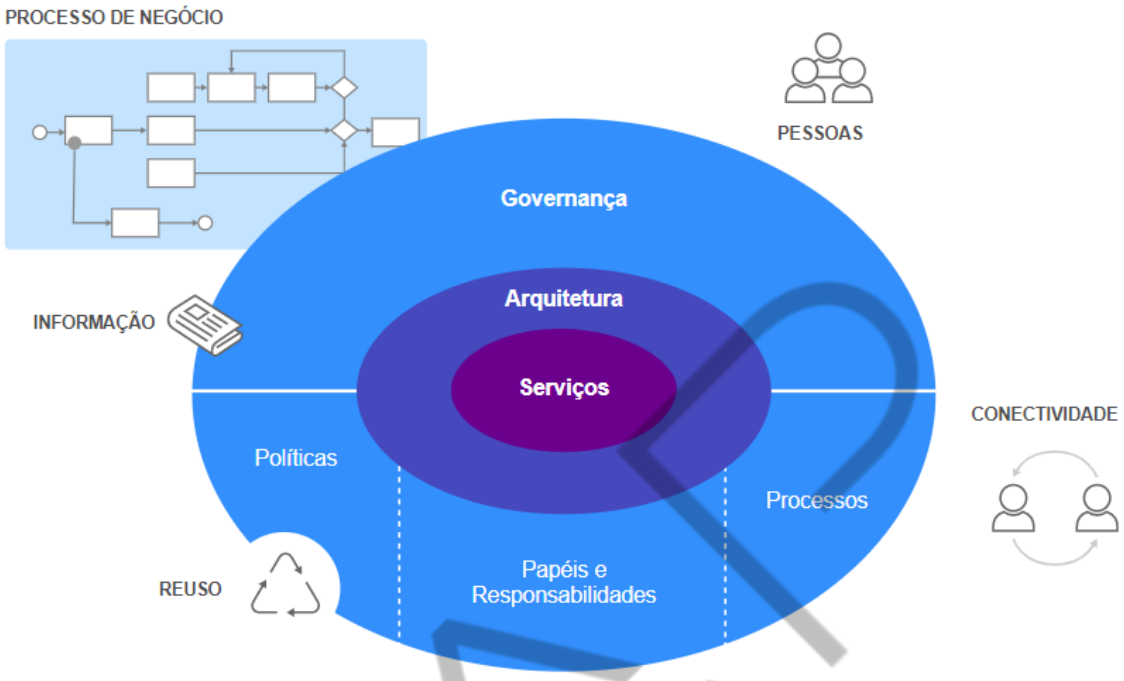

Figura 1.4 – Componentes da SOA.
Fonte: Adaptado para Fiap (2020)

Para obter sucesso na implementação de SOA é necessário ter infraestrutura bem desenvolvida, arquitetura de referência consolidada e política de governança capaz de orquestrar os trabalhos. SOA atingiu seu ápice sofrendo com diversos pontos de implementação, como:

- ✓ **complexidade**: grande quantidade de serviços a ser gerenciada
- ✓ **performance**: dependência do servidor que está alocado e da rede
- ✓ **robustez**: não tem como reverter o processo em caso de exceção
- ✓ **disponibilidade**: todos os serviços ficam indisponíveis se cair a rede ou o servidor
- ✓ **testabilidade**: dificuldade para realizar *debug* no serviço
- ✓ **segurança**: os dados são trafegados pela rede e podem ser interceptados, além de qualquer aplicação pode consumir um serviço que está disponível na rede

Como todas as tecnologias, metodologias e arquiteturas passam por constante evolução, surgiu a MSA, com uma proposta diferenciada da SOA. É o que você vai aprender a partir de agora.

1.2 Como surgiu a MSA - Arquitetura de Microsserviços?

A Arquitetura de Microsserviços (MSA), na sua essência, não é diferente da SOA, foi até definida como uma abordagem refinada de SOA, porque tem as mesmas metas e objetivos. Porém, possui algumas características importantes que diferem o desenvolvimento de aplicações e que nos levam a visualizar a MSA como uma arquitetura escalável.

Na maioria dos casos, serviços desenvolvidos para a estratégia arquitetural SOA são independentes uns dos outros, mas são implementados em uma mesma instância do servidor, compartilhando o mesmo tempo de execução com todos os outros serviços. Assim como as aplicações de software monolíticas, os serviços são incrementados com o tempo e acumulam funcionalidades diversas. Com o passar do tempo, as aplicações da SOA se transformam em blocos monolíticos, o que não as diferem das aplicações monolíticas convencionais (por exemplo: os *ERPs - Enterprise Resource Planning*, traduzindo para português, Planejamento dos Recursos da Empresa).

