

# Embarcadero Conference 2024

---

Inovação faz parte do nosso DNA!

# Desbravando arquitetura de microsserviços com Delphi

{Arthur Steinbach

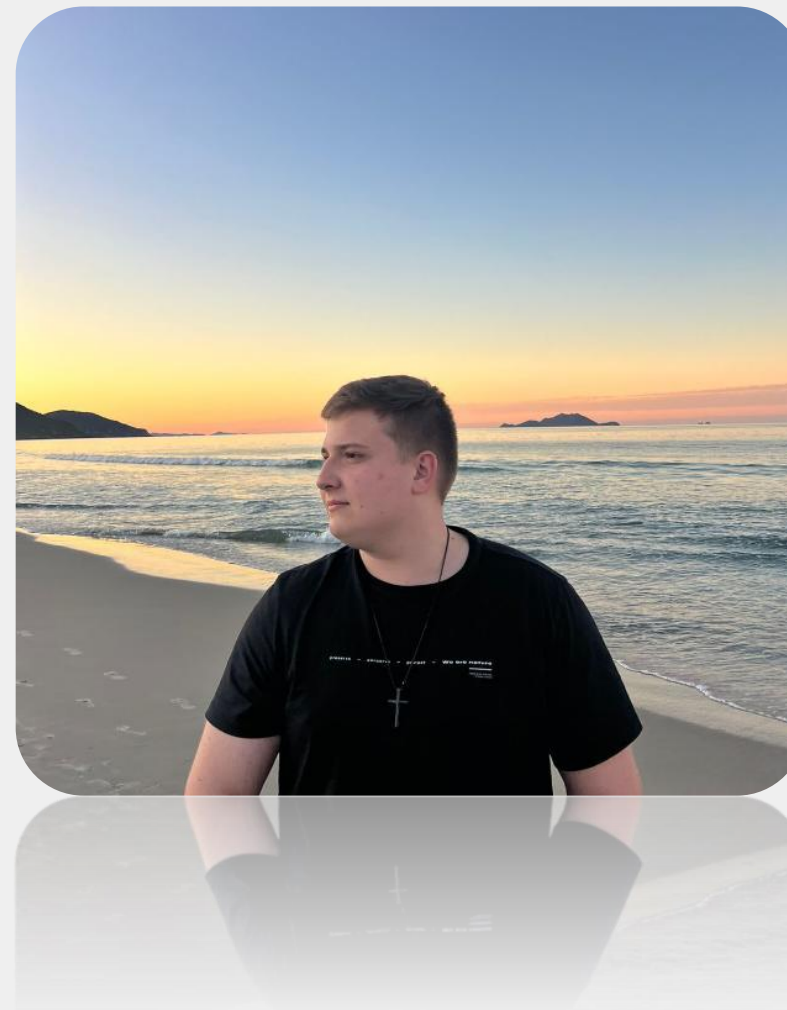
Abordaremos o uso da arquitetura de microsserviços no universo do Delphi, seus conceitos básicos de funcionamento e aplicação, os benefícios no desenvolvimento, como ela pode ser construída, o que são containers e como eles atuam neste modelo de construção de software. Faremos ainda uma comparação, demonstrando as paridades e diferenças entre as principais tecnologias usadas neste meio Apache Kafka e RabbitMQ



# Quem sou eu?

Prazer!

Me chamo Arthur Steinbach, tenho 19 anos, iniciei meus estudos na programação tinha apenas 11 anos e desde lá me apaixonei pela área iniciando na linguagem C com o Arduino e posteriormente conheci o Java até encontrar o Delphi, qual é a minha principal ferramenta de trabalho. Iniciei no Delphi na versão 5 migrando a aplicação pra versão 7 e posteriormente, em um trabalho formal, trabalhei com as versões XE2, 10.4 e 11. Hoje trabalho principalmente com a 11 e 12.



