RPC e Microserviços: Comunicação em Sistemas Distribuídos

Quando desenvolvemos sistemas de software, muitas vezes precisamos que diferentes programas ou partes de um sistema conversem entre si. Essa comunicação pode acontecer dentro de um mesmo computador ou entre computadores diferentes conectados em rede.

Para resolver esse desafio, surgiram dois conceitos fundamentais:

RPC (Remote Procedure Call)

Chamadas de procedimentos remotos que permitem executar funções em sistemas distantes

Microserviços

Arquitetura que organiza sistemas em pequenos serviços independentes

O que é RPC?

Definição

Remote Procedure Call é um protocolo que permite a execução de procedimentos em servidores remotos como se fossem chamadas locais, abstraindo a complexidade da comunicação de rede.

Comunicação Transparente

Cliente chama funções remotas sem se preocupar com detalhes de rede

Abstração de Rede

Marshalling/Unmarshalling automático de dados para transmissão

RPC: Pontos Fortes e Desafios

Pontos Fortes

Baixa latência, interface simples, suporte a múltiplas linguagens, comunicação síncrona eficiente

S

W

Desafios

Acoplamento temporal, complexidade em falhas de rede, dificuldade em debugging distribuído

Ameaças

Overhead de protocolo, competição com REST/GraphQL, complexidade operacional

T

O

Oportunidades

Integração com service mesh, otimizações de performance, uso em edge computing

A Conexão Entre RPC e Microserviços

Esses dois temas estão fortemente ligados. Enquanto o RPC é um mecanismo técnico de comunicação, os microserviços são uma forma de organizar sistemas modernos, que dependem justamente de mecanismos como RPC para funcionarem.

RPC — Remote Procedure Call

O que é RPC?

RPC é uma técnica que permite que um programa execute uma função em outro processo ou máquina como se fosse local.

Exemplo simples: Imagine que você está escrevendo um programa em Python. Normalmente, você chama funções como soma(2,3). Se essa função estivesse em outro computador, você precisaria abrir uma conexão de rede, serializar os dados, enviar a mensagem pela rede, esperar resposta e desserializar. Com RPC, tudo isso acontece automaticamente.

Origem

O conceito foi formalizado por Birrell e Nelson (1984), que buscavam abstrair a complexidade da comunicação distribuída.

Como Funciona o RPC?

O RPC segue este fluxo:

01

Cliente chama função local

O cliente chama uma função local (stub do cliente)

02

Empacotamento

O stub empacota os parâmetros (marshalling)

03

Envio pela rede

A mensagem é enviada pela rede até o servidor

04

Recebimento e desempacotamento

O servidor recebe e desempacota os dados (unmarshalling)

05

Execução

Executa a função real no servidor

06

Retorno do resultado

Retorna o resultado para o cliente

Tipos de RPC

RPC Tradicional

ONC RPC, CORBA: usavam protocolos binários e bibliotecas próprias

XML-RPC

Usa XML para representar chamadas

JSON-RPC

Usa JSON, muito usado em sistemas web

gRPC

Google, 2015: baseado em HTTP/2 e Protocol Buffers, rápido e eficiente

Microserviços

O que são?

> Segundo Fowler e Lewis (2014): *'Microservices are an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms.'*

Ou seja, em vez de um monólito, temos vários serviços pequenos, cada um com uma responsabilidade clara.

Arquitetura de Microserviços

Conceito

Arquitetura que decompõe aplicações em pequenos serviços independentes, cada um responsável por uma funcionalidade específica e comunicando-se via APIs.

Independência

Cada serviço pode ser desenvolvido, implantado e escalado isoladamente

Isolamento de Dados

Cada serviço possui seu próprio banco de dados e lógica de negócio

Características dos Microserviços

Ciclo de vida independente

Cada serviço tem seu próprio ciclo de vida, podendo ser desenvolvido, testado e implantado independentemente

Comunicação leve

HTTP/REST, gRPC, mensageria - protocolos simples e eficientes para comunicação entre serviços

Diversidade tecnológica

Pode ser escrito em linguagens diferentes, permitindo escolher a melhor tecnologia para cada problema

Escalabilidade independente

Cada serviço pode ser escalado conforme sua demanda específica, otimizando recursos

Exemplo Prático

Imagine um sistema de e-commerce dividido em microserviços:

Serviço de Usuários

Gerencia contas e logins

Serviço de Produtos

Gerencia catálogo

Serviço de Pedidos

Processa compras

Serviço de Estoque

Controla quantidade disponível

Quando um cliente faz uma compra, o Serviço de Pedidos chama remotamente uma função do Serviço de Estoque (via RPC ou REST) para verificar disponibilidade.