Algumas características de aplicações baseadas na arquitetura monolítica, incluindo SOA, são:

- aplicações monolíticas são projetadas, desenvolvidas e implantadas como uma única unidade;
- são muito complexas, o que leva um aumento de gastos e aumento de tempo com atualização, manutenção e implementação de novos recursos;
- dificultam a prática de metodologias ágeis de desenvolvimento e entrega (como Agile e DevOps - que enfatizam a velocidade);
- para atualizar uma parte da aplicação, é necessário reimplantar todo o pacote (Ex: aplicações com arquivos WAR de 2 GB) e testar completamente;

- difícil de escalar por ser dimensionada como uma única aplicação com requisitos de recursos conflitantes (Ex: um serviço exige mais memória, enquanto o outro exige mais CPU);
- em caso de instabilidade de um serviço, toda a aplicação pode ser “derrubada”, afetando a confiabilidade;
- difícil praticar inovação de tecnologias ou *frameworks*, visto que todas as funcionalidades são construídas com base em tecnologias ou *frameworks* homogêneos.

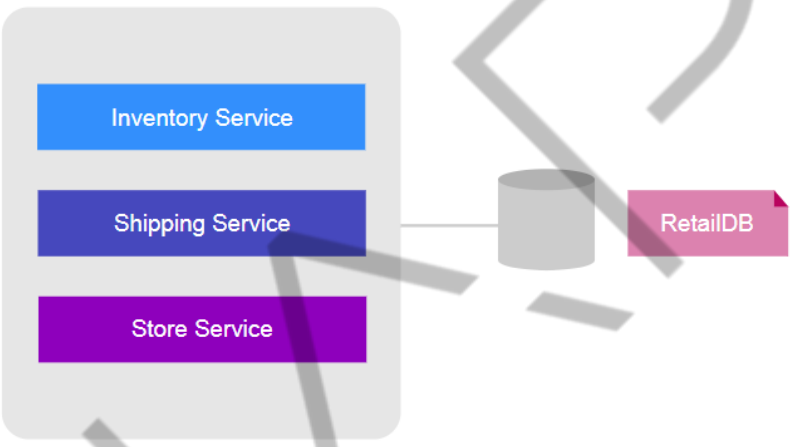

Figura 1.5 – Arquitetura Monolítica.
Fonte: Adaptado para Fiap (2020)

Diante deste cenário, surgiu a MSA, sendo projetada para superar algumas das limitações mencionadas, introduzindo o conceito de "microsserviço". A base da MSA promove o desenvolvimento de uma única aplicação que é composta por um conjunto de serviços pequenos e independentes, que são executados em sua própria instância, e que são desenvolvidos e implantados de forma independente.

