Microserviços: arquitetura e desafios

Fundamentos

Benefícios dos microserviços

Desacoplamento

- Desacoplamento
- Escalabilidade
- Resiliência
- Comunicação leve
- Automação e DevOps

Princípios de design

Desacoplamento e responsabilidade única

Desacoplamento

- Interfaces bem definidas
- Dados isolados
- Mensageria e eventos

Microserviços devem ser independentes, com mínima dependência entre eles. Isso facilita mudanças e escalabilidade.

SRP

- “Single Responsibility Principle”
- Modularidade
- Um dos cinco princípios do SOLID

Cada serviço deve ser responsável por uma única parte do negócio. Isso reduz a complexidade e melhora a manutenção.

Princípios de arquitetura

Protocolos eficientes e implantação independente

Comunicação entre serviços

- Uso de APIs: APIs REST, gRPC, ou mensageria
- Deploy independente

Conectar serviços de várias formas e atualizar cada um sem parar o sistema todo.

Uso de APIs

REST - Twitter API

Muito usado para comunicação HTTP entre microserviços.

gRPC - Google Cloud

Um framework de comunicação mais eficiente e rápido, baseado em HTTP/2 e Protobuf.

Mensageria - Uber

Utilizado quando a comunicação precisa ser assíncrona.

Implementação e integração contínua

Integração e testes contínuos de código.

Implementação contínua

- Menor chance de erros
- Correções e novas funcionalidades entregues rapidamente
- Automatização do processo de entrega
- Compatibilidade entre versões.

Integração contínua

- Detecção rápida de erros
- Manutenção da compatibilidade entre serviços
- Aceleração do desenvolvimento
- Complexidade dos testes
- Sincronização das versões

Importância para microserviços

- Automação
- Gerenciamento de serviços

Gerenciamento de estados e dados

Desafios e padrões em microserviços.

Desafios

- Gerenciamento de estados: Alguns microserviços precisam gerenciar estados, embora geralmente sejam stateless.
- Consistência de dados: Cada serviço tem seu próprio banco de dados, o que pode complicar a consistência de dados.

Padrões

- Stateful vs. Stateless
- Event Sourcing
- CQRS (Command Query Responsibility Segregation)

Transações

- Sagas
- BASE (Basically Available, Soft state, Eventually consistent)

Ferramentas

- Redis: Para gerenciamento de dados.
- Kafka: Para eventos e comunicação entre serviços.
- Consul: Para configuração distribuída e descoberta de serviços.

Resiliência e tolerância a falhas

Garantindo que o sistema continue funcionando apesar das falhas.

Resiliência e tolerância a falhas em sistemas distribuídos

Resiliência

Capacidade de recuperação rápida de falhas, permitindo operação contínua.

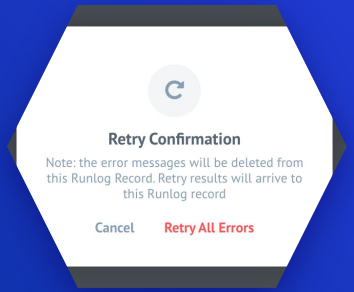
Tolerância e falhas

Adaptação a falhas sem causar interrupções significativas.

Estratégias

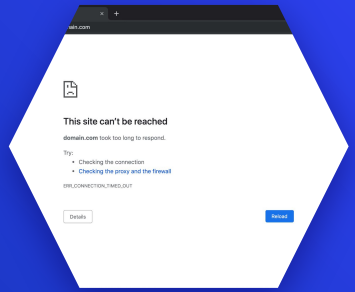
Retries

Novas tentativas em caso de falha.



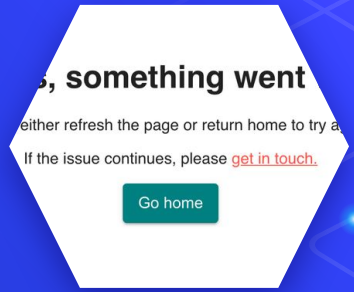
Timeouts

Limite de espera para evitar bloqueios.



Fallbacks

Respostas alternativas para manter a estabilidade.



Resumo de problemas e soluções

Soluções para desafios em microserviços.

Complexidade

- **Desafio:** Aumenta a complexidade de desenvolvimento e manutenção.
- **Solução:**
  - Domain-Driven Design
  - API Gateway
  - Kubernetes

Latência

- **Desafio:** Comunicação em rede pode causar atrasos significativos.
- **Solução:**
  - Cacheamento com Redis
  - Mensageria assíncrona com Kafka ou RabbitMQ
  - Otimização das chamadas remotas.

Gerenciamento de versões

- **Desafio:** Evolução independente pode causar problemas de compatibilidade.
- **Solução:**
  - Versionamento de API
  - Feature toggles para lançamentos graduais
  - Documentação clara com OpenAPI