Conexão entre RPC e Microserviços

Benefícios dos Microserviços

85%	3x	70%	50%
Empresas Adotando	**Mais Rápido**	**Redução de Falhas**	**Escalabilidade**

Escalabilidade

> Cada serviço escala independentemente conforme demanda

- Alocação precisa de recursos
- Escalabilidade horizontal automática
- Otimização de custos

Desenvolvimento

> Equipes independentes aceleram o ciclo de desenvolvimento

- Deploy contínuo por serviço
- Menor tempo de time-to-market
- Tecnologias diversas por serviço

Resiliência

> Falhas isoladas não comprometem o sistema inteiro

- Circuit breakers automáticos
- Recuperação rápida de falhas
- Degradação gradual

Exemplos Reais

Netflix

Migrou para microserviços para lidar com milhões de streams simultâneos. Usa gRPC e mensageria para coordenar centenas de serviços independentes.

Uber

Começou como monólito em Python, mas escalou para microserviços em Go e Java, conectados por RPC para processar milhões de corridas diárias.

Comunicação entre Microserviços

Síncrona - RPC

Comunicação direta e rápida usando gRPC, Thrift ou REST para operações que necessitam resposta imediata

Assíncrona - Mensageria

Comunicação via filas e tópicos usando RabbitMQ, Kafka ou AWS SQS para processamento em lote

