Semana 18 - Aula 1

Tópico Principal da Aula: Frameworks back-end e integração com serviços externos

Subtítulo/Tema Específico: Desenvolvimento de microsserviços e mensageria

Código da aula: [SIS]ANO2C2B3S18A1

Objetivos da Aula:

- Projetar e desenvolver microsserviços utilizando frameworks back-end.
- Compreender a divisão de uma aplicação em microsserviços e seus benefícios.
- Explorar formas de comunicação entre microsserviços, avaliando suas vantagens e desvantagens.
- Identificar desafios e soluções para falhas na comunicação entre serviços.

Recursos Adicionais (Sugestão, pode ser adaptado):

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.

Exposição do Conteúdo:

Referência do Slide: Slide 10 - Quais são as principais vantagens da arquitetura de microsserviços em comparação com uma aplicação monolítica?

- **Definição:** **A arquitetura de microsserviços é uma abordagem na qual uma aplicação é dividida em serviços menores e independentes, cada um executando uma função específica. Diferentemente da arquitetura monolítica, que funciona como uma unidade única e coesa, os microsserviços permitem maior flexibilidade e escalabilidade.**
- **Aprofundamento/Complemento:** As principais vantagens dos microsserviços sobre as aplicações monolíticas são:
    - **Escalabilidade independente:** Cada serviço pode ser escalado de forma independente dos outros, otimizando o uso de recursos. Por exemplo, em um e-commerce, o serviço de busca pode ser escalado durante picos de acesso sem a necessidade de escalar o sistema inteiro.
    - **Facilidade de manutenção:** Como os serviços são menores e focados em uma única funcionalidade, a manutenção e a introdução de novas funcionalidades tornam-se mais simples e rápidas.
    - **Independência de desenvolvimento e tecnologia:** As equipes de desenvolvimento podem trabalhar de forma autônoma em diferentes serviços. Além disso, é possível utilizar diferentes tecnologias e linguagens de programação para cada microsserviço, escolhendo a mais adequada para cada tarefa.
    - **Resiliência:** A falha em um microsserviço não compromete toda a aplicação, desde que existam mecanismos de tratamento de falhas. Isso aumenta a disponibilidade geral do sistema.

- **Exemplo Prático:** Imagine um aplicativo de streaming de vídeo como a Netflix. Em uma arquitetura de microsserviços, poderíamos ter serviços separados para: autenticação de usuários, catálogo de filmes, recomendações, streaming de vídeo e faturamento. Se o serviço de recomendações falhar, os usuários ainda poderão pesquisar e assistir a vídeos, pois os outros serviços continuarão funcionando de forma independente.
- **Vídeos Sugeridos:**
    - O que é a Arquitetura de Microsserviços vs Monolítica: https://youtu.be/cqnwYicxUYk?si=zbgNDACarJVjW112

Referência do Slide: Slide 10 - Por que é importante escolher o modo de comunicação adequado (síncrono ou assíncrono) entre microsserviços?

- **Definição:** A escolha do modo de comunicação entre microsserviços, síncrono ou assíncrono, é crucial pois afeta diretamente a eficiência, a confiabilidade e a resiliência do sistema.
- **Aprofundamento/Complemento:**
    - **Comunicação Síncrona: Neste modelo, um serviço envia uma requisição para outro e aguarda uma resposta para continuar sua operação. É ideal para operações que necessitam de uma resposta imediata. O protocolo HTTP, com requisições REST, é um exemplo comum de comunicação síncrona. A desvantagem é que, se o serviço chamado estiver lento ou indisponível, o serviço solicitante ficará bloqueado, podendo gerar um efeito cascata de falhas.**
    - **Comunicação Assíncrona: Neste modelo, um serviço envia uma mensagem e não precisa esperar por uma resposta imediata. A comunicação geralmente é feita por meio de um intermediário, como uma fila de mensagens (message broker). Este padrão é mais resiliente a falhas e lida melhor com altas cargas de trabalho, pois desacopla os serviços. Se o serviço destinatário estiver indisponível, a mensagem permanecerá na fila para ser processada quando ele voltar a operar.**
- **Exemplo Prático:**
    - **Síncrono:** Ao realizar uma compra online, o serviço de pagamento precisa verificar em tempo real com o serviço da operadora de cartão de crédito se a transação foi aprovada. A resposta (aprovada ou negada) é necessária imediatamente para confirmar ou não a compra para o cliente.
    - **Assíncrono:** Após a confirmação da compra, o serviço de pedidos envia uma mensagem para uma fila para que o serviço de "envio de e-mail de confirmação" a processe. O serviço de pedidos não precisa esperar o e-mail ser efetivamente enviado para finalizar a compra. O e-mail pode ser enviado alguns segundos ou minutos depois, sem impactar a experiência do usuário no fechamento do pedido.

