semana 8- aula 01
Linguagens de programação back-end
Arquitetura de aplicações back-end
Arquitetura monolítica vs microsserviços
Código da aula: [SIS]ANO2C2B2S8A1

Objetivos da Aula:

- ❖ Diferenciar entre arquitetura monolítica e microsserviços.

Exposição

Análise Comparativa: Arquitetura Monolítica vs. Microsserviços

A escolha da arquitetura de software é uma decisão crucial que impacta a escalabilidade, manutenibilidade, flexibilidade e velocidade de desenvolvimento de uma aplicação. As arquiteturas monolítica e de microsserviços representam abordagens distintas para a construção de sistemas, cada uma com suas próprias vantagens e desvantagens.

Arquitetura Monolítica

Em uma arquitetura monolítica, todos os componentes de uma aplicação (interface do usuário, lógica de negócios, acesso a dados) são construídos e implantados como uma única unidade. Imagine um grande bloco de construção onde todas as partes estão interligadas e dependem umas das outras.

Exemplos:

- Aplicações web tradicionais: Muitas aplicações web construídas com frameworks como Ruby on Rails, Django (Python) ou Spring Boot (Java) podem seguir uma arquitetura monolítica, onde o front-end e o back-end residem em um único código-fonte e são implantados juntos.
- Sistemas de gestão empresarial (ERP): Alguns sistemas ERP legados, que abrangem diversas funcionalidades como contabilidade, recursos humanos e gestão de estoque, podem ser monolíticos.
- Plataformas de e-commerce iniciais: Plataformas de comércio eletrônico que integram catálogo de produtos, carrinho de compras, processamento de pagamentos e gestão de pedidos em uma única aplicação.

Vantagens:

- Desenvolvimento inicial mais simples: Para aplicações menores e com equipes reduzidas, a arquitetura monolítica pode ser mais rápida de

desenvolver inicialmente, pois há menos complexidade na comunicação e implantação.
- Implantação facilitada: Geralmente, a implantação envolve copiar um único pacote (por exemplo, um arquivo WAR ou JAR) para um servidor.
- Depuração e testes mais fáceis: Depurar e testar podem ser mais diretos, pois todo o código reside em um único lugar.
- Menor latência em chamadas internas: A comunicação entre os componentes dentro do mesmo monólito é geralmente mais rápida, pois não envolve a rede.

Desvantagens:

- Escalabilidade limitada: Escalar a aplicação significa escalar toda a unidade, mesmo que apenas um componente específico esteja sob carga. Isso pode levar ao desperdício de recursos.
- Dificuldade em adotar novas tecnologias: Alterar a tecnologia ou linguagem de programação de um componente pode ser complexo e arriscado, exigindo a reescrita ou modificação de grande parte da aplicação.
- Implantações mais arriscadas: Qualquer alteração, mesmo em uma pequena parte do código, requer a reimplantação de toda a aplicação, aumentando o risco de introduzir problemas em outras áreas.
- Menor resiliência: Se um componente falhar, pode afetar toda a aplicação.
- Desenvolvimento mais lento com o tempo: À medida que a aplicação cresce, a complexidade do código aumenta, tornando o desenvolvimento, teste e manutenção mais difíceis e lentos.
- Acoplamento forte: Os componentes são fortemente acoplados, o que dificulta a reutilização de código e a autonomia das equipes.

Arquitetura de Microsserviços

Em contraste, uma arquitetura de microsserviços estrutura uma aplicação como uma coleção de pequenos serviços independentes. Cada serviço é responsável por uma funcionalidade de negócio específica e se comunica com outros serviços através de APIs bem definidas (geralmente usando protocolos leves como HTTP/REST ou gRPC). Pense em vários blocos de construção menores e independentes que trabalham juntos para formar a aplicação.

