Aula 02: A Arquitetura de Micro Serviços e a Evolução do Backend

Objetivo da Aula:

- Apresentar a evolução do desenvolvimento backend, desde os sistemas monolíticos até a arquitetura de micro serviços.
- Explicar os componentes e as características da arquitetura de micro serviços.
- Mostrar exemplos de como essa arquitetura é organizada e implementada.

Sumário

1. A Evolução do Backend

O desenvolvimento back end tem uma longa trajetória, passando por diferentes abordagens à medida que a complexidade dos sistemas aumentava e novas demandas surgiam.

1.1. Sistemas Monolíticos (Início)

No começo da era da computação, as aplicações eram monolíticas. Isso significa que todo o código-fonte e as funcionalidades estavam centralizadas em um único projeto. Embora simples de desenvolver e implantar, os sistemas monolíticos apresentavam desafios à medida que os projetos cresciam.

Características dos Monolitos:

- **Código Centralizado**: Todo o código da aplicação (autenticação, banco de dados, lógica de negócios) está em um único pacote.
- **Escalabilidade Limitada**: A aplicação inteira precisa ser escalada mesmo que apenas uma funcionalidade necessite de mais recursos.
- **Manutenção Complexa**: Alterações ou atualizações em uma pequena parte da aplicação exigem recompilar e reinstalar o sistema inteiro.

Exemplo de um sistema monolítico (Node.js):

```
const express = require('express');
const app = express();

// Funcionalidade de autenticação
app.get('/login', (req, res) => {
    res.send('Login Page');
});

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

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

```
0');
});
```

1.2. Arquitetura em Camadas (MVC)

Para lidar com a complexidade crescente, as arquiteturas baseadas em camadas, como o **Model-View-Controller (MVC)**, começaram a surgir. Elas separavam a lógica da interface do usuário (view), a lógica de negócios (model) e o controle de fluxo (controller), melhorando a organização do código.

Características da Arquitetura MVC:

- Separação de responsabilidades (UI, lógica de negócios e controle).
- Mais organizada, mas ainda sofre com escalabilidade limitada em sistemas grandes.

No entanto, conforme os sistemas começaram a crescer ainda mais, o MVC mostrou limitações em termos de escalabilidade e complexidade de manutenção.

1.3. Arquitetura Orientada a Serviços (SOA)

A **Arquitetura Orientada a Serviços (SOA)** foi uma tentativa de modularizar os sistemas de software. Em vez de um grande monolito, a SOA dividia a aplicação em serviços reutilizáveis que se comunicavam entre si. Embora tenha sido um avanço, a SOA apresentava problemas de sobrecarga de comunicação e complexidade na integração de sistemas.

Características da SOA:

- Reutilização de serviços em diferentes partes da aplicação.
- Pesada, com integração complexa entre serviços.

1.4. Arquitetura de Micro Serviços

A arquitetura de micro serviços evoluiu a partir da SOA, mas com o foco em serviços ainda mais independentes e leves. Em vez de serviços pesados e complexos, os micro serviços promovem a criação de pequenos serviços que podem ser facilmente escalados e desenvolvidos separadamente. Cada serviço é responsável por uma funcionalidade específica do sistema.

---

2. O Que É a Arquitetura de Micro Serviços?

A **arquitetura de micro serviços** é um estilo arquitetural que divide uma aplicação em um conjunto de serviços menores, independentes e orientados a tarefas específicas. Esses serviços podem ser desenvolvidos, implantados e escalados de forma autônoma.

Características da Arquitetura de Micro Serviços:

- **Descentralização**: Diferentes serviços têm suas próprias responsabilidades e podem ser mantidos e desenvolvidos independentemente.
- **Escalabilidade Independente**: Cada serviço pode ser escalado de acordo com suas necessidades, sem impactar os outros.
- **Desenvolvimento Autônomo**: Times diferentes podem trabalhar em diferentes serviços sem interferência, usando diferentes linguagens e tecnologias, se necessário.
- **Resiliência**: A falha de um serviço não impacta o sistema inteiro. Outros serviços continuam funcionando.

---

3. Componentes da Arquitetura de Micro Serviços

3.1. Serviço Individual

Cada serviço na arquitetura de micro serviços representa uma pequena unidade de funcionalidade. Um serviço pode ser responsável pelo gerenciamento de usuários, outro por processamento de pagamentos, e assim por diante.

Exemplo de serviço em **Node.js** (Micro serviço de usuários):