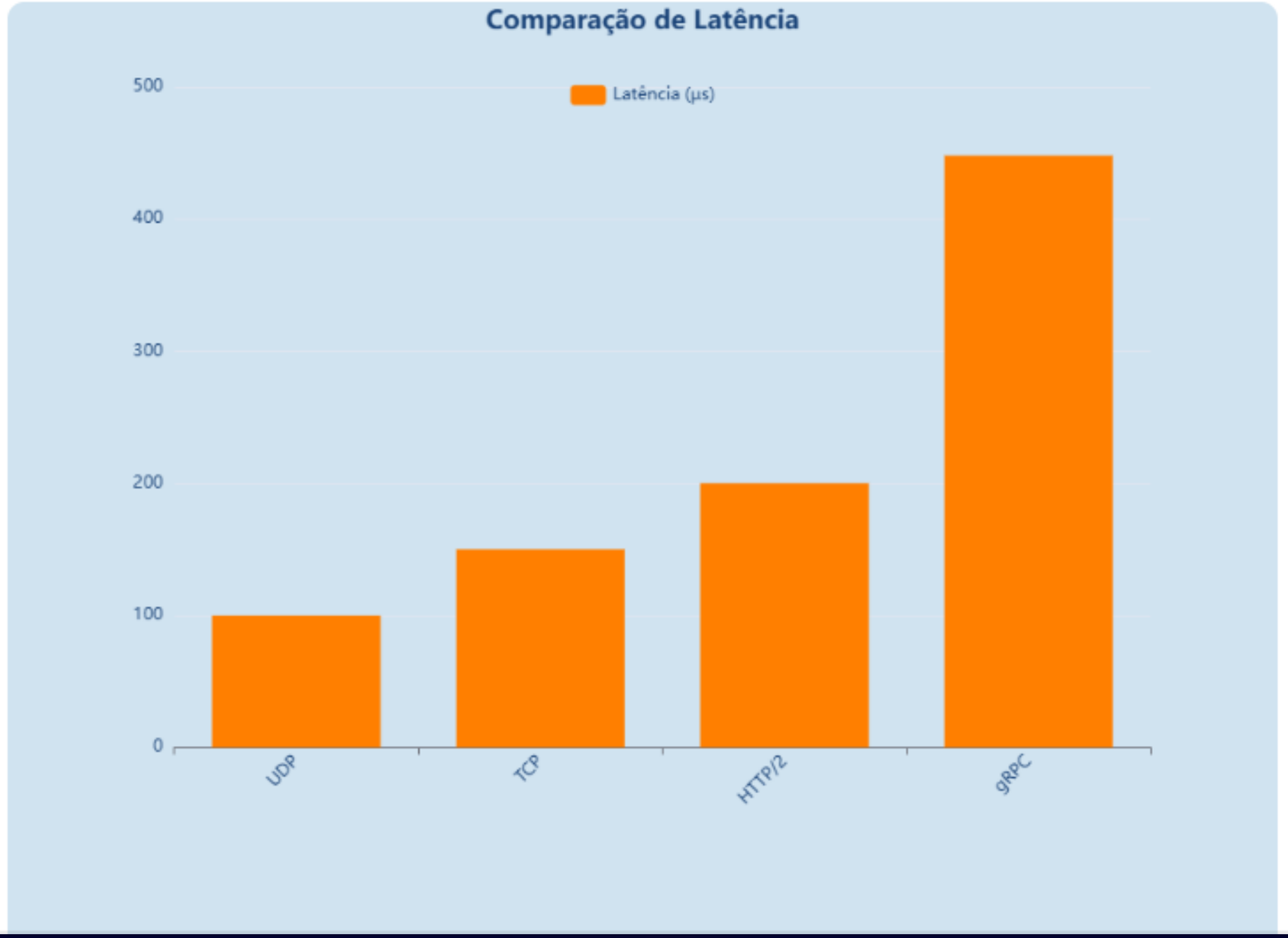
Service Mesh

Infraestrutura dedicada para gerenciar comunicação, segurança e observabilidade entre serviços

API Gateway

Ponto único de entrada que roteia requisições, aplica políticas de segurança e rate limiting

Performance: RPC vs Alternativas



gRPC Performance

Até 448% mais lento que UDP, mas oferece features de segurança e tipagem

Trade-offs

Escolha entre performance bruta vs funcionalidades avançadas e segurança

Benefícios e Desafios

Benefícios

Escalabilidade

Cada serviço pode ser escalado independentemente conforme a demanda

Independência de equipes

Times podem trabalhar em paralelo sem interferir uns nos outros

Resiliência

Falha em um serviço não derruba todo o sistema

Desafios

Complexidade de orquestração

Coordenar múltiplos serviços requer ferramentas e processos sofisticados

Comunicação distribuída

Latência e falhas de rede se tornam preocupações constantes

Observabilidade

Monitoramento distribuído é essencial mas complexo de implementar

Netflix: 700+ microserviços

Empresas Líderes

- Netflix: Streaming global com 700+ serviços
- Amazon: E-commerce com milhares de serviços
- Uber: 2.200 microserviços organizados em 70 domínios

Evolução para serverless

Tendências 2025

- Serverless containers com microserviços
- Edge computing distribuído
- AI/ML integrada aos serviços