## O que são microserviços ?

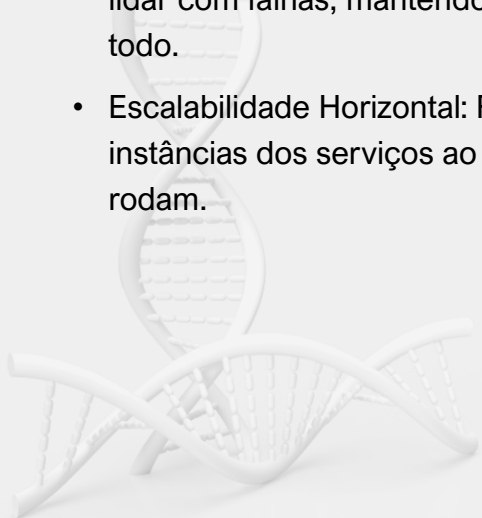
Microserviços são uma abordagem moderna para construir e organizar sistemas complexos. Em vez de criar uma única aplicação monolítica, os microserviços dividem o sistema em pequenas partes independentes. Cada uma dessas partes, ou 'serviços', realiza uma tarefa específica e se comunica com as outras para formar o sistema completo. Essa estrutura modular permite desenvolver, testar, e escalar cada serviço separadamente, oferecendo mais flexibilidade, agilidade e resiliência ao sistema como um todo.



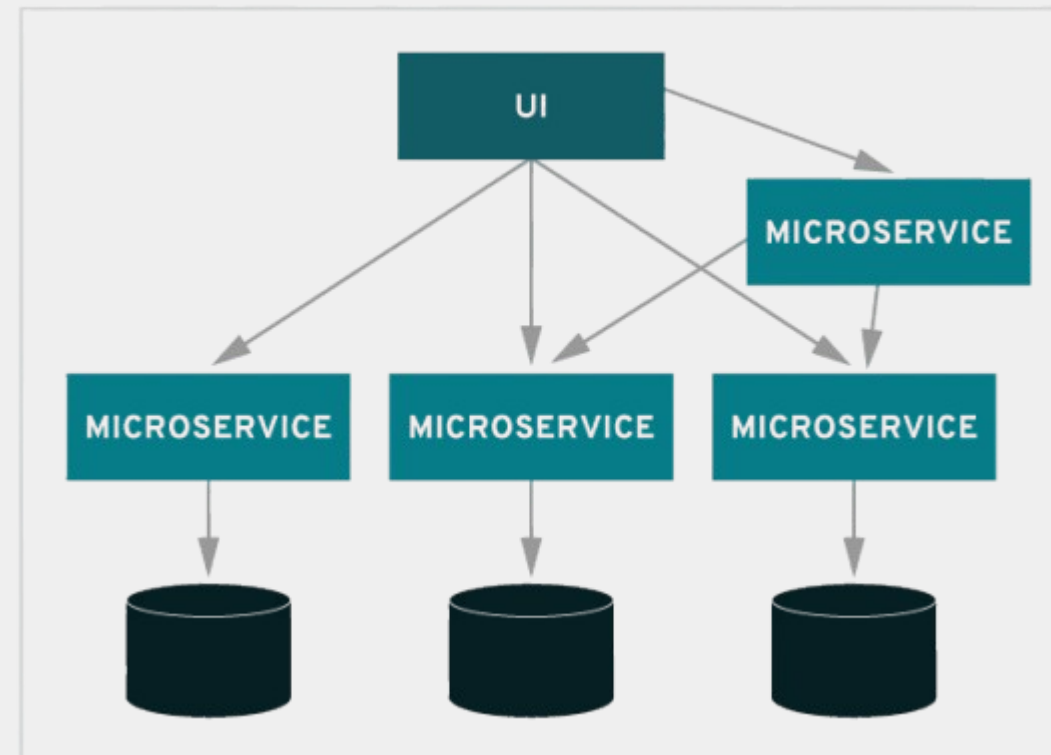
# MICROSSERVIÇOS

## Quais suas conceitos básicos?

- Independência de Serviços: Cada serviço é desenvolvido, implantado e escalado de forma independente.
- Comunicação através de APIs: Os serviços se comunicam entre si usando APIs, geralmente através de HTTP/REST ou mensageria.
- Descentralização de Dados: Em vez de um banco de dados monolítico, cada serviço pode ter seu próprio banco de dados.
- Comunicação através de mensageria: As informações podem ser passadas de um serviço para o outro por meio de Message Broker's;
- Resiliência e Tolerância a Falhas: Os serviços são projetados para lidar com falhas, mantendo a funcionalidade do sistema como um todo.
- Escalabilidade Horizontal: Facilita a escalabilidade adicionando mais instâncias dos serviços ao invés de reforçar a máquina em que eles rodam.



