Aula01: Programação com Micro Serviços

Contextualização sobre Micro Serviços

1. Surgimento dos Micro Serviços:

A arquitetura de micro serviços surgiu em resposta às necessidades de escalabilidade e flexibilidade dos sistemas modernos. Com o crescimento da web, aplicativos começaram a se tornar cada vez mais complexos e a estrutura tradicional monolítica enfrentava dificuldades em acompanhar esse crescimento.

Monolito era o modelo mais comum de desenvolvimento, onde todo o código e todas as funcionalidades de uma aplicação estavam agrupadas em um único sistema. Isso funcionava bem para aplicativos menores, mas conforme a demanda crescia, a abordagem monolítica gerava problemas, como:

- **Dificuldade em escalar** partes específicas do sistema sem escalar a aplicação inteira.
- **Tempos longos de desenvolvimento e deploy**: qualquer pequena mudança exigia que a aplicação inteira fosse recompilada e implantada.
- **Manutenção complicada**: com o crescimento do sistema, era difícil gerenciar a complexidade, causando problemas de dependências e introdução de bugs.

Empresas como Netflix, Amazon e Google, que operavam em uma escala massiva, começaram a enfrentar esses problemas e precisavam de uma maneira mais eficiente de construir e manter suas aplicações. A solução foi dividir o monolito em componentes menores, mais fáceis de gerenciar. Essa estratégia deu origem à arquitetura de micro serviços.

---

2. Dores que os Micro Serviços Vieram Resolver:

A adoção da arquitetura de micro serviços foi motivada por vários problemas específicos que surgiram com o crescimento dos sistemas monolíticos:

1. **Escalabilidade Rígida**:
   No modelo monolítico, para escalar uma parte específica da aplicação, como o processamento de pagamentos, era necessário escalar toda a aplicação. Isso desperdiça recursos, pois partes da aplicação que não precisam de escalabilidade adicional também são escaladas.

   - **Solução com Micro Serviços**: Os micro serviços permitem que cada componente seja escalado independentemente. Por exemplo, se o módulo de pagamentos precisar de mais recursos, você pode escalar apenas o serviço responsável por pagamentos sem afetar o restante da aplicação.

2. **Ciclos de Desenvolvimento Lentíssimos**:
   Como toda a aplicação está embutida em uma única base de código, qualquer alteração em uma pequena parte exigia uma nova compilação e um novo deploy de toda a aplicação. Isso retardava os ciclos de desenvolvimento e criava gargalos.

   - **Solução com Micro Serviços**: Cada micro serviço tem seu próprio ciclo de vida. Atualizações e deploys podem ser feitos em serviços individuais sem impactar o sistema inteiro, permitindo ciclos de desenvolvimento mais rápidos e isolados.

3. **Manutenção e Confiabilidade**:
   Em sistemas monolíticos grandes, era difícil identificar problemas e implementar novas funcionalidades sem impactar o código existente. Um bug em uma pequena parte do sistema podia causar a falha da aplicação inteira.

   - **Solução com Micro Serviços**: Com a separação em serviços, um erro em um micro serviço não afeta o funcionamento dos outros. Além disso, a arquitetura permite que equipes diferentes se concentrem em partes específicas, facilitando a manutenção.

4. **Escalabilidade de Times**:
   Em organizações grandes, o trabalho colaborativo no monolito era problemático. Como todos trabalhavam em um único repositório de código, as equipes tinham que coordenar constantemente para evitar conflitos, o que limitava a produtividade.

   - **Solução com Micro Serviços**: Equipes podem ser designadas para trabalhar em diferentes micro serviços de maneira autônoma, sem a necessidade de coordenação constante com outras equipes. Isso

aumenta a eficiência e permite que múltiplas funcionalidades sejam desenvolvidas em paralelo.