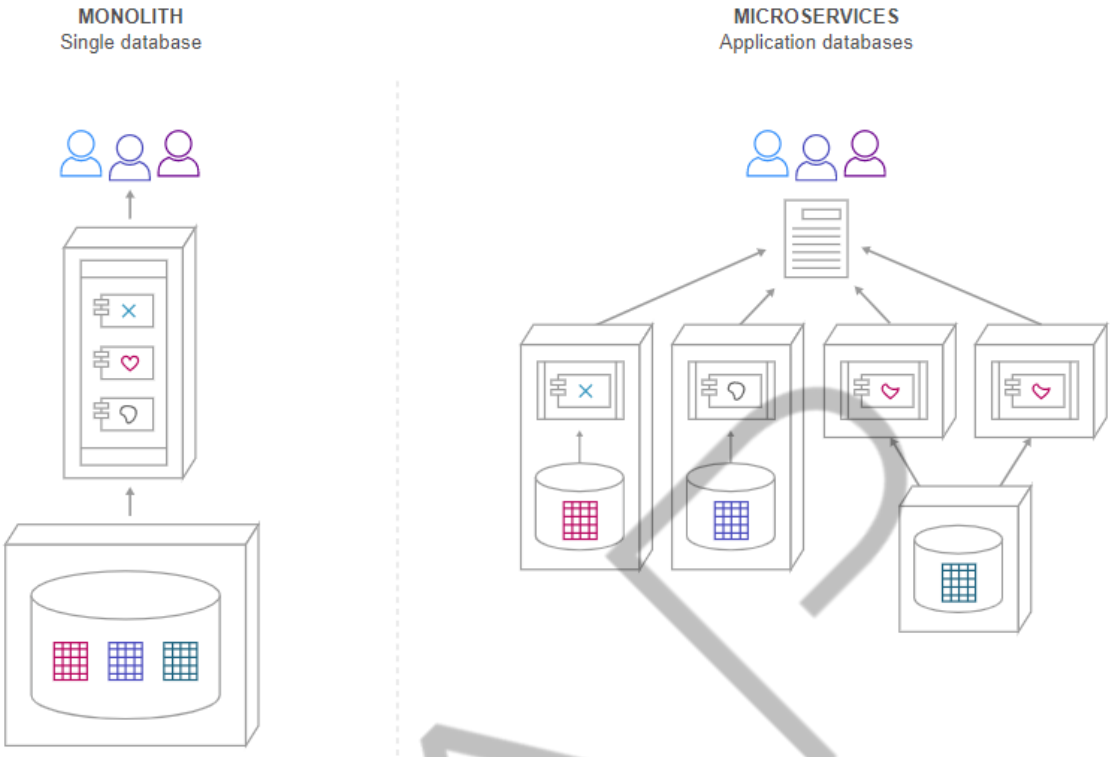

Figura 1.6 – Arquitetura Monolítica x Microsserviços.
Fonte: Martin Fowler – Adaptado para Fiap (2020)

Como exemplo, veja a diferença de um e-commerce implementado seguindo o modelo arquitetural da SOA e outro seguindo o modelo arquitetural da MSA:

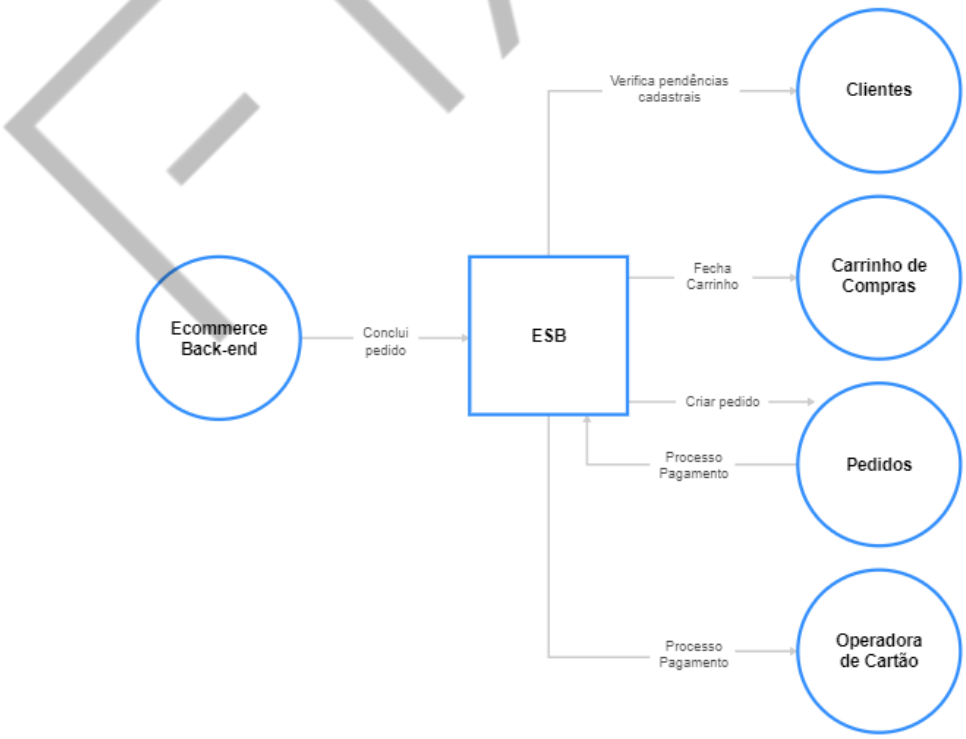

Figura 1.7 – Exemplo de e-commerce em SOA.
Fonte: Adaptado para Fiap (2020)