Referência do Slide: Slide 10 - Como vocês lidariam com a possibilidade de falha na comunicação entre os microsserviços?

- **Definição:** Lidar com falhas de comunicação é fundamental em uma arquitetura de microsserviços para garantir a robustez e a resiliência do sistema. **Estratégias como retries, fallbacks e o uso de filas de mensagens são implementadas para assegurar a entrega de dados e a continuidade do serviço.**
- **Aprofundamento/Complemento:**
  - **Retries (Repetições Automáticas): Consiste em tentar reenviar uma requisição que falhou por um número determinado de vezes. É útil para falhas transitórias, como uma breve instabilidade na rede. É importante implementar essa estratégia com um "exponential backoff", que aumenta o tempo de espera entre cada nova tentativa para não sobrecarregar o serviço alvo.**
  - **Fallback (Plano de Contingência): Quando uma requisição a um serviço falha após as tentativas de retry, um plano de contingência é acionado. Isso pode significar, por exemplo, retornar um valor padrão, dados de um cache ou uma resposta que indique que a funcionalidade está temporariamente indisponível, mas sem que isso cause um erro crítico na aplicação principal.**
  - **Filas de Mensagens (Queues): Para comunicação assíncrona, as filas de mensagens são uma solução robusta para falhas. Se o serviço consumidor estiver fora do ar, a mensagem permanece na fila e será processada assim que o serviço se recuperar, garantindo que a informação não seja perdida**.
  - **Circuit Breaker (Disjuntor): É um padrão que monitora as chamadas a um serviço. Se o número de falhas atinge um determinado limite, o "disjuntor" abre e impede novas chamadas a esse serviço por um tempo, evitando o esgotamento de recursos com requisições que provavelmente falhariam. Após um tempo, ele tenta uma chamada de teste e, se bem-sucedida, fecha o circuito, retomando o fluxo normal.**
- **Exemplo Prático:** Em um site de e-commerce, se o serviço de recomendação de produtos está fora do ar, ao invés de exibir uma mensagem de erro na página principal, a aplicação pode usar um fallback para exibir uma lista de produtos mais vendidos que foi salva em cache anteriormente. Isso mantém a página funcional e a experiência do usuário fluida, mesmo com a falha de um serviço secundário.

Referência do Slide: Slide 10 - No cenário apresentado, quais serviços externos seriam úteis para gerenciar a comunicação entre os microsserviços?

- **Definição: Para gerenciar a comunicação assíncrona, ferramentas como RabbitMQ e Kafka são extremamente úteis, atuando como *message brokers*. Para comunicação síncrona, o uso de APIs REST é uma abordagem padrão para tarefas como validação de endereços.**
- **Aprofundamento/Complemento:**
  - **RabbitMQ: É um *message broker* de código aberto que implementa o protocolo AMQP (Advanced Message Queuing Protocol). Ele é conhecido por sua flexibilidade em roteamento de mensagens e por**

garantir a entrega. É ideal para cenários que exigem lógicas complexas de enfileiramento e distribuição de tarefas.

- **Apache Kafka: É uma plataforma de streaming de eventos distribuída. Diferente de um *message broker* tradicional, o Kafka foi projetado para alta performance (throughput) e para armazenar logs de eventos de forma durável e replicada. É excelente para pipelines de dados em tempo real, analytics e sistemas que precisam processar um grande volume de mensagens sequencialmente.**
- **APIs REST: Para comunicação síncrona, a criação e o consumo de APIs RESTful é a abordagem mais comum. Um serviço pode expor um *endpoint* (URL) que outro serviço consome para obter uma informação específica, como um serviço de CEP que, ao receber um número, retorna o endereço completo.**

- **Exemplo Prático:** Em um sistema de logística, quando um pedido é despachado, o serviço de "Pedidos" pode publicar um evento "PedidoDespachado" em um tópico do Kafka. O serviço de "Notificações" e o serviço de "Rastreamento" podem "escutar" esse tópico e reagir ao evento de forma independente: o primeiro enviando um SMS ao cliente e o segundo atualizando o status do rastreio. Para uma tarefa mais simples, como processar o pagamento, o RabbitMQ poderia ser usado para enfileirar a tarefa de "gerar nota fiscal" após a confirmação do pagamento.