---

3. Como a Arquitetura de Micro Serviços Resolve Essas Dores:

- **Modularidade e Independência**:
  Em vez de uma aplicação ser um bloco monolítico gigante, a arquitetura de micro serviços separa a aplicação em módulos independentes, ou seja, pequenos serviços que executam uma única funcionalidade bem definida (por exemplo, gerenciamento de usuários, catálogo de produtos, processamento de pagamentos). Isso traz flexibilidade para desenvolver, testar e implantar cada serviço separadamente.
- **Escalabilidade Independente**:
  Cada micro serviço pode ser escalado independentemente de acordo com as necessidades de uso. Se um serviço de pagamento está sobrecarregado, ele pode ser escalado sem a necessidade de escalar toda a aplicação. Isso é fundamental para sistemas que lidam com altos volumes de dados e usuários.
- **Resiliência**:
  Em um sistema monolítico, se uma parte do sistema falhar, o sistema inteiro pode sair do ar. Com micro serviços, cada serviço roda de forma independente. Se um serviço falhar, os demais continuam operando, o que melhora a resiliência do sistema.
- **Ciclos de Deploy e Desenvolvimento Ágeis**:
  Como os micro serviços são independentes, equipes podem trabalhar simultaneamente em diferentes partes do sistema sem afetar os demais serviços. Isso reduz o tempo de desenvolvimento e permite a entrega contínua, onde novas funcionalidades podem ser lançadas mais rapidamente.

---

4. Exemplos de Uso de Micro Serviços:

- **Netflix**: Uma das pioneiras no uso de micro serviços, a Netflix desmembrou seu monolito em centenas de serviços independentes. Cada funcionalidade, desde o gerenciamento de perfis de usuários até o streaming de vídeo, é operada por um serviço diferente, o que permite uma escala massiva e um tempo de inatividade quase zero.

- **Amazon**: Com o crescimento exponencial do comércio eletrônico da Amazon, a empresa migrou para micro serviços para facilitar a escalabilidade de funcionalidades como processamento de pedidos, recomendação de produtos e inventário. Cada uma dessas funcionalidades agora opera em um serviço separado.
- **Uber**: O Uber também adotou a arquitetura de micro serviços para lidar com sua base global de usuários. O sistema de geolocalização, o cálculo de tarifas e o gerenciamento de motoristas são operados por serviços independentes, permitindo que a empresa faça ajustes e melhorias contínuas.

---

5. Como Micro Serviços São Usados Hoje:

Atualmente, a arquitetura de micro serviços é amplamente utilizada por grandes e pequenas empresas, especialmente aquelas que operam em ambientes com alta demanda, como:

- **E-commerce** (Amazon, eBay): onde diferentes partes do sistema, como pagamento, inventário e logística, podem ser desenvolvidas e mantidas separadamente.
- **Streaming de vídeo** (Netflix, YouTube): permitindo a entrega contínua de conteúdo sem interrupções.
- **Aplicativos móveis e serviços na nuvem** (Uber, Spotify): que exigem escalabilidade, resiliência e desenvolvimento ágil.

Além disso, micro serviços são uma parte central da **cultura DevOps**. A separação de responsabilidades e a automação de processos de deploy e testes contínuos são facilitados por essa arquitetura.

---

1. O que são Micro Serviços?

Micro serviços são uma abordagem de desenvolvimento de software onde uma aplicação é dividida em pequenos serviços independentes. Cada serviço é responsável por uma funcionalidade específica, pode ser desenvolvido em linguagens diferentes e funciona de forma independente. Esses serviços se comunicam entre si, geralmente por meio de APIs, como REST ou gRPC.

Características Principais:

- **Autonomia**: Cada serviço tem sua própria lógica, banco de dados e ciclo de vida.
- **Comunicação via APIs**: Serviços comunicam-se normalmente por HTTP (REST) ou usando protocolos de mensageria.
- **Independência Tecnológica**: Os serviços podem ser escritos em diferentes linguagens e tecnologias.
- **Escalabilidade**: Serviços individuais podem ser escalados separadamente.

---

2. A Evolução do Backend

Antigamente, a maioria dos sistemas era desenvolvida usando arquitetura **monolítica**, onde todas as funcionalidades do sistema eram agrupadas em uma única aplicação. Isso dificultava a escalabilidade e a manutenção conforme o sistema crescia.

Exemplo de Monolito:

```
// Node.js (Monolito)
const express = require('express');
const app = express();

// Funcionalidade de Usuário
app.get('/users', (req, res) => {
    res.send('Listagem de usuários');
});

// Funcionalidade de Produtos
app.get('/products', (req, res) => {
    res.send('Listagem de produtos');
});

app.listen(3000, () => {
    console.log('Aplicação monolítica rodando na porta 300
0');
});
```

No exemplo acima, as funcionalidades de usuários e produtos estão dentro da mesma aplicação.