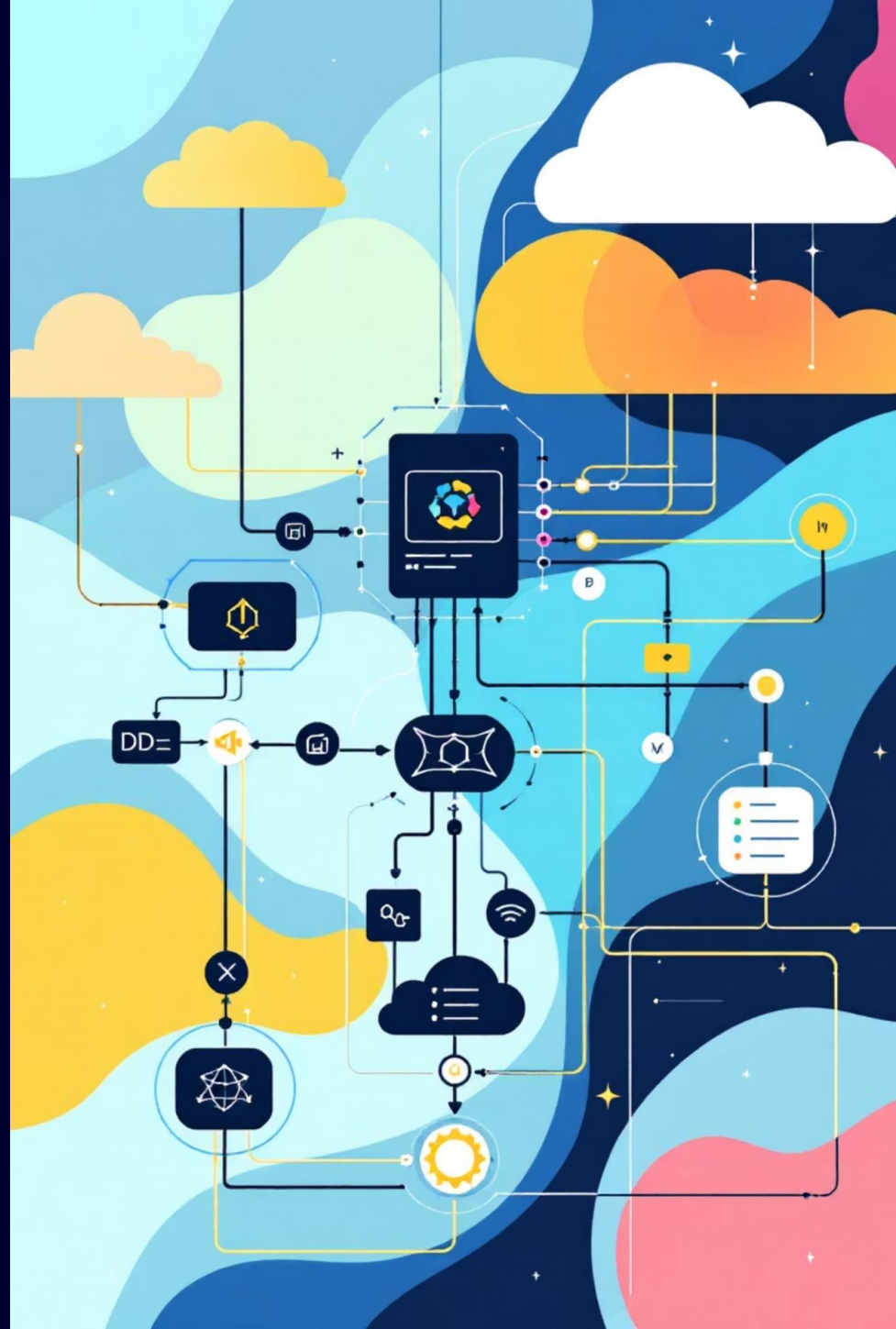
Observabilidade avançada

Ferramentas Modernas

- Service mesh com Istio/Linkerd
- Observabilidade com Prometheus/Grafana
- Deploy automatizado com Kubernetes

Módulo 2: RPC, Microserviços Avançados e Arquiteturas Modernas

Este módulo dá continuidade aos conceitos de RPC (Remote Procedure Call) e Microserviços, explorando práticas avançadas de comunicação, orquestração, resiliência e segurança em sistemas distribuídos. O objetivo é fornecer ao aluno uma visão clara de como arquiteturas modernas funcionam, do RPC simples até soluções baseadas em eventos, orquestradores e práticas de observabilidade.

