Certified Tech Developer
The Ultimate Degree

Microsserviços

Arquitetura Monolítica vs. Microsserviços

Para definir o que é um **microsserviço** é importante **compará-lo** com a arquitetura **monolítica**. Vamos ver como eles se diferenciam.

A seguir, veremos graficamente como interagem os serviços e as camadas da aplicação em cada arquitetura.

Comunicação entre microsserviços

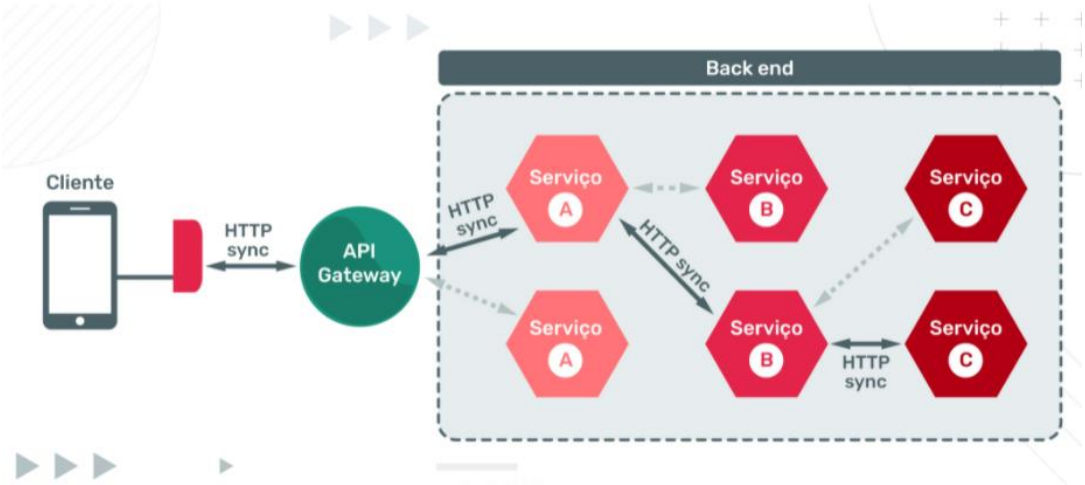
Em uma arquitetura **monolítica**, os componentes se comunicam entre si por meio de chamadas à nível de linguagem.

Na arquitetura de **microsserviços**, os serviços precisam se comunicar usando um **protocolo de comunicação**, como **HTTP** ou **AMQP**.

Não existe uma solução única de comunicação entre os microsserviços. Depende do protocolo e pode ser **síncrona ou assíncrona**.

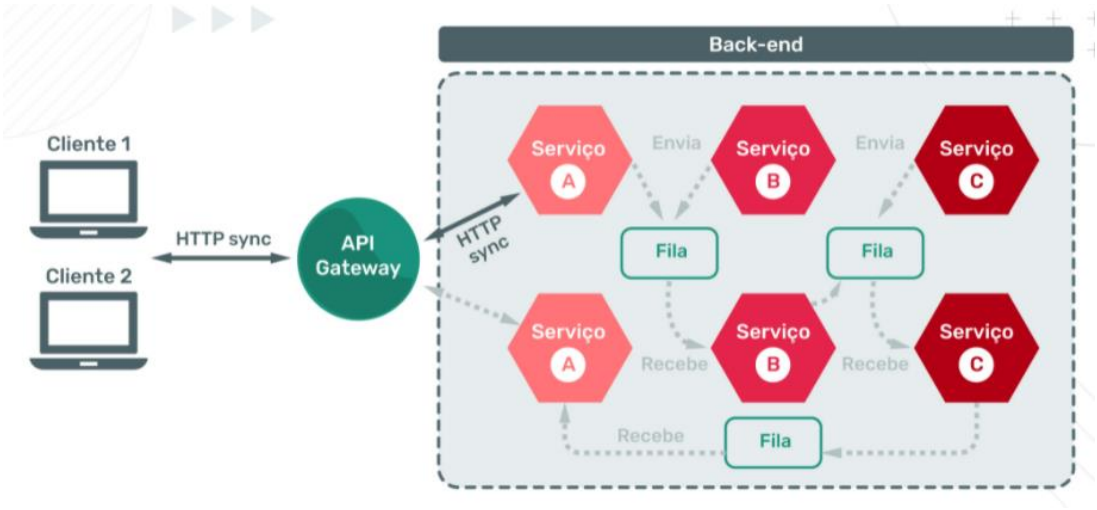
Comunicação síncrona

Neste tipo de comunicação é necessário um endereço de serviço predefinido para enviar a **solicitação (request)**, e ambos (remetente e destinatário da chamada) devem estar em operação neste momento. O cliente só pode continuar sua tarefa ao receber uma **resposta (response)** do servidor. A abordagem de request/response normalmente usa o protocolo HTTP e inclui **REST, GraphQL e gRPC.**