## MICROSERVICES







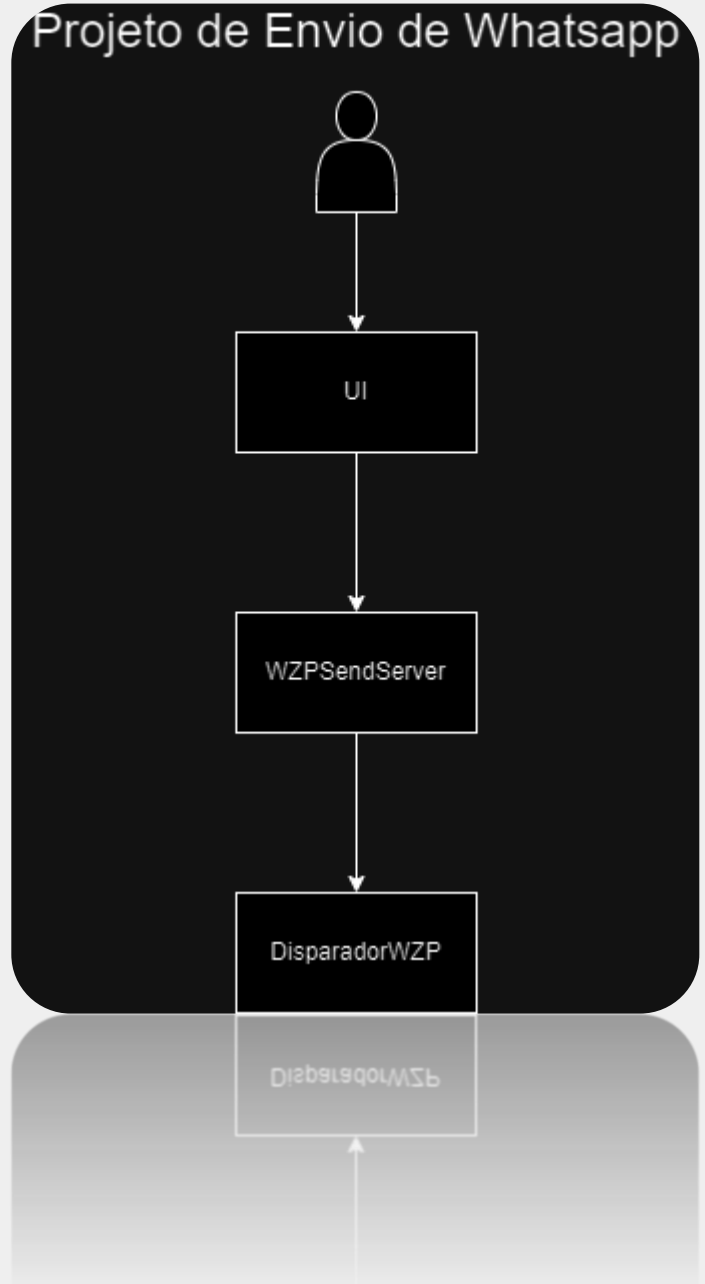
## Quais os benefícios de trabalhar com esta arquitetura ?

- Escalabilidade facilitada: Cresça conforme sua demanda.
- Independência de componentes: Mantenha e atualize partes do sistema sem dores de cabeça.
- Flexibilidade tecnológica: Use a ferramenta certa para cada serviço.
- Agilidade na entrega: Libere novas funcionalidades mais rápido.
- Resiliência: Falhas em um serviço não derrubam o sistema inteiro.