Exemplos:

- Plataformas de streaming de vídeo (Netflix): A Netflix utiliza uma arquitetura de microsserviços onde diferentes funcionalidades como recomendação de vídeos, gerenciamento de usuários, faturamento e streaming são tratadas por serviços independentes.
- Plataformas de e-commerce modernas (Amazon): A Amazon possui uma arquitetura complexa de microsserviços para gerenciar seu vasto catálogo de

produtos, sistema de pedidos, logística, recomendações e muitos outros aspectos.
- Aplicações de redes sociais (Twitter): O Twitter também utiliza microsserviços para lidar com diferentes partes de sua plataforma, como a timeline, mensagens diretas e feeds.

Vantagens:

- Escalabilidade granular: Cada serviço pode ser escalado independentemente com base em suas necessidades de carga, otimizando o uso de recursos.
- Flexibilidade tecnológica: Diferentes serviços podem ser construídos usando diferentes tecnologias e linguagens de programação, permitindo que cada equipe escolha a melhor ferramenta para sua necessidade.
- Implantações mais frequentes e menos arriscadas: Pequenas alterações em um serviço podem ser implantadas independentemente, reduzindo o risco de afetar outras partes da aplicação.
- Maior resiliência: Se um serviço falhar, os outros serviços podem continuar funcionando (desde que haja mecanismos de tratamento de falhas adequados).
- Desenvolvimento paralelo e mais rápido: Equipes menores e focadas podem trabalhar em serviços específicos de forma independente, acelerando o desenvolvimento.
- Reutilização de serviços: Serviços podem ser reutilizados por diferentes partes da aplicação ou até mesmo por outras aplicações.
- Melhor isolamento de falhas: Falhas em um serviço são isoladas e têm menor probabilidade de derrubar toda a aplicação.

Desvantagens:

- Maior complexidade: A arquitetura distribuída introduz complexidade em termos de comunicação entre serviços, gerenciamento de rede, tratamento de falhas distribuídas e monitoramento.
- Desenvolvimento mais complexo: Desenvolver, testar e depurar aplicações distribuídas pode ser mais desafiador.
- Latência de rede: A comunicação entre serviços através da rede pode introduzir latência.
- Gerenciamento distribuído de transações: Garantir a consistência de dados em transações que envolvem múltiplos serviços pode ser complexo (necessidade de usar padrões como Sagas).
- Sobrecarga operacional: Gerenciar um grande número de serviços independentes requer mais esforço em termos de infraestrutura, implantação e monitoramento.
- Testes de integração mais complexos: Testar a interação entre os diferentes serviços é mais desafiador do que em um monólito.

Conclusão

A escolha entre arquitetura monolítica e microsserviços depende das necessidades específicas do projeto, do tamanho da equipe, da complexidade da aplicação e dos requisitos de escalabilidade e manutenção.

- Arquiteturas monolíticas podem ser adequadas para aplicações menores, com requisitos de escalabilidade modestos e equipes com menos experiência em sistemas distribuídos. Elas oferecem simplicidade inicial e facilidade de implantação em cenários mais simples.
- Arquiteturas de microsserviços são mais adequadas para aplicações grandes e complexas, com requisitos de alta escalabilidade, equipes distribuídas e a necessidade de adotar diferentes tecnologias. Elas oferecem maior flexibilidade, resiliência e agilidade no desenvolvimento, mas introduzem maior complexidade operacional.

Em muitos casos, pode-se começar com uma arquitetura monolítica e, à medida que a aplicação cresce e se torna mais complexa, migrar gradualmente para uma arquitetura de microsserviços (padrão Strangler Fig). Não existe uma solução única, e a decisão deve ser tomada com base em uma análise cuidadosa dos trade-offs envolvidos.

Design Patterns em Arquiteturas

Monolítica:

- Singleton: Garante uma única instância de uma classe para gerenciar recursos compartilhados globalmente dentro da aplicação.
- Factory Method: Desacopla a criação de objetos do código principal, permitindo flexibilidade na escolha da classe a ser instanciada.

Microsserviços:

- Service Registry: Centraliza o gerenciamento e a descoberta dos serviços e suas localizações, facilitando a comunicação entre eles.
- Circuit Breaker: Aumenta a resiliência da aplicação, interrompendo a comunicação com serviços com falhas para evitar que a falha se propague.