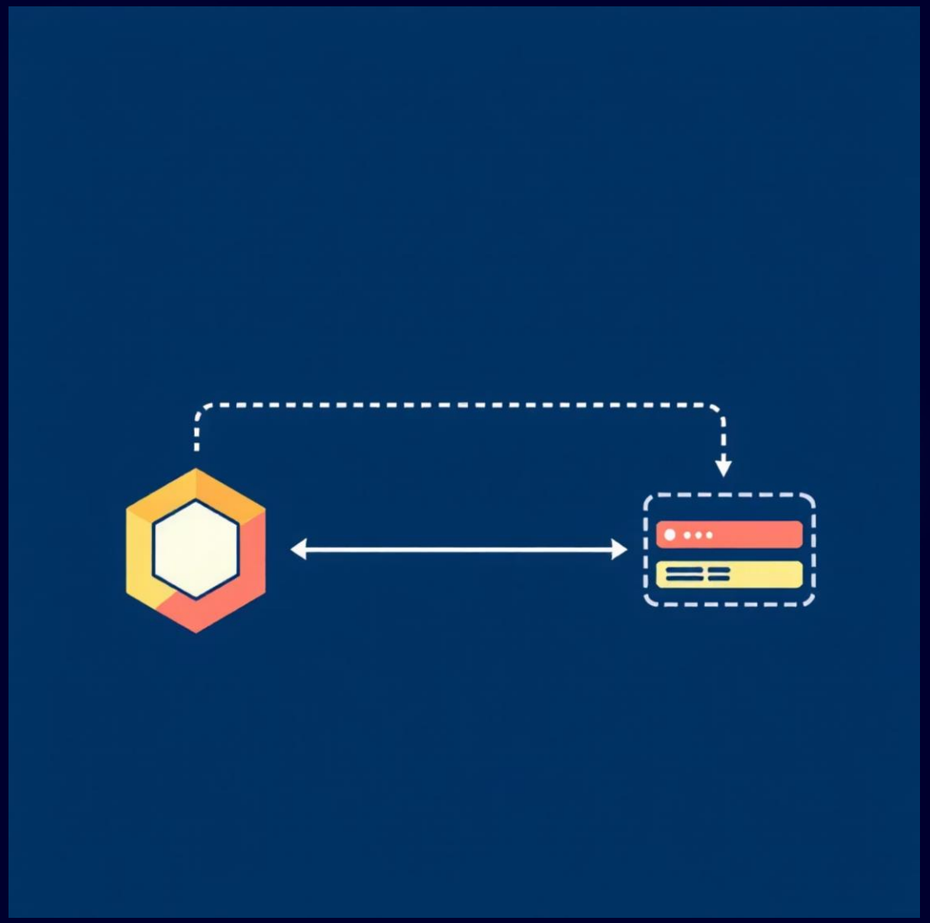


Revisão: RPC e Microserviços

RPC (Remote Procedure Call)

Permite que um processo invoque funções em

outro processo **ou** servidor como se fosse local.



Microserviços

Arquitetura em que o sistema é dividido em serviços independentes que se comunicam entre si, geralmente usando RPC.

Mensageria e Arquitetura Orientada a Eventos (EDA)

Enquanto RPC lida com chamadas diretas (requisição/resposta), a mensageria permite comunicação assíncrona entre serviços através de ferramentas como RabbitMQ e Apache Kafka.

RPC

Chamadas diretas

Requisição/Resposta

Mensageria

Comunicação assíncrona

RabbitMQ, Apache Kafka

EDA

Serviços reagem a eventos

Event-Driven Architecture

Exemplo Prático

Pagamento Aprovado

Evento "Pagamento Aprovado" é gerado

Serviço de Notificações

Reage enviando e-mails

Serviço de Estoque

Reage atualizando quantidade disponível

Orquestração e Service Mesh

Com o crescimento de serviços, é necessário gerenciar melhor a comunicação através de ferramentas especializadas de orquestração e controle de tráfego.

API Gateway

Ponto único de entrada para clientes, centralizando o acesso aos microserviços e fornecendo funcionalidades como autenticação, rate limiting e roteamento.

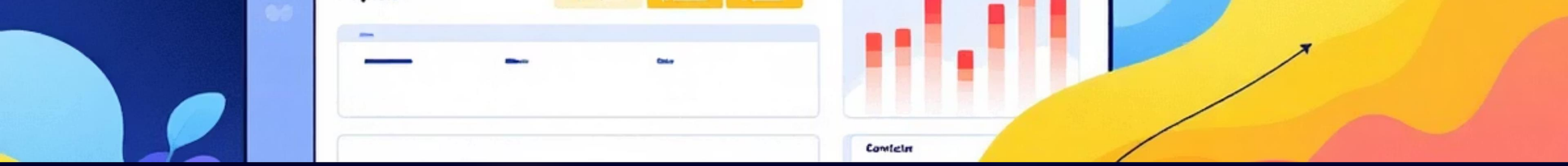
Kubernetes

Orquestração e escalabilidade de microserviços, gerenciando automaticamente a distribuição, scaling e health checks dos containers.