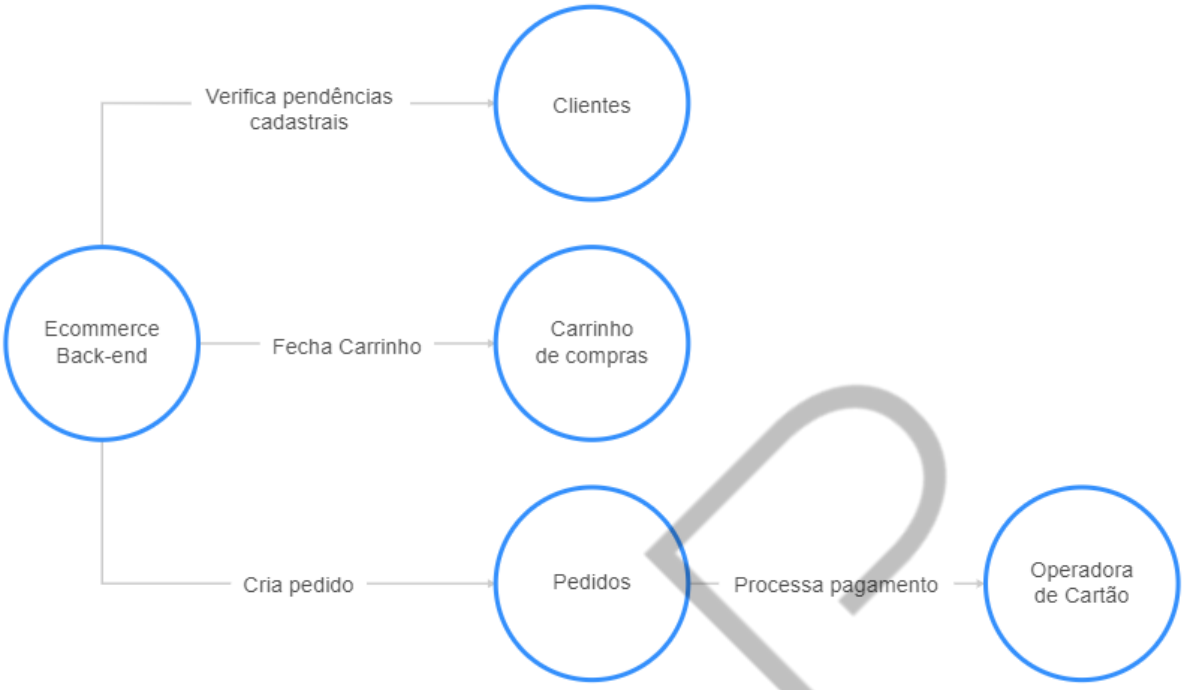

Figura 1.8 – Exemplo de e-commerce em MSA.
Fonte: Adaptado para Fiap (2020)

Por que escolher uma implementação com MSA?

Devemos ser realistas e entender que microsserviços não vão resolver todos os problemas de TI da empresa, principalmente se for implementada uma adaptação cega dos conceitos da MSA.

A resposta pode ser encontrada através da análise das necessidades da aplicação a ser desenvolvida. Se esta aplicação necessita de escalabilidade, como uma aplicação web voltada para o consumidor ou aplicação SaaS (*Software as a Service*), provavelmente a escolha certa é implementar com MSA. Devido a essas e outras necessidades, empresas de grande porte com aplicações complexas, tais como eBay, Netflix, Amazon.com, LinkedIn e Groupon trabalham com a abordagem da MSA, sendo que a Netflix foi a pioneira na implementação desta arquitetura.

Algumas das diversas vantagens que uma arquitetura de microsserviços pode oferecer são:

- baixo acoplamento.
- Reusabilidade.
- aplicações distribuídas na nuvem.
- agilidade para o negócio.

Também é importante citar os pontos de atenção ao optar pela MSA:

- as aplicações são mais complexas e constituídas por mais elementos.
- exige um alto nível de automação, como um PaaS (*Platform as a Service*).
- apresenta complexidade no gerenciamento de dados distribuídos.

Vamos, então, entender os conceitos de Microsserviços e da Arquitetura de Microsserviços (MSA).

1.2.1 MSA e Microsserviços: o que são?

De acordo com Kim (2019), o estilo da **MSA** é uma abordagem para desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um rodando em seu próprio processo e comunicando-se com mecanismos leves, geralmente uma API (*Application Programming Interface*) de recursos HTTP (*Hiper Text Transfer Protocol*).

> The microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. (KIM; BAEK, 2019).

