O artigo de Martin Fowler e James Lewis, "Microservices", apresenta uma ideia que mudou o jeito como muitas empresas e desenvolvedores pensam sobre sistemas. Em vez de construir uma única aplicação enorme e centralizada, é proposto dividir o software em várias partes menores e independentes. Cada uma dessas partes é responsável por uma função específica do negócio e se comunica com as outras através de APIs simples, geralmente usando protocolos leves. Isso facilita bastante a integração e a evolução de cada parte do sistema.

Um ponto importante do texto é a ideia de organizar as equipes de desenvolvimento em torno das necessidades de negócio, e não das camadas técnicas. Isso significa que cada equipe cuida de um conjunto completo de funcionalidades: da interface do usuário ao banco de dados. Essa abordagem dá mais autonomia para as equipes e as aproxima da realidade do que o negócio realmente precisa.

Fowler e Lewis também destacam que cada serviço deve ser visto como um produto, e não como um projeto temporário. Isso significa que a equipe é responsável por todo o ciclo de vida do software, incluindo a operação no dia a dia. A arquitetura de microsserviços valoriza a ideia de ter "endpoints inteligentes e pipes burros": a lógica complexa fica nos próprios serviços, enquanto a comunicação entre eles é mantida o mais simples e direta possível, sem a necessidade de ferramentas de integração pesadas.

A descentralização é um pilar fundamental dos microsserviços, trazendo benefícios claros. A liberdade para cada equipe escolher as tecnologias que melhor atendem às suas necessidades, uma prática conhecida como "Polyglot Persistence", permite que um serviço de processamento de imagens use uma linguagem otimizada para performance, enquanto outro de relatórios adote uma linguagem mais rápida para prototipagem. Essa flexibilidade, no entanto, exige lidar com desafios de consistência de dados e comunicação entre os serviços. Para que tudo funcione bem, é essencial investir em automação, integração contínua e monitoramento constante, pois as falhas são inevitáveis e o sistema precisa ser robusto para lidar com elas. O monitoramento em tempo real se torna crucial para identificar problemas de forma proativa e garantir que o sistema continue funcionando mesmo se uma parte dele falhar.

Apesar de todos os pontos positivos, Fowler é bastante realista e adverte que os microsserviços não são uma solução mágica. Ele reconhece que essa arquitetura adiciona uma camada de complexidade significativa, principalmente na comunicação remota, nos testes e na refatoração. Por isso, um dos conselhos mais valiosos do artigo é que muitas empresas de sucesso começaram com um monólito e só depois o dividiram em microsserviços quando o sistema se tornou grande demais para ser gerenciado. A arquitetura de microsserviços exige uma equipe experiente, capaz de lidar com os desafios técnicos e de organização que surgem pelo caminho. Em vez de ser a primeira escolha, a arquitetura de microsserviços deve ser vista como uma estratégia para resolver a complexidade que um sistema monolítico grande demais acaba criando.

Inicialmente, o sistema de cardápio, processamento de pedidos, pagamentos e gestão de entregas, está em um único aplicativo monolítico. Isso funciona bem no começo, mas conforme o negócio cresce e novos restaurantes entram na rede, começam a surgir problemas: cada alteração no cardápio exige um novo deploy de todo o sistema, e qualquer

instabilidade na função de pagamentos pode afetar a visualização do cardápio e até impedir que novos pedidos sejam feitos.

Aplicando a arquitetura de microserviços, a rede poderia separar o sistema em serviços independentes, como Serviço de Cardápio, Serviço de Pedidos e Serviço de Pagamentos. Assim, o cardápio poderia ser atualizado e implantado sem interferir nos pagamentos ou no fluxo de pedidos. Se o serviço de pagamentos sofrer instabilidade, o cliente ainda conseguiria visualizar o cardápio e realizar um pedido, que ficaria pendente até que o pagamento estivesse disponível.