A arquitetura de microservices descreve uma maneira de abordar a construção do software de forma modular, ou seja, cada serviço será independente dentro de seu escopo, em torno de uma capacidade de negócio, podendo ter, em uma mesma aplicação, diferentes serviços em diferentes linguagens e tecnologias de armazenamento. Dessa forma, ainda que a maioria dos sistemas empresariais sejam desenvolvidas no modelo monolítico, indo contra a ideia de microserviços, tal abordagem traz diversos desafios, como a recompilação e redeploy para qualquer mudança, a dificuldade de manter-se a modularidade, sendo difícil isolar alterações, e a incapacidade de escalar uma parte específica do sistema, tendo que levar a aplicação como um todo junto.

Assim sendo, ainda que não exista uma definição concreta do que seja a arquitetura de microserviços, algumas características são frequentemente observada nesses sistemas, mesmo que não sejam presentes em todas, está contida na maioria, sendo elas:

- Componentização por Serviços:

A indústria de software sempre teve o anseio por construir sistemas a partir de códigos e componentes reutilizáveis, e isso está presente na arquitetura de microserviços, onde temos bibliotecas e serviços, tendo interfaces explícitas de implementação, evitando dependências excessivas entre componentes, e valorizando o ponto principal de independência e eficiência de cada serviço.

- Organização por capacidades de negócio:

É muito comum no mercado de tecnologia dividirem as equipes por camadas de tecnologia, como o time de design, back-end, front-end, infra e relacionados, porém essa estratégia pode gerar ineficiência, pois até as mudanças mais simples exigem a colaboração de múltiplos times, tornando o processo de desenvolvimento mais lento. Dessa forma, a arquitetura propõe uma divisão em capacidades de negócio, onde cada serviço cobre um domínio específico da empresa, e a equipe inclui tudo que precisa, combinando pessoas de diferentes áreas, para lidar com aquele serviço.

Além da diferença organizacional, a visão sobre o que está sendo desenvolvido é diferente do tradicional, onde o software é um projeto que "termina após a entrega", abordando um time responsável pelo software durante a sua vida útil, tendo um sistema iterativo e incremental.

Esse modelo de desenvolvimento traz consigo a cultura do "You build it, you run it", onde as equipes que desenvolvem um serviço também são responsáveis por sua operação e manutenção. Isso evita o problema recorrente da separação entre times de desenvolvimento e suporte, garantindo que quem constrói o sistema tenha total conhecimento sobre ele e possa resolver falhas mais rapidamente.

Outro ponto fundamental na arquitetura de microservices é a governança descentralizada. Diferente dos sistemas monolíticos tradicionais, onde há um controle rígido sobre padrões tecnológicos, microservices permitem que cada equipe escolha a tecnologia mais adequada para seu serviço. Essa liberdade, no entanto, não significa falta de organização: boas práticas como a reutilização de

bibliotecas internas e a definição de contratos de serviço garantem que os sistemas continuem interoperáveis e fáceis de manter.

Ainda, como os microservices permitem o uso de diferentes tecnologias dentro da mesma aplicação, é essencial adotar estratégias para manter a compatibilidade entre serviços. Técnicas como Tolerant Reader e Consumer-Driven Contracts ajudam a garantir que os sistemas possam evoluir sem quebrar integrações existentes, permitindo mudanças graduais e mais seguras.

Por fim, a adoção de microservices exige um modelo de monitoramento e automação robusto. Como cada serviço é independente e pode estar rodando em diferentes ambientes, ferramentas de observabilidade, logging e deploy contínuo são fundamentais para garantir estabilidade e rastreamento eficiente de erros. Empresas como Netflix e Amazon utilizam essas práticas para manter seus serviços altamente disponíveis e resilientes.

Dessa forma, a arquitetura de microservices não é apenas uma questão técnica, mas também organizacional. Para funcionar bem, ela precisa de times capacitados e processos bem definidos, garantindo que a flexibilidade e escalabilidade prometidas realmente se traduzam em benefícios concretos para o negócio.