```
const express = require('express');
const app = express();

let users = [{ id: 1, name: 'Alice' }, { id: 2, name: 'Bob'
}];

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

```
app.listen(3000, () => {
    console.log('Micro serviço de Usuários rodando na porta 3000');
});
```

3.2. API Gateway

O **API Gateway** é uma camada que atua como uma fachada para o conjunto de micro serviços. Ele fornece uma única entrada para os clientes e distribui as solicitações para os micro serviços apropriados. O API Gateway também pode implementar autenticação, roteamento, monitoramento e balanceamento de carga.

Funções do API Gateway:

- Roteamento de requisições para os micro serviços.
- Centralização de autenticação e autorização.
- Agregação de dados de diferentes serviços para criar respostas mais completas.

3.3. Comunicação entre Serviços

Os micro serviços geralmente se comunicam por meio de APIs REST (HTTP), mas podem usar outros mecanismos de comunicação, como **gRPC** (mais eficiente em termos de performance) ou sistemas de mensageria como **RabbitMQ** e **Kafka** para enviar mensagens assíncronas.

Exemplo de comunicação via HTTP entre serviços:

```
const axios = require('axios');

// Serviço de Produtos chamando o serviço de Usuários
axios.get('http://localhost:3000/users')
  .then(response => {
    console.log('Usuários:', response.data);
  })
  .catch(error => {
```

```
    console.error('Erro ao chamar serviço de usuários', err
or);
  });
```

3.4. Banco de Dados Descentralizado

Cada micro serviço pode ter seu próprio banco de dados, evitando o acoplamento entre os serviços. Isso melhora a modularidade, mas pode introduzir desafios na consistência dos dados e na sincronização entre serviços.

Vantagem: Maior autonomia e flexibilidade para escolher o banco de dados que melhor se adapta a cada serviço.

Desafio: Garantir consistência de dados entre serviços, especialmente em operações transacionais.

3.5. Monitoramento e Logging

Com múltiplos serviços rodando em paralelo, é fundamental implementar um sistema robusto de monitoramento e logs para detectar falhas e gargalos. Ferramentas como **Prometheus** para monitoramento e **ELK (Elasticsearch, Logstash, Kibana)** para centralização de logs são comuns em arquiteturas de micro serviços.

---

4. Comparação com Arquiteturas Anteriores

Monolítico vs Micro Serviços

Monolítico	Micro Serviços
Um único código que contém todas as funcionalidades	Dividido em pequenos serviços independentes
Atualizações e deploys lentos	Ciclos de desenvolvimento e deploys rápidos
Escalabilidade limitada	Escalabilidade independente de cada serviço
Falha de um componente afeta o sistema todo	Falhas isoladas a serviços individuais

5. Como a Arquitetura de Micro Serviços é Usada Hoje

A arquitetura de micro serviços está amplamente presente em sistemas que precisam lidar com alta demanda e escalabilidade, como:

- **E-commerce**: Onde diferentes partes do sistema, como carrinho de compras, catálogo e pagamentos, podem ser escaladas separadamente.
- **Streaming**: Netflix e YouTube utilizam essa arquitetura para servir milhões de vídeos a usuários de maneira eficiente.
- **Serviços Bancários**: Bancos adotam micro serviços para segmentar funcionalidades críticas como autenticação, movimentação financeira e monitoramento de fraudes.

---

6. Conclusão

Nesta aula, revisamos a evolução do backend, desde sistemas monolíticos até a adoção de micro serviços, entendendo como essa arquitetura traz benefícios em termos de escalabilidade, resiliência e flexibilidade. Também cobrimos os principais componentes de uma arquitetura de micro serviços e como eles interagem para formar um sistema robusto e escalável.

Para saber mais:

- Pesquise sobre ferramentas de comunicação entre micro serviços (como gRPC, RabbitMQ).