Microservices, por Martin Fowler, discute o conceito de uma arquitetura contrária aos Monólitos, onde toda a camada da aplicação se encontra em um único processo. É um estilo arquitetural que melhor reflete o mundo real e suas divisões de departamentos e áreas representados como serviços, o que resulta em menor acoplamento e maior facilidade de se implementar e realizar o deploy de alterações sem prejudicar o resto da aplicação, aumentando a eficiência operacional. Entretanto, os microsserviços trazem consigo diversos desafios que impactam desde a implementação até a estrutura organizacional de uma empresa.

Microsserviços consiste numa abordagem na arquitetura de software onde o sistema é subdividido em pequenos serviços executados como processos diferentes, cada um dentro de seu contexto delimitado, idealmente apenas com acoplamento de domínio. Isso facilita o deploy da aplicação, de forma que alterações pontuais não exijam o deploy de toda aplicação, mas apenas do serviço alterado, o que também favorece a escalabilidade – pode-se escalar os serviços necessários horizontalmente, ao invés de toda a aplicação – além de aumentar a manutenibilidade de grandes sistemas monolíticos que, em sua maioria, têm imensa dificuldade de manter uma estrutura modular.

Não há uma checklist que determine se um sistema é monolítico ou não, mas há traços comuns na maior parte daqueles que os implementam, embora não sejam regras. Eles incluem os seguintes:

1) **Componentização via serviços:** a definição de componentes no artigo é de uma unidade de código independente, substituível e atualizável. Bibliotecas são blocos de código chamados dentro de um mesmo processo, enquanto serviços são blocos de código em diferentes processos que se comunicam através de mecanismos de chamada remota. O uso de componentes como serviços permite o deploy individual de cada componente; o uso de tecnologias diferentes, que resolvam problemas de forma mais eficiente, dentro do mesmo sistema; e o uso de Published Interfaces mais bem definidas. O principal contraponto seria o custo adicional de chamadas remotas para comunicação inter-processo.
2) **Organização refletida na organização do negócio:** o autor utiliza a Lei de Conway, que determina que a arquitetura final de um sistema é uma reflexão da estrutura organizacional da empresa e como os times se comunicam. No caso de microsserviços, apenas uma equipe multidisciplinar deve ser designada para o desenvolvimento de um serviço e esta mesma equipe pode estar designada para mais de um serviço, de forma a prevenir que eles sejam subdivididos devido às dinâmicas de comunicação entre times. Em sistemas monolíticos, essa divisão se torna extremamente difícil de se fazer, exigindo muito mais disciplina e confiança entre os membros do projeto e sendo muito mais suscetível a erros.
3) **Produtos, não projetos:** microsserviços demandam maior monitoramento por parte dos desenvolvedores, forçando-os a participarem de todo o ciclo de vida do software, ao invés de tratá-lo como um mero projeto a ser finalizado e entregue.
4) **Smart endpoints e dumb pipes:** serviços utilizam protocolos comumente utilizados na web (como HTTP) ou algum tipo de barramento leve considerada “dumb” (cuja única função é o roteamento de mensagens). Em monólitos, os componentes estão no mesmo processo e isso permite a comunicação através de chamadas de métodos e funções. O principal desafio na transição de uma arquitetura monolítica para uma em microsserviços é a transição da comunicação em memória para RPC (remote procedure calls).

5) **Governança descentralizada:** cada serviço pode se adequar às suas necessidades. Times também podem produzir ferramentas que podem ser compartilhadas por pessoas com as mesmas dificuldades.
6) **Gerenciamento de dados descentralizado:** este princípio se manifesta de diversas maneiras diferentes. Uma delas se refere ao uso de bancos de dados dedicados à serviços e contextos delimitados específicos, ao invés de bancos utilizados por todos, o que ajuda a conservar a linguagem ubíqua. Um problema é a coordenação de transações distribuídas, o que causa o maior uso de coordenação sem transação de serviços.
7) **Automação de infraestrutura:** o uso de automação e CI/CD facilita facilita, especialmente, o deploy e monitoramento de microsserviços.
8) **Design para falha:** ao contrário das aplicações monolíticas, em que uma falha em algum componente representa uma falha no todo, as aplicações de um sistema de microsserviços exige o tratamento de erros para cada serviço, aumentando a sua complexidade devido ao extenso error handling. Isso também coloca uma ênfase ainda maior na necessidade de monitoramento em tempo real.
9) **Design evolucionário:** microsserviços permitem que mudanças num dado serviço sejam independentes dos outros, permitindo mudanças mais frequentes e seguras. Problemas em um dado serviço ou até mesmo a sua destruição não devem afetar o sistema como um todo. Se dois serviços precisam ser atualizados simultaneamente para funcionarem, então eles provavelmente deveriam ser um único serviço. Esses fatores também permitem melhor planejamento de releases mais granulares.