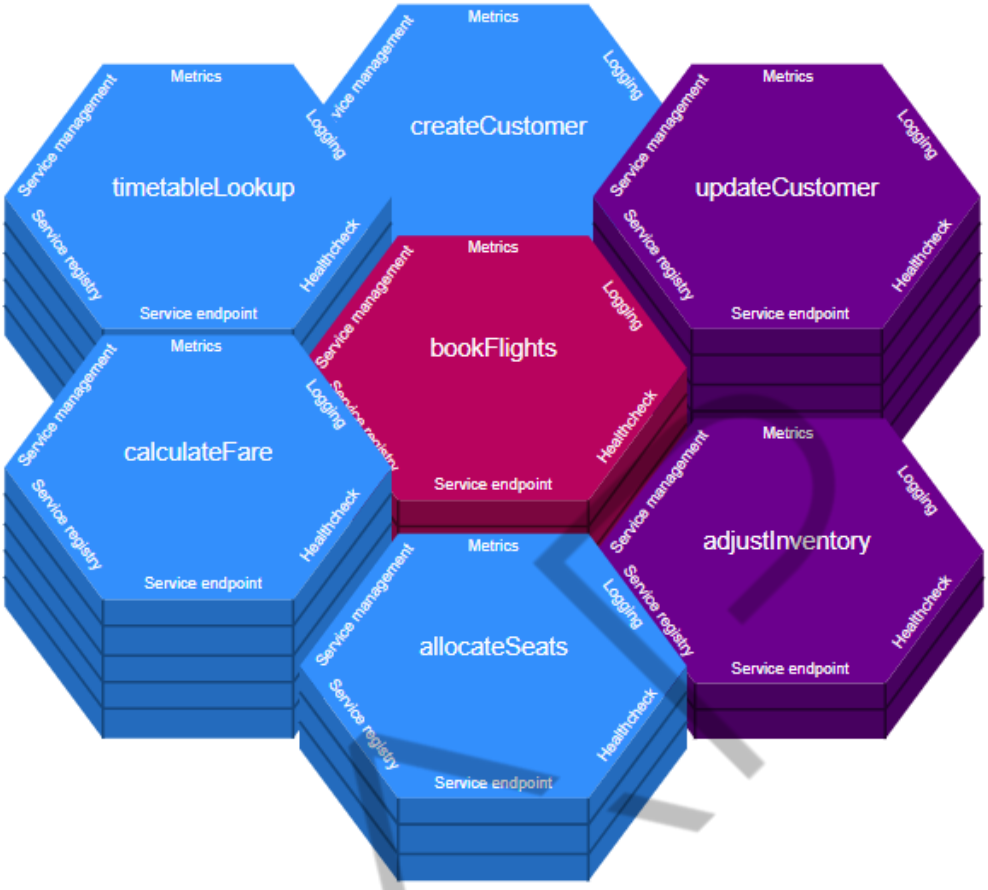

Figura 1.9 – Exemplo funcionamento de Microsserviços
Fonte: Adaptado para Fiap (2020)

Nas palavras de Newman, **Microsserviços** não significa poucas linhas de código, significa “pequenos serviços que funcionam de forma independente e que trabalham juntos”.

> “Microservices are small autonomous services that work together”. (Sam Newman)

Desenvolver um microsserviço requer pouco conhecimento do negócio como um todo, visto que será desenvolvido partes/funcionalidades específicas para a aplicação. Indo mais a fundo, as características e habilidades que um microsserviço deve ter são:

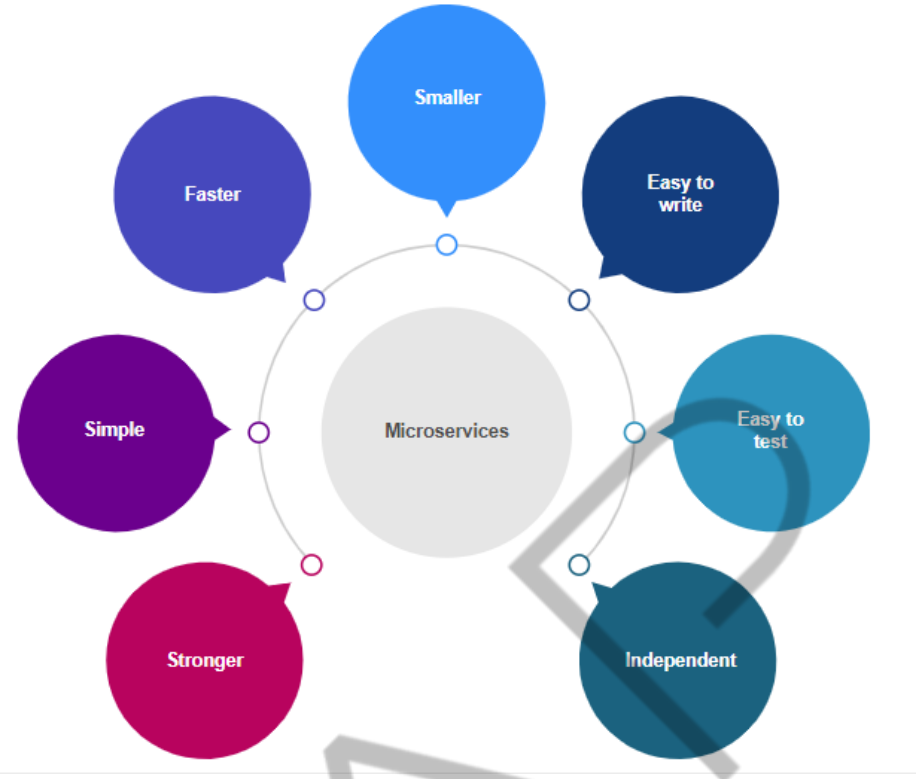

Figura 1.10 – Definição de Microsserviços.
Fonte: Adaptado para Fiap (2020)

