PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Disciplina: Projeto de Software
Professor: João Paulo Carneiro Aramuni
Aluno: João Pedro Santana Marques

**Martin Fowler - Microsserviços**

**Microsserviços**

A arquitetura de microsserviços surgiu nos últimos anos como uma abordagem para desenvolver aplicações como um conjunto de serviços independentes. Embora não haja uma definição exata, algumas características comuns incluem a organização baseada em capacidades de negócio, implantação automatizada, inteligência nos endpoints e controle descentralizado de linguagens e dados.

Essa abordagem contrasta com o modelo monolítico, no qual a aplicação é construída como uma única unidade, integrando interface, lógica de negócio e banco de dados. Embora monólitos possam ser bem-sucedidos, apresentam desafios, como ciclos de mudança acoplados e dificuldades de escalabilidade, tornando sua manutenção complexa ao longo do tempo.

Os microsserviços oferecem vantagens como implantação e escalabilidade independentes, permitindo que diferentes partes da aplicação sejam escritas em linguagens distintas e gerenciadas por equipes separadas. Essa abordagem se inspira em princípios antigos, como os do Unix, e tem se tornado uma escolha padrão para aplicações empresariais, especialmente na era da computação em nuvem.

**Microsserviços e suas características**

A arquitetura de microsserviços surgiu como alternativa ao modelo monolítico, permitindo que aplicações sejam desenvolvidas como um conjunto de serviços independentes. Cada serviço pode ser implantado e escalado separadamente, além de poder ser desenvolvido em diferentes linguagens e gerenciado por equipes distintas.

**Componentização via serviços**

Sistemas são divididos em componentes independentes (serviços), ao invés de bibliotecas internas.
Serviços comunicam-se por chamadas remotas (ex: APIs REST) e podem ser substituídos ou atualizados sem afetar toda a aplicação.

**Organização por capacidades de negócio**

Em vez de dividir equipes por tecnologia (UI, backend, banco de dados), o foco é em funcionalidades de negócio.
Cada equipe é responsável por um conjunto de serviços que entregam valor diretamente ao usuário.

**Produtos, não projetos**

Ao contrário do modelo tradicional de desenvolvimento baseado em projetos (onde o software é entregue e depois mantido por outra equipe), a abordagem de microsserviços defende que as equipes sejam responsáveis pelo produto durante todo o seu ciclo de vida. Esse conceito, inspirado no modelo da Amazon, aproxima os desenvolvedores da realidade de produção e das necessidades dos usuários.

**Comunicação: smart endpoints e dumb pipes**

Microsserviços evitam a complexidade de ferramentas centralizadas, como os Enterprise Service Buses (ESBs), que frequentemente dificultam a flexibilidade do sistema. Em vez disso, favorecem endpoints inteligentes e canais de comunicação simples (dumb pipes), onde a lógica fica nos serviços e não na infraestrutura de comunicação.

**Governança descentralizada**

A governança centralizada geralmente leva à padronização de plataformas tecnológicas, o que pode ser limitante, pois nem todo problema exige a mesma solução. No modelo descentralizado, ao dividir uma aplicação monolítica em microsserviços, as equipes ganham liberdade para escolher a melhor tecnologia para cada necessidade, como usar Node.js para uma página de relatórios simples ou C++ para componentes de alto desempenho.

Além disso, em vez de impor padrões rígidos, as equipes de microsserviços preferem compartilhar ferramentas úteis, frequentemente adotando práticas de código aberto dentro da empresa, como faz a Netflix. Isso permite a reutilização de soluções testadas sem impedir abordagens alternativas.

**Gerenciamento descentralizado de dados**

A descentralização do gerenciamento de dados se manifesta de diversas formas. No nível conceitual, diferentes sistemas podem ter visões distintas do mundo, como ocorre em grandes empresas: a visão de um cliente no setor de vendas pode ser diferente da visão do suporte, com atributos que variam em significado ou até mesmo não existem em determinados contextos.

**Automação de infraestrutura**

Nos últimos anos, a automação de infraestrutura evoluiu significativamente, impulsionada pelo crescimento da computação em nuvem, especialmente a AWS. Essa evolução reduziu a complexidade operacional envolvida na construção, implantação e manutenção de microsserviços.

Equipes que adotam microsserviços geralmente têm experiência com Continuous Integration (CI) e Continuous Delivery (CD), utilizando extensivamente técnicas de automação. Um dos princípios fundamentais desse modelo é garantir confiança no software por meio de testes automatizados e implantação automatizada em diferentes ambientes.

**Projetado para falhas**

Ao utilizar serviços como componentes, aplicações precisam ser projetadas para tolerar falhas. Como qualquer serviço pode ficar indisponível, os clientes devem lidar com essas falhas da melhor forma possível, o que adiciona complexidade em comparação com sistemas monolíticos. Equipes que trabalham com microsserviços precisam considerar constantemente como as falhas impactam a experiência do usuário.

**Design evolutivo**

A abordagem de design evolutivo é fundamental para o desenvolvimento baseado em microsserviços, pois permite que mudanças sejam implementadas de forma ágil sem comprometer a estabilidade da aplicação. O objetivo não é reduzir as mudanças, mas sim controlá-las e torná-las rápidas e eficientes.

Ao dividir um sistema em componentes, a principal preocupação é garantir que cada parte possa ser substituída ou atualizada sem afetar as demais. Esse princípio leva algumas equipes a projetar serviços com a expectativa de que eles possam ser descartados em vez de evoluídos ao longo do tempo.

**Microsserviços são o futuro?**

O autor acredita que essa abordagem é promissora para aplicações empresariais, citando exemplos de empresas como Amazon, Netflix e The Guardian que já a adotaram. No entanto, ele não afirma com certeza que microsserviços são o futuro das arquiteturas de software, pois ainda é cedo para avaliar seus impactos a longo prazo.

Um dos desafios mencionados é que os efeitos reais das decisões arquiteturais só se tornam evidentes após anos. Enquanto algumas equipes conseguem manter um monólito modular, outras acabam com sistemas difíceis de manter. microsserviços prometem minimizar essa deterioração ao estabelecer limites mais rígidos entre os componentes, mas essa vantagem ainda precisa ser comprovada.

Outro ponto levantado é a dificuldade de definir corretamente os limites dos serviços. Quando os componentes não são bem divididos, a complexidade pode ser apenas transferida para a comunicação entre os serviços, tornando o sistema ainda mais difícil de gerenciar. Além disso, a adoção dessa arquitetura pode ser mais eficiente para equipes experientes, enquanto times menos habilidosos podem acabar criando sistemas ainda mais caóticos.