### **O que é computação serverless?**

### A computação serverless é um modelo de computação em nuvem que permite que os desenvolvedores executem código sem precisar gerenciar servidores. Neste modelo, os provedores de nuvem, como AWS, Azure ou Google Cloud, cuidam da infraestrutura subjacente, permitindo que os desenvolvedores se concentrem na lógica do aplicativo.

#### **Vantagens:**

### **Escalabilidade automática:** A infraestrutura escala automaticamente para atender à demanda, acomodando picos de acesso sem necessidade de configuração manual.

### **Modelo de custo eficiente:** Os custos são baseados no uso real, permitindo que as organizações paguem apenas pelo tempo de execução do código.

### **Redução da sobrecarga de manutenção:** Os desenvolvedores não precisam se preocupar com a configuração e manutenção do servidor, liberando tempo para focar em desenvolvimento.

#### **Desvantagens:**

### **Limitações de execução:** Muitas plataformas impõem limites de duração para a execução de funções, o que pode não ser adequado para todos os tipos de aplicação.

### **Dependência do fornecedor:** O lock-in pode ocorrer, dificultando a migração para outras plataformas de nuvem.

### **Desafios de depuração:** A natureza distribuída pode complicar o diagnóstico de falhas e a rastreabilidade de eventos.

### **O que é uma arquitetura de microserviços?**

### A arquitetura de microserviços é um estilo de design de software onde uma aplicação é composta de pequenos serviços independentes, cada um responsável por uma funcionalidade específica. Esses serviços se comunicam entre si através de APIs, permitindo que sejam desenvolvidos, implantados e escalados de forma independente.

#### **Diferenças entre microserviços e arquitetura monolítica:**

### **Monolítica:** Uma aplicação monolítica é construída como uma única unidade. Qualquer alteração ou atualização requer a recompilação e o redeploy de toda a aplicação.

### **Microserviços:** Em uma arquitetura de microserviços, cada serviço pode ser atualizado de forma independente, facilitando a manutenção e a agilidade na entrega de novos recursos.

### **Exemplos de empresas que utilizam serverless e microserviços:**

### **Serverless:**

### **Netflix:** Utiliza serviços serverless para processamento de dados e funções específicas.

### **Amazon:** AWS Lambda é amplamente utilizado para diversas operações.

### **Microserviços:**

### **Spotify:** Implementa microserviços para gerenciar playlists, usuários e recomendações.

### **Uber:** Adota microserviços para facilitar a comunicação entre diferentes partes do sistema, como pedidos, pagamentos e localização.

### **Tabela comparativa: Serverless vs. Microserviços**

| **Característica** | **Serverless** | **Microserviços** |
| --- | --- | --- |
| **Escalabilidade** | Escalabilidade automática baseada na demanda | Escalabilidade manual por serviço |
| **Custo** | Paga-se apenas pelo uso | Custos fixos para infraestrutura e manutenção |
| **Gestão de Infraestrutura** | Sem necessidade de gestão de infraestrutura | Exige gestão ativa de infraestrutura |
| **Tempo de Execução** | Limitações na duração das execuções | Sem limitações de tempo para serviços |
| **Independência** | Dependência do fornecedor | Serviços independentes que podem ser atualizados separadamente |
| **Desenvolvimento** | Foco em funções pequenas e eventos | Foco em serviços maiores e mais complexos |
| **Casos de uso** | Aplicações que respondem a eventos | Aplicações complexas com múltiplas funcionalidades |
| **Desafios** | Complexidade na depuração e monitoramento | Gerenciamento de comunicação entre serviços e dados |

### 

### **Definição do Projeto**

Estamos criando um aplicativo de streaming de vídeo, onde as pessoas poderão assistir filmes e séries. Para isso, precisamos de uma boa estrutura que permita que muitas pessoas assistam ao mesmo tempo, sem travar.

### **Escolha da Arquitetura**

Decidimos usar uma arquitetura de microserviços. Isso significa que vamos dividir nosso aplicativo em partes menores, cada uma cuidando de uma tarefa específica. Assim, se um serviço precisar de uma atualização, podemos fazer isso sem afetar os outros.

#### **Exemplos de microserviços que vamos criar:**

* **Serviço de Usuários:** Cuida do cadastro e login dos usuários.
* **Serviço de Catálogo:** Gerencia a lista de filmes e séries disponíveis.
* **Serviço de Streaming:** Cuida de enviar os vídeos para os usuários.
* **Serviço de Recomendações:** Sugere filmes com base no que o usuário já assistiu.

### **Escolha das Ferramentas**

**Para desenvolver nosso aplicativo, vamos usar algumas ferramentas:**

* **Docker:** Para empacotar cada microserviço como um contêiner, facilitando a instalação e execução em qualquer lugar.
* Kubernetes: Para gerenciar esses contêineres, garantindo que todos os serviços funcionem juntos sem problemas.
* AWS (Amazon Web Services): Usaremos os serviços da AWS para armazenar nossos vídeos e fazer o streaming.

### **Desenvolvendo os Microserviços**

**Inicialmente, iremos programar os microserviços:**

* **Serviço de Usuários:** Vamos criar uma API que permita que os usuários se cadastrem e façam login. Utilizariamos Node.js e MongoDB para armazenar as informações.
* **Serviço de Catálogo:** Esse serviço é responsável por enviar a lista de filmes e séries. usariamos um banco de dados SQL para armazenar as informações sobre cada filme.
* **Serviço de Streaming:** Esse é um dos mais importantes! usariamos a tecnologia de streaming da AWS para garantir que os vídeos sejam entregues rapidamente aos usuários.
* **Serviço de Recomendações:** Aqui, estamos usariamos um algoritmo simples que analisa o que o usuário já assistiu e sugere novos filmes que ele pode gostar.

### **Testes e Integração**

Depois que todos os serviços estiverem prontos, vamos testar cada um deles para garantir que estão funcionando bem. Vamos integrar tudo para que os microserviços possam se comunicar.

### **Estratégia de Implantação**

Decidimos usar CI/CD (Integração Contínua e Entrega Contínua). Isso significa que, sempre que fazemos uma mudança em nosso código, ele é testado automaticamente e, se tudo estiver certo, é enviado para o ambiente de produção. Assim, podemos atualizar nosso aplicativo rapidamente.

### **Monitoramento**

Para garantir que tudo funcione bem após o lançamento, vamos usar ferramentas de monitoramento como o Prometheus e o Grafana. Elas nos ajudarão a ver se os serviços estão funcionando corretamente e se os usuários estão tendo uma boa experiência.