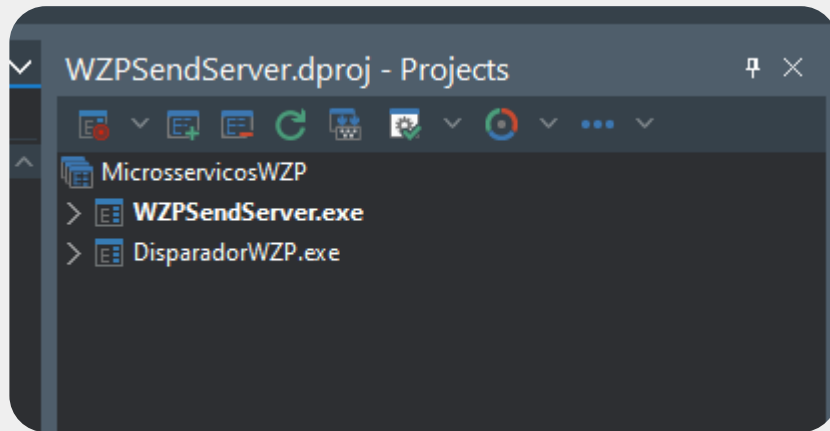


## Mas como podemos construir uma aplicação com microsserviços ?

Ex: Serviço de envio de whatsapp, criaremos uma API utilizando a tecnologia Horse que receberá a demanda de envio e repassará a responsabilidade de enviar para o disparador de whatsapp.



## E na prática, como isso funciona ?



```
procedure TControllerMensagens.DoPostEnviarMensagem(pRequest: THorseRequest; pResponse: THorseResponse);
begin
    var lDmGlobal := TDmGlobal.Create(nil);
    try
        var lJSONBody := pRequest.Body<TJSONObject>;
        var lDestino: string;
        var lMensagem: string;

        if not lJSONBody.TryGetValue<string>('destino', lDestino) or
           not lJSONBody.TryGetValue<string>('mensagem', lMensagem) then
        begin
            pResponse.Send('Falta argumentos!').Status(THTTPStatus.BadRequest);
        end else
        begin
            lDmGlobal.EnviaMensagem(lDestino, lMensagem);
            pResponse.Send('Mensagem Enviada para a Fila!').Status(THTTPStatus.Created);
        end;
    finally
        lDmGlobal.Free;
    end;
end;

class procedure TControllerMensagens.PostEnviarMensagem(pRequest: THorseRequest; pResponse: THorseResponse);
begin
    var lInstancia := Self.Create;
    try
        lInstancia.DoPostEnviarMensagem(pRequest, pResponse);
    finally
        lInstancia.Free;
    end;
end;

class procedure TControllerMensagens.Registrar;
begin
    THorse.Post('/enviarmensagem', Self.PostEnviarMensagem);
end;

end;
```





## Mais um pouco de código

```
procedure TDMGlobal.EnviaMensagem(const pDestino, pMensagem: string);
begin
    qryInsertMensagem.Active := False;
    qryInsertMensagem.SQL.Clear;
    qryInsertMensagem.SQL.Add('insert into envia_whats ');
    qryInsertMensagem.SQL.Add('(destino, mensagem, status) values ( ');
    qryInsertMensagem.SQL.Add(':destino, :mensagem, :status)');
    qryInsertMensagem.ParamByName('destino').AsString := pDestino;
    qryInsertMensagem.ParamByName('mensagem').AsString := pMensagem;
    qryInsertMensagem.ParamByName('status').AsByte := TStatusMensagem.Pendente.ConverterParaInt;
    qryInsertMensagem.ExecSQL;
end;
```

```
begin
    qryInsertMensagem.ExecSQL;
    qryInsertMensagem.ParamByName('status').AsByte := TStatusMensagem.Pendente.ConverterParaInt;
end;
```