Service Mesh

Istio e Linkerd controlam tráfego, segurança e monitoramento entre serviços, fornecendo uma camada de infraestrutura dedicada.

> **Exemplo:** Kubernetes distribui múltiplas instâncias de um serviço de pagamento, enquanto Istio garante segurança (TLS) e monitoramento de chamadas entre serviços.



Boas Práticas em Sistemas Distribuídos

1

Observabilidade

Logs, métricas e tracing distribuído usando ferramentas como Prometheus e Jaeger para monitoramento completo do sistema.

2

Resiliência

Implementação de padrões como Retry, Timeout e Circuit Breaker para garantir estabilidade.

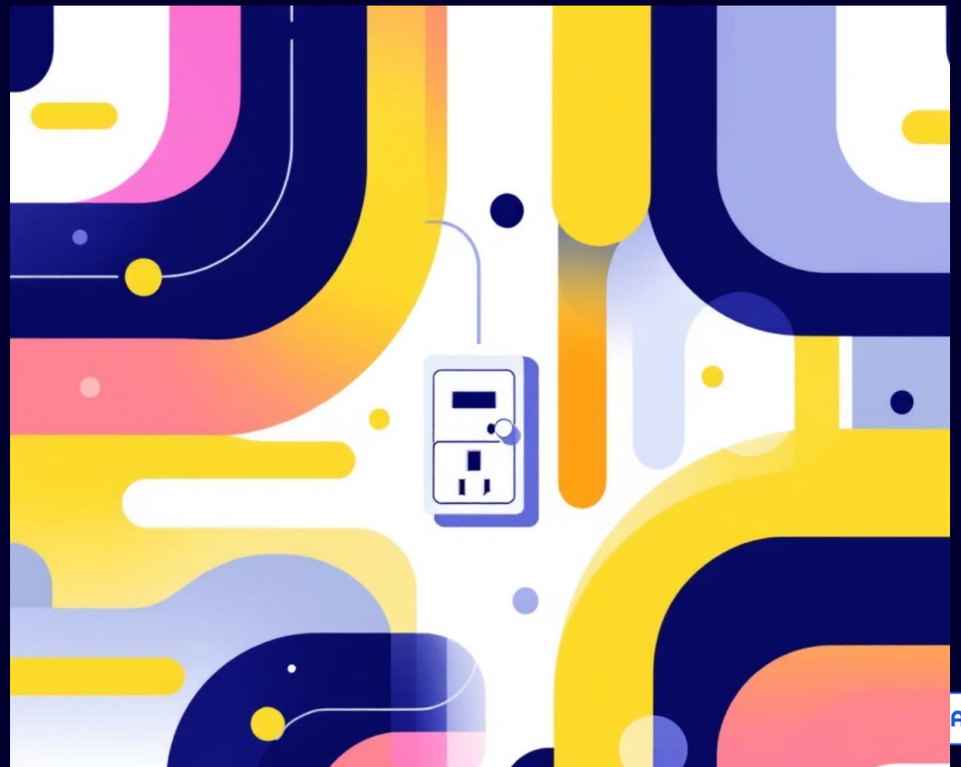
3

Segurança

Autenticação com JWT/OAuth2 e criptografia com TLS para proteger comunicações.

Padrões de Resiliência em Ação

- O sistema tenta novamente (Retry) caso um serviço falhe temporariamente
- Circuit Breaker evita sobrecarga de serviços que estão indisponíveis
- Timeout previne esperas indefinidas por respostas



Evolução Arquitetural

A evolução das arquiteturas de software reflete a necessidade crescente de sistemas mais flexíveis, escaláveis e organizados por domínio de negócio.

1. **SOA (Service-Oriented Architecture)**

   Serviços maiores e mais acoplados, focados em reutilização e padronização de interfaces.

2. **Microserviços**

   Serviços menores, independentes e altamente escaláveis, com responsabilidades bem definidas.