Exemplo de Micro Serviço:

```
// Micro serviço de Usuários em Node.js
const express = require('express');
const app = express();

app.get('/users', (req, res) => {
    res.send('Listagem de usuários');
});

app.listen(4000, () => {
    console.log('Micro serviço de Usuários rodando na porta 4000');
});

// Micro serviço de Produtos em Node.js
const express = require('express');
const app = express();

app.get('/products', (req, res) => {
    res.send('Listagem de produtos');
});

app.listen(5000, () => {
    console.log('Micro serviço de Produtos rodando na porta 5000');
});
```

No exemplo de micro serviços, temos duas aplicações independentes, uma para a gestão de usuários e outra para produtos, rodando em diferentes portas.

---

3. Benefícios da Arquitetura de Micro Serviços

- **Facilidade de Escalabilidade**: Como os serviços são independentes, é possível escalar partes do sistema separadamente com base na demanda de cada funcionalidade.
- **Desenvolvimento Descentralizado**: Diferentes times podem trabalhar em diferentes micro serviços sem depender uns dos outros.
- **Resiliência**: Um serviço pode falhar sem causar a queda do sistema inteiro.

Exemplo de Escalabilidade:

Se a demanda por produtos aumentar, apenas o serviço de produtos pode ser escalado horizontalmente, ou seja, podemos rodar várias instâncias dele sem afetar os outros serviços.

---

4. Desafios da Arquitetura de Micro Serviços

- **Complexidade de Integração**: Coordenar e gerenciar a comunicação entre os serviços pode ser complicado.
- **Gerenciamento de Dados**: Manter a consistência entre diferentes bancos de dados em serviços independentes pode ser desafiador.
- **Latência de Rede**: A comunicação entre micro serviços pode gerar latência adicional.

---

5. Exemplos Práticos

Exemplo de Micro Serviço em Node.js

Aqui está um exemplo de um micro serviço de gerenciamento de usuários em Node.js:

```
// users-service.js (Micro Serviço de Usuários)
const express = require('express');
const app = express();

// Simulação de banco de dados em memória
let users = [
    { id: 1, name: 'John Doe' },
    { id: 2, name: 'Jane Doe' }
];
```

```
// Rota para listar usuários
app.get('/users', (req, res) => {
    res.json(users);
});

// Rota para adicionar um novo usuário
app.post('/users', (req, res) => {
    const newUser = { id: users.length + 1, name: req.body.
name };
    users.push(newUser);
    res.status(201).json(newUser);
});

app.listen(3001, () => {
    console.log('Micro serviço de Usuários rodando na porta
3001');
});
```

Exemplo de Micro Serviço em Java (Spring Boot)

Aqui está um exemplo de micro serviço de produtos em Java usando Spring Boot:

```
// ProductsService.java (Micro Serviço de Produtos)
package com.example.products;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestControll
er;

import java.util.Arrays;
import java.util.List;

@RestController
public class ProductsController {
```

```
    @GetMapping("/products")
    public List<String> getProducts() {
        return Arrays.asList("Product 1", "Product 2", "Pro
duct 3");
    }
}
```

Configuração básica para rodar o micro serviço:

```
<!-- pom.xml -->
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

Rodando esse serviço, ele estará disponível na porta padrão (8080).

---

6. Comunicação entre Micro Serviços

A comunicação entre os micro serviços é um ponto essencial. Em geral, eles utilizam APIs REST ou filas de mensagens (RabbitMQ, Kafka) para se comunicar. Aqui está um exemplo simples de comunicação entre dois micro serviços usando HTTP:

- O serviço de "Produtos" faz uma chamada HTTP ao serviço de "Usuários" para buscar dados de um usuário específico.

Exemplo de Comunicação:

```
// Serviço de Produtos chamando serviço de Usuários
const axios = require('axios');

app.get('/products/:id', async (req, res) => {
    const userId = req.params.id;
```

```
    try {
        // Faz uma requisição ao serviço de usuários
        const response = await axios.get(`http://localhost:
3001/users/${userId}`);
        res.json({
            productId: req.params.id,
            user: response.data
        });
    } catch (error) {
        res.status(500).send('Erro na comunicação com o ser
viço de Usuários');
    }
});
```

Conclusão

Nesta aula, introduzimos o conceito de micro serviços, discutimos sua evolução, benefícios e desafios, e mostramos exemplos práticos de como implementar um micro serviço em Node.js e Java. Nas próximas aulas, continuaremos explorando como projetar, integrar e testar esses serviços.