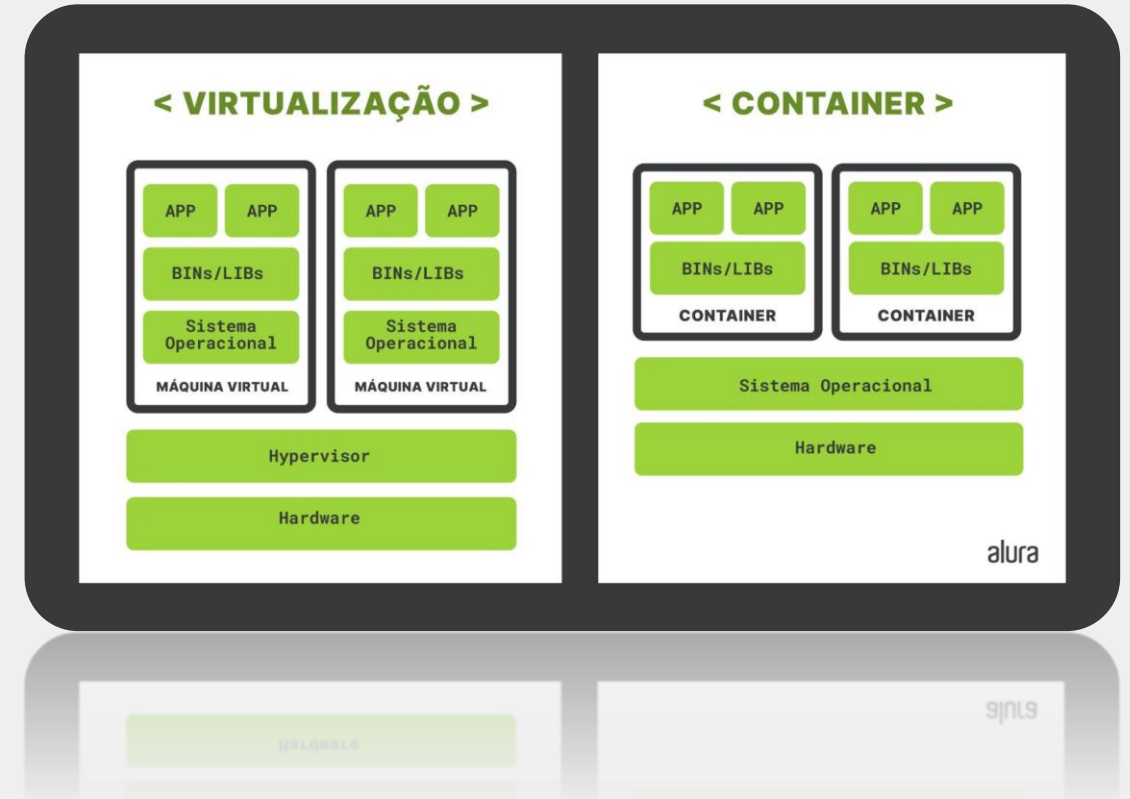
```
procedure TFPrincipal.LerFilaEDisparar;  
begin  
  var lDmGlobal := TDmGlobal.Create(Self);  
  try  
    var lListaEnvio := lDmGlobal.RetornarListaEnvio;  
    for var lItemLista in lListaEnvio do  
      begin  
        WPPConnect.SendTextMessageEx(lItemLista.Destino, lItemLista.Mensagem, 'createChat: true');  
      end;  
    finally  
      lDmGlobal.Free;  
    end;  
  end;  
end;
```

```
cuq?  
cuq?  
TDmGlobal.Free;
```



# O que são containers ?

Containers são unidades leves e independentes que embalam uma aplicação juntamente com todas as suas dependências necessárias como bibliotecas, binários e arquivos de configuração em um único pacote. Isso garante que a aplicação funcione de maneira consistente em qualquer ambiente, seja no desenvolvimento, teste ou produção.