Este é o pacote de funcionalidades que possibilitou a ascensão desse tipo de desenvolvimento de aplicações dentro das empresas, o desenvolvimento baseado em microsserviços. Em resumo, microsserviços (ou *microservices*) devem ser:

- ✓ **stronger**: “mais forte” porque promove um escalonamento independente
- ✓ **simple**: pequeno conjunto de responsabilidades
- ✓ **faster**: rápido para executar
- ✓ **smaller**: possui serviços ou funções pequenas
- ✓ **easy to write**: facilidade de escrever e alterar
- ✓ **easy to test**: facilidade para testar de forma independente
- ✓ **independent**: possui sua própria fonte de dados, seu próprio processo de *deploy* e sua própria máquina (ou container)

Porém, custos e benefícios devem ser considerados na tomada de decisão sobre mudar os rumos “tecnológicos” da empresa e adotar ou não uma nova

arquitetura, qualquer que seja. Esses são os pontos importantes que vamos abordar em seguida.

1.2.3 Custo x Benefício de implementar a MSA

É importante ter em mente que a implementação da MSA não se difere da implementação de outras arquiteturas, tecnologias ou metodologias de software relativamente novas. Se a empresa for criar uma MSA ou implantar microsserviços, com certeza os custos iniciais serão mais altos, visto que é necessário ter ambiente adequado e equipe devidamente treinada.

Ao se pensar em uma verdadeira arquitetura de microsserviços, devemos frisar que esta arquitetura veio com o propósito de reduzir tempo de desenvolvimento e dinheiro gastos com manutenção de aplicações e, uma forma de conseguir atingir esse objetivo, é pensar em uma arquitetura de microsserviços que faz uso de APIs simples e que desenvolve sistemas com menor quantidade possível de dependências de aplicações. Porém, é importante destacar que, à medida em que as organizações crescem, elas precisam estar atentas ao custo fixo de cada serviço.

Figura 1.11 – Custo x Benefício de implementar a MSA.
Fonte: Google Imagens (2019)

Para se obter maior desempenho e melhor administração da arquitetura, o ideal é estruturar equipes pequenas e multifuncionais com organização flexível para trabalhar em projetos independentes, provendo condições para que os

desenvolvedores aumentem a velocidade de desenvolvimento, sempre prezando pela qualidade. Werner Vogels, CTO da Amazon, descreveu esta situação com a frase "você constrói, você o executa".

Pensando em adotar o modelo arquitetural de microsserviços, é preciso quebrar alguns paradigmas dentro da estrutura e da cultura da empresa. A Lei de Conway diz que “qualquer organização que projete um sistema (definido de maneira ampla) produzirá um design cuja estrutura é uma cópia da estrutura de comunicação da organização.”

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure. - Conway’s Law

- a empresa deve se reconfigurar em torno de serviços compartilhados.
- as equipes devem ser totalmente *servicebased* (baseada em serviços).
- uma arquitetura de microsserviços bem definida complementa uma estrutura Agile ou DevOps.
- o seu microsserviço deve ser desenhado para ser uma engrenagem de uma máquina gigante.

IMPORTANTE: É preciso quebrar alguns paradigmas dentro da estrutura e da cultura da empresa.