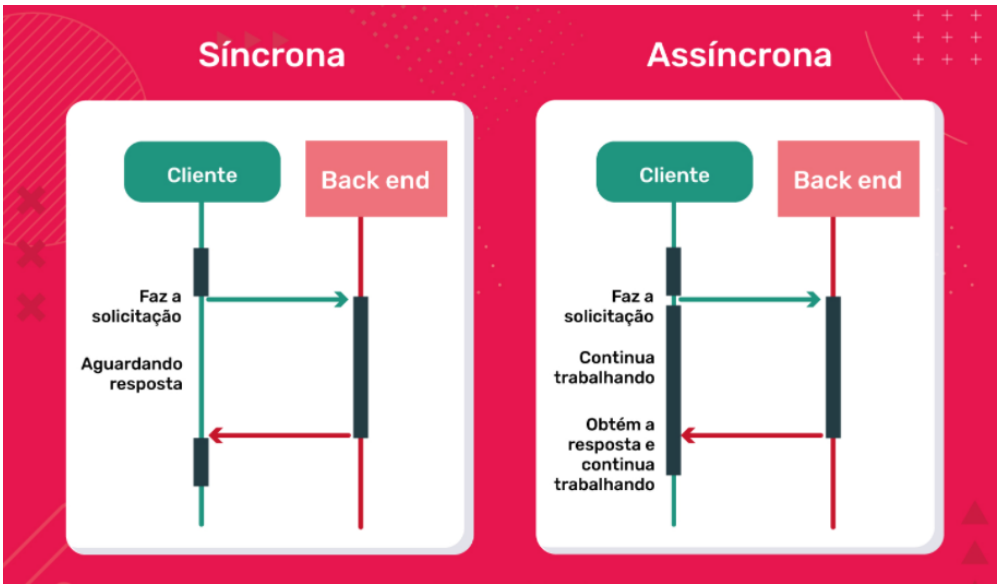
Comunicação assíncrona

Por outro lado, neste tipo de comunicação uma mensagem é enviada para uma **fila** ou agente de mensagens. A mensagem é enfileirada, e no caso do serviço de recebimento estar inativo ele retorna o processamento mais tarde quando estiver ativo. O remetente da mensagem não espera nenhuma resposta. Protocolos assíncronos como **MQTT, STOMP e AMQP** são gerenciados por plataformas como o **Apache Kafka Stream e RabbitMQ**.



Resumindo

Assim, as duas formas de comunicação que acabamos de ver estão representadas na imagem a seguir. A comunicação síncrona, antes de uma solicitação, deve aguardar a resposta do servidor (backend) e a comunicação assíncrona, quando o cliente faz uma solicitação, continua funcionando até que a resposta seja obtida.



	AMQP	MQTT	XMPP	STOMP
Objetivo	Substituição de protocolos proprietários	Mensagens para dispositivos com recursos limitados	Mensagens instantâneas, adotadas para uso mais amplo	Mensagem orientada para middleware
Formato	Binário	Binário	Baseado em XML	Baseado em texto
API	Dividido em classes (> 40 métodos no RabbitMQ)	Simples (5 operações básicas)	Diferentes itens XML com vários tipos	10 comandos básicos
Confiabilidade	Confirmação das transações do editor e do assinante das mensagens	Confirmação de entrega da mensagem	Confirmação de entrega da mensagem e nova tentativa	Confirmação de recebimento e transação do assinante
Segurança	SASL e TLS/SSL	Não permite TLS/SSL	SASL e TLS/SSL	Depende da plataforma de mensagens
Extensibilidade	Pontos de extensão	Nenhum	Extensível	Depende da plataforma de mensagens

O que escolher?

A comunicação **REST/HTTP** funciona para padrões de **Request/Response síncronos**, para arquiteturas orientadas a serviços (SOA) e APIs expostas publicamente.

Algumas **desvantagens** são:

- **Baixo desempenho:** a solicitação (request) não obtém uma resposta até que todas as chamadas internas sejam concluídas, o que pode resultar em tempos de resposta mais lentos. Ele também pode cair se houver muitas chamadas.
- **Perda de autonomia:** se os microsserviços se conectam por HTTP e dependem da resposta de outra pessoa, eles não podem ser totalmente autônomos.
- **Tratamento de falhas complexas:** se houver uma cadeia de chamadas HTTP e um microsserviço intermediário falhar, toda a cadeia falhará. Novas tentativas são realizadas e a cadeia é quebrada.

O padrão **assíncrono** geralmente é recomendado para **comunicação interna entre microsserviços** para reduzir a quantidade de chamadas em cadeia e tornar-se independente do ciclo de Request/Response.