Além disso, a descentralização da gestão de dados nos microservices impacta diretamente na maneira como as empresas lidam com a segurança e conformidade dos dados. Em um modelo monolítico, a aplicação geralmente conta com uma única camada de controle de acesso e auditoria centralizada, facilitando a aplicação de políticas de segurança. No entanto, em uma arquitetura de microservices, cada serviço pode ter seu próprio mecanismo de autenticação, autorização e registro de atividades, o que pode levar a desafios na implementação de conformidade regulatória, como GDPR ou LGPD. Para lidar com isso, as empresas precisam adotar abordagens padronizadas, como autenticação federada via OpenID Connect, além de estratégias centralizadas de monitoramento e rastreamento para garantir que os serviços sigam as diretrizes de segurança exigidas.

Outro ponto relevante é o impacto da descentralização no desempenho e escalabilidade do sistema. Quando cada microservice gerencia seu próprio armazenamento de dados, é possível otimizar a tecnologia de banco de dados para atender melhor às necessidades específicas do serviço, permitindo escolhas como bancos NoSQL para alta escalabilidade ou bancos relacionais para integridade transacional. No entanto, essa flexibilidade também aumenta a complexidade operacional, exigindo soluções eficientes para gerenciamento de cache distribuído, replicação de dados e balanceamento de carga. Estratégias como Event Sourcing e CQRS (Command Query Responsibility Segregation) são frequentemente adotadas para lidar com esses desafios, garantindo que as operações de leitura e escrita possam ser otimizadas sem comprometer a integridade ou a consistência dos dados entre os serviços.

A automação de infraestrutura evoluiu significativamente nos últimos anos, especialmente com o crescimento da computação em nuvem e da AWS, reduzindo a complexidade operacional no desenvolvimento e operação de microserviços. Equipes experientes em Continuous Delivery e Continuous Integration fazem amplo uso de automação de infraestrutura, garantindo testes

automatizados e implantação contínua para cada ambiente, conforme ilustrado na pipeline de build.

A automação facilita a adoção de boas práticas, resultando em ferramentas úteis para desenvolvedores e equipes de operações, como a criação de artefatos e gerenciamento de código. Exemplos incluem as ferramentas open-source da Netflix e o Dropwizard. Uma vez que a automação da entrega está bem estabelecida para um monólito, implantar múltiplas aplicações se torna uma tarefa simples e previsível, alinhada ao objetivo do Continuous Delivery de tornar o deployment um processo "entediante".

No entanto, operar microserviços em produção pode ser significativamente diferente de um monólito, exigindo uma abordagem mais sofisticada para monitoramento, resiliência e gerenciamento de falhas.

Aplicações baseadas em microserviços devem ser projetadas para tolerar falhas, pois qualquer chamada a um serviço pode não responder. Estratégias como o uso do Simian Army da Netflix ajudam a testar a resiliência das aplicações.

Padrões como Circuit Breaker, Bulkhead e Timeout são fundamentais para evitar falhas em cascata. A Netflix utiliza esses padrões para garantir a estabilidade dos seus sistemas. Monitoramento em tempo real é essencial para detectar falhas rapidamente e restaurar o serviço automaticamente, acompanhando métricas técnicas e de negócio para identificar problemas emergentes.

Chamadas síncronas entre serviços podem amplificar o impacto do downtime. Algumas empresas limitam essas chamadas por requisição, enquanto outras adotam designs assíncronos para melhorar a resiliência.

A transparência no monitoramento de microserviços é crucial, com dashboards detalhados mostrando status, throughput, latência e outros indicadores operacionais. Equipes que adotam microserviços geralmente vêm de um histórico de design evolutivo, permitindo mudanças rápidas e controladas sem desacelerar o desenvolvimento.

A modularidade orientada à mudança é essencial para definir a divisão de serviços, garantindo que componentes frequentemente alterados juntos permaneçam no mesmo módulo. A abordagem do Guardian demonstra uma transição gradual do monólito para microserviços, onde novos recursos são desenvolvidos separadamente e consomem a API do monólito.

A granularidade dos microserviços permite liberar mudanças rapidamente sem a necessidade de um deploy completo da aplicação, embora isso exija atenção para evitar que alterações quebrem dependências. Em vez de versionamento rigoroso, a preferência na arquitetura de microserviços é projetar serviços que sejam tolerantes a mudanças nos seus fornecedores, minimizando impactos na integração.