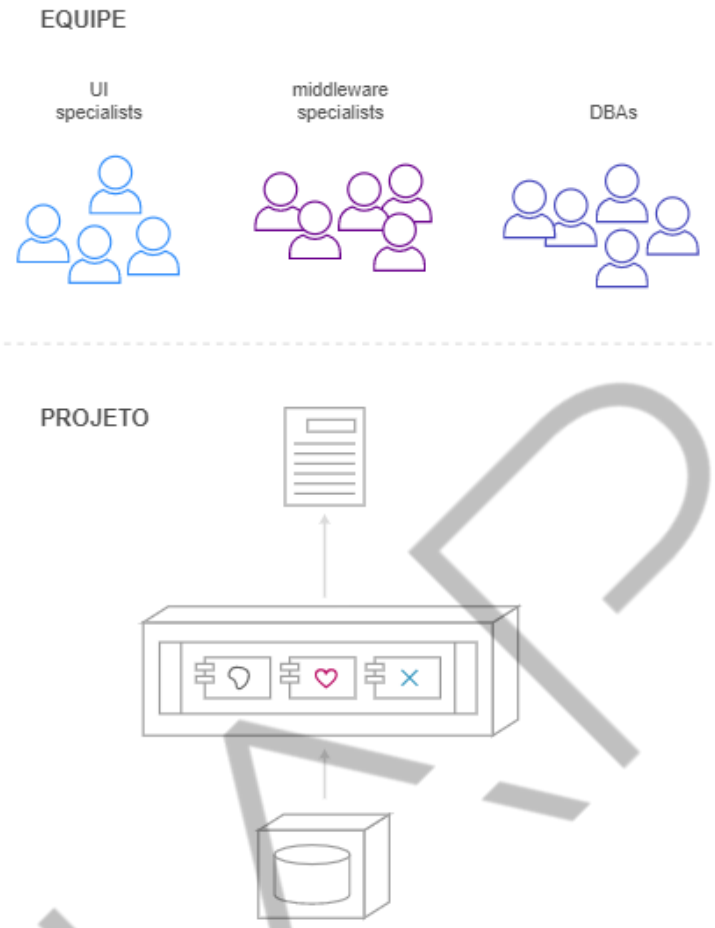

Figura 1.12 – Estrutura da equipe por projeto.
Fonte: Adaptada para Fiap (2020)

Seguindo essas orientações, os benefícios serão significativos, pois os custos iniciais serão pagos em poucos anos, retornando então grandes quantidades de negócios e valor técnico para a empresa. Além de alcançar vantagem competitiva, com a capacidade para refinar uma aplicação de forma mais rápida do que outras empresas que ainda não implementaram uma estratégia de microsserviços. Estima-se um aumento de 90% em serviços entregues por ano ou a capacidade de implementar 20 novos serviços a cada cinco semanas.

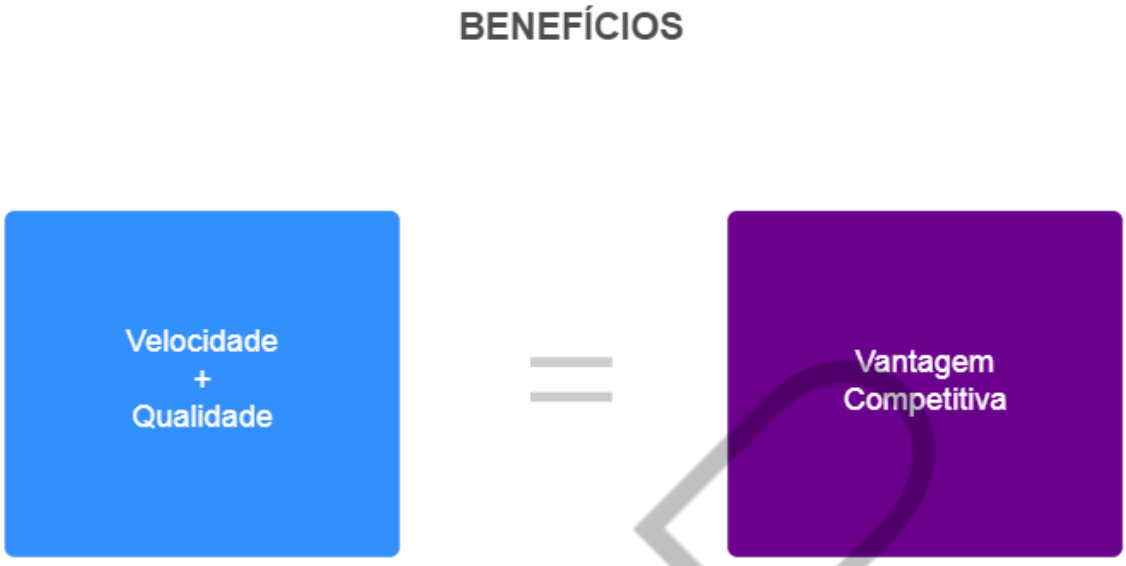

Figura 1.13 – Benefícios da MSA.
Fonte: Elaborado pelo autor (2019)

Agora que você já sabe todas as características de ambas as arquiteturas, SOA e MSA, vamos apenas sintetizar o ciclo de vida de cada uma delas, para entender como estão se comportando no mercado.

1.3 SOA x MSA

Abaixo apresento dois gráficos que foram gerados através do Google Trends, que nos mostram que a Arquitetura Orientada a Serviços – SOA (*Service-Oriented Architecture*), apesar da sua popularidade, se manteve no mercado por um período relativamente curto (mais precisamente entre 2012 e 2015) dando lugar à Arquitetura de Microsserviços – MSA (*MicroService Arquitecture*), que está em uma crescente de utilização desde de 2014, ano em que começou a ganhar popularidade, justamente quando SOA entrou em decadência.

Figura 1.14 – SOA no mundo – Período: 01/01/2005 a 01/01/2016.
Fonte: Google Trends (Maio/2019)

Figura 1.15 – Microservices no mundo – Período: 01/01/2013 a 01/05/2019.
Fonte: Google Trends (Maio/2019)

Através de uma linha do tempo, é possível visualizar a caminhada de ambas arquiteturas.