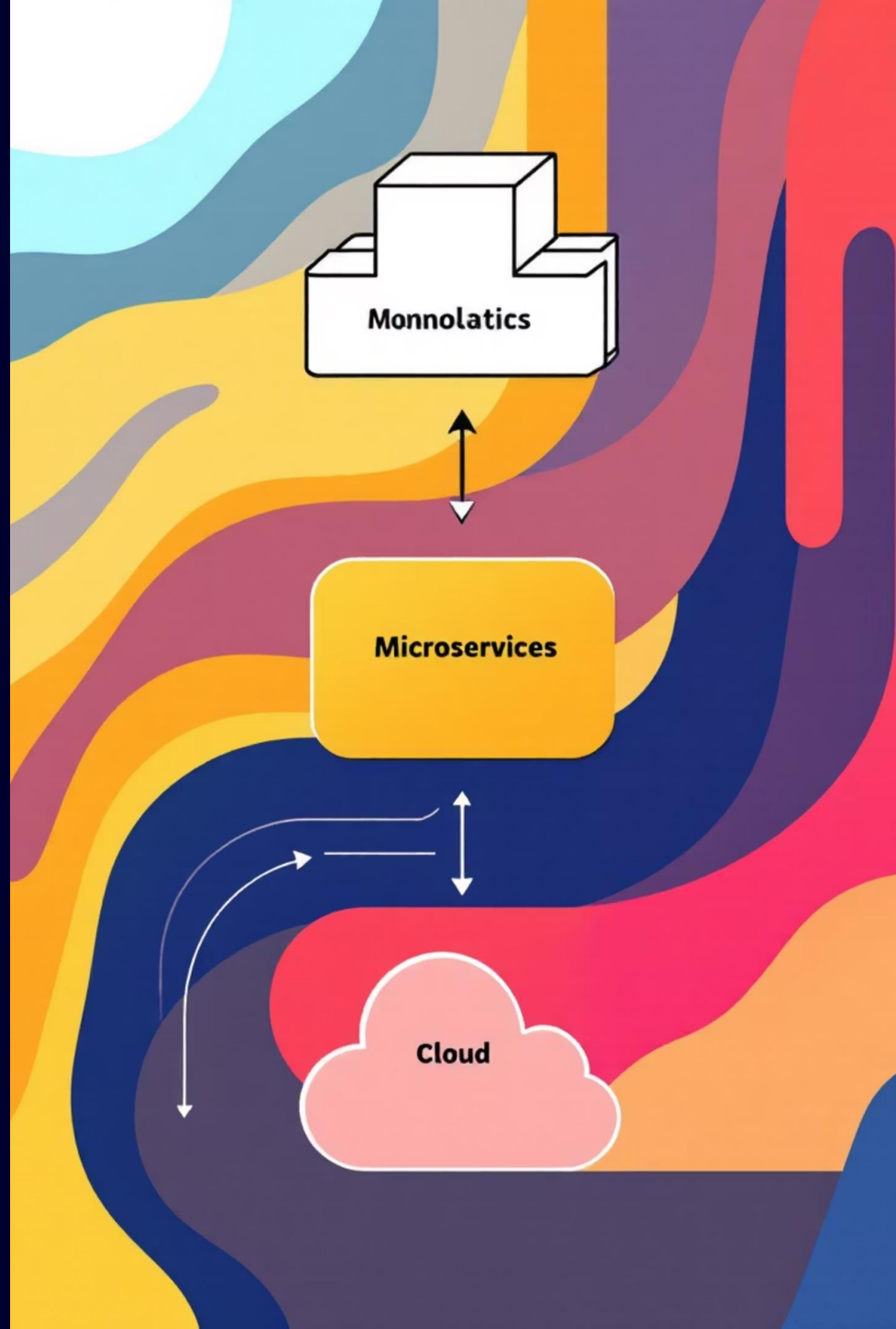
3. **Serverless**

   Execução de funções sob demanda usando plataformas como AWS Lambda e GCP Functions.

4. **DDD (Domain-Driven Design)**

   Organização de serviços com base no domínio de negócio, alinhando tecnologia com estratégia empresarial.

Exemplo em E-commerce: Microserviços divididos em Pagamentos, Estoque e Notificações. Cada serviço é independente, mas trabalha em conjunto para entregar valor ao negócio.



Exemplo Integrador: Sistema de E-commerce

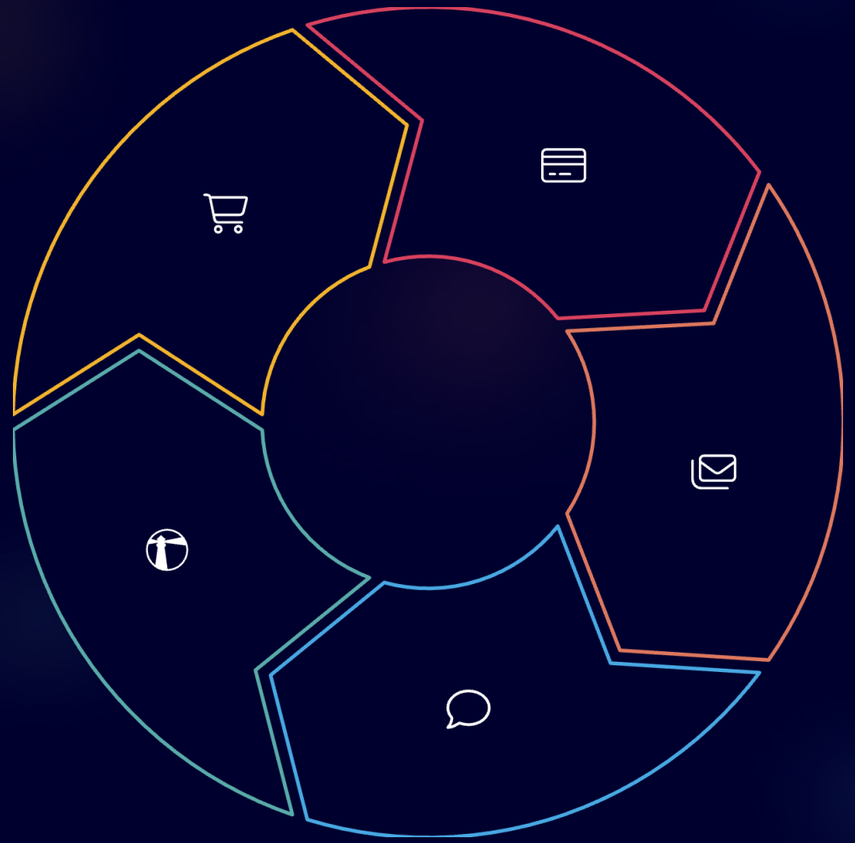
Vamos consolidar todos os conceitos através de um exemplo prático que demonstra como diferentes tecnologias e padrões trabalham em conjunto.

Serviço de Pedidos

Chama Pagamentos via RPC para processar transação

Observabilidade

Prometheus coleta métricas e Jaeger rastreia chamadas



Pagamentos

Aprova pagamento e emite evento "Pagamento Aprovado"

Notificações

Reage ao evento e envia e-mail de confirmação ao cliente

Kubernetes

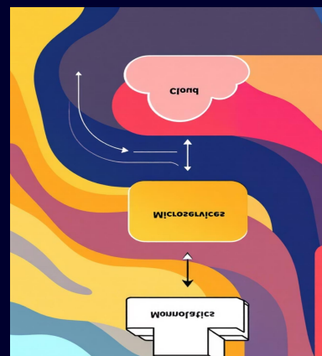
Gerencia escalabilidade automática dos serviços

Tecnologias Integradas

- **RPC:** Comunicação síncrona entre serviços
- **Eventos:** Comunicação assíncrona via mensageria
- **Orquestração:** Kubernetes para gestão de containers

Benefícios Alcançados

- Escalabilidade independente por serviço
- Resiliência através de padrões de falha
- Visibilidade completa do sistema



Referências

Birrell, A. D.; Nelson, B. J. (1984)

Implementing Remote Procedure Calls. ACM. Trabalho seminal que formalizou o conceito de RPC.

Fowler, M.; Lewis, J. (2014)

Microservices: a definition of this new architectural term. martinfowler.com. Definição clássica de microserviços.

Newman, S. (2015)

Building Microservices. O'Reilly. Guia prático para implementação de arquiteturas de microserviços.

Tanenbaum, A. S.; Van Steen, M. (2017)

Distributed Systems: Principles and Paradigms. Pearson. Fundamentos teóricos de sistemas distribuídos.