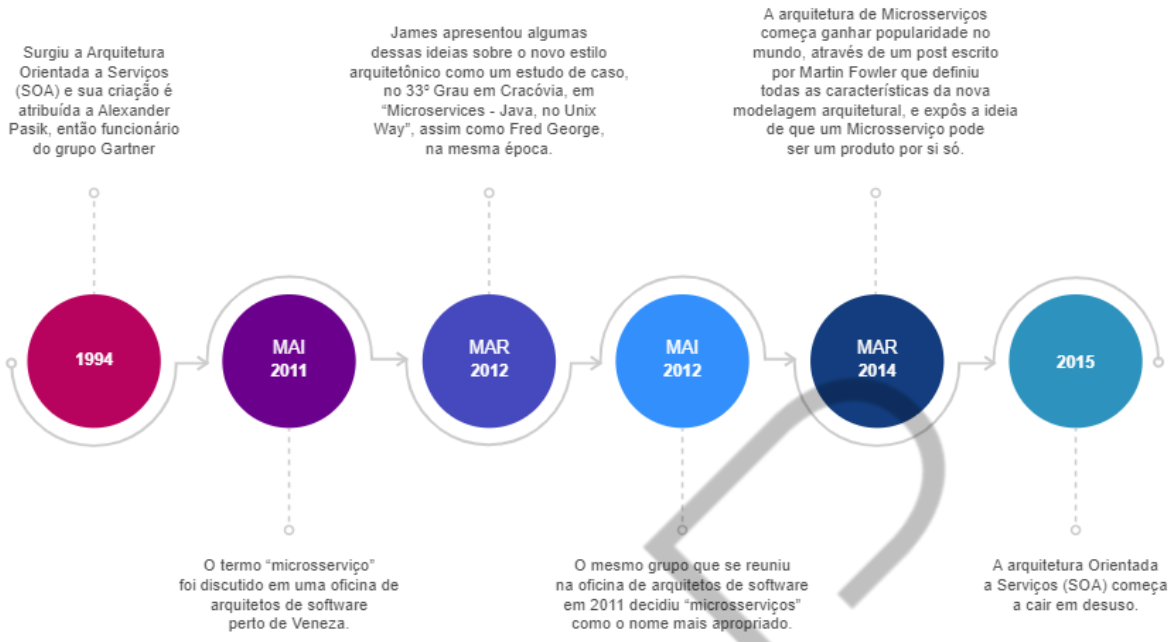

Figura 1.16 – Linha do Tempo SOA x MSA.
Fonte: elaborado por Christiane e FIAP (2019)

Podemos entender SOA sob a ótica de uma perspectiva estratégica, considerando que os princípios, padrões, análise e design do paradigma são orientados a serviços e toda a inteligência fica no barramento, enquanto que a MSA está mais voltada para uma perspectiva tática, considerando que as práticas se aproximam da implementação de serviços, e a inteligência está dentro do serviço em si, os "*endpoints* inteligentes" (*smart endpoints*).

O resultado desses gráficos pode ser explicado pela principal diferença que fez a MSA ganhar mercado e desbancar a SOA, o desacoplamento entre áreas e sistemas. Como Ian Robinson disse: "Seja da web, não por trás da web." Essa foi uma grande mudança na forma como as pessoas construíam aplicações anteriormente, onde a comunicação entre sistemas era baseada em equipes altamente dependentes.

"Be of the web, not behind the web" Ian Robinson

Figura 1.17 – Definições SOA x mas.
Fonte: Adaptada para Fiap (2020)

Em suma, a MSA trouxe a necessidade de quebrar alguns paradigmas dentro da estrutura e da cultura da empresa, e foi percebida com “bons olhos” no mercado, levando ao sucesso na utilização deste modelo arquitetural, que se mantém em foco na resolução de problemas de performance e agilidade para as empresas.

Você chegou ao final deste capítulo muito mais forte para enfrentar o que vem pela frente. No próximo capítulo, vamos abordar sobre algumas formas de implementação de microsserviços.

REFERÊNCIAS

ERL, Thomas. **SOA: Princípios do Design de Serviços.** Prentice Hall, 2009.

FOWLER, Martin. **Microservices.** 2014. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 10 out. 2019.

KIM, Kuinam J., BAEK, Nakhoon**. Information Science and Applications 2018: ICISA 2018.** Singapore: Springer, 2019.

NEWMAN, Sam. **Building Microservices.** O'Reilly Media, 2015. ISBN 978-1491950357.

GLOSSÁRIO

API - Application Programming Interface	API é uma estrutura de códigos de programação padronizados para permitir a conexão entre sistemas.