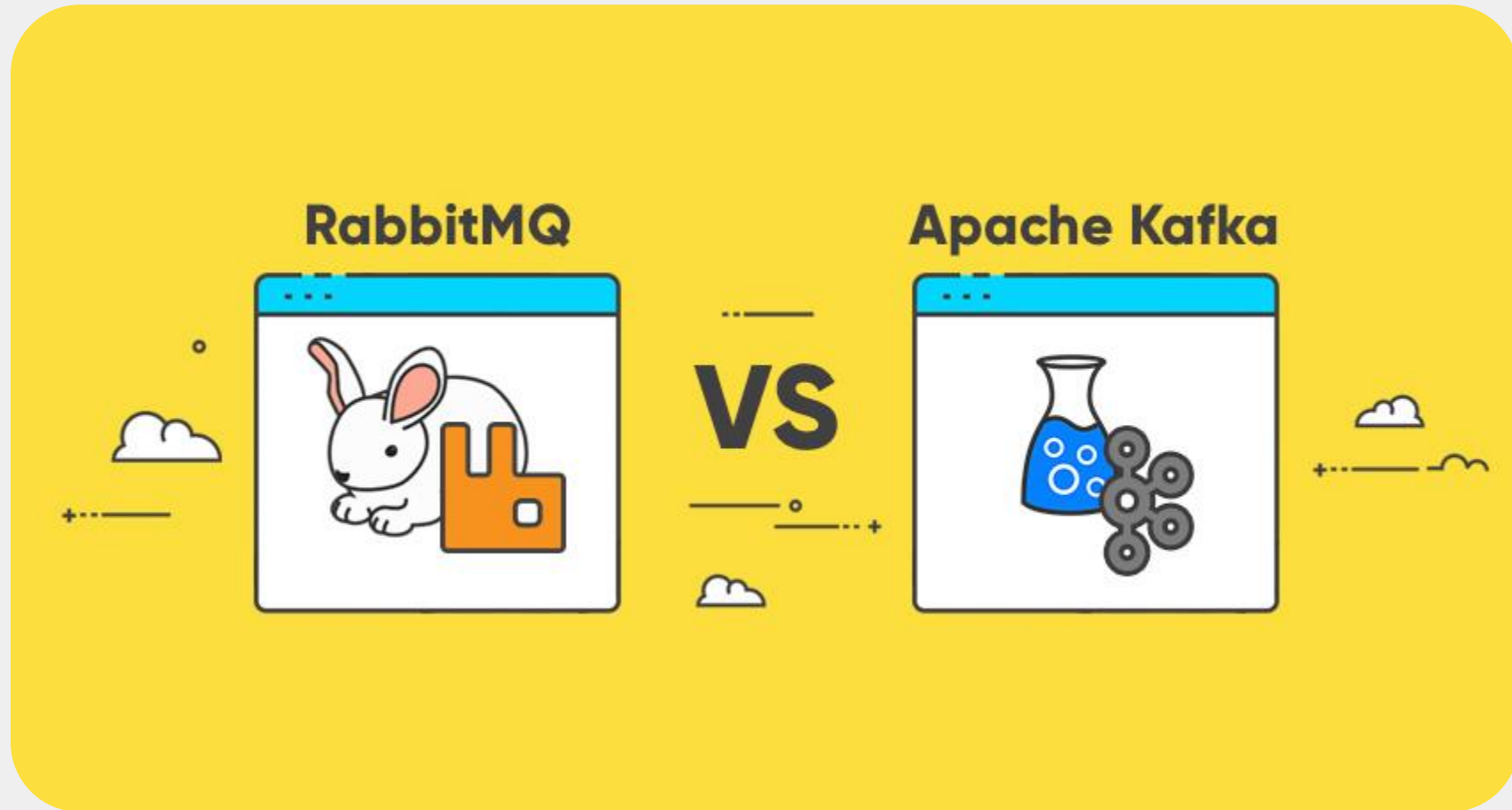
## Qual a atuação do container na arquitetura de microsserviços?



No contexto de microsserviços, os containers são fundamentais, pois permitem que cada serviço seja isolado e gerenciado de forma independente. Isso facilita o desenvolvimento, a implantação e a escalabilidade das aplicações, além de simplificar a manutenção e as atualizações. Com containers, é possível rodar múltiplos microsserviços em diferentes ambientes, sem se preocupar com incompatibilidades ou conflitos de dependências.



# RabbitMQ X Apache Kafka



## O que é RabbitMQ?

RabbitMQ é uma ferramenta essencial para comunicação entre serviços em uma arquitetura de microsserviços. Funciona como um intermediário de mensagens, garantindo que os dados sejam enviados e recebidos de maneira segura e eficiente entre diferentes partes do sistema. Com o RabbitMQ, cada serviço pode se comunicar de forma assíncrona, sem precisar esperar pela resposta imediata do outro serviço, o que melhora a escalabilidade e a resiliência da aplicação.





## O que é Apache Kafka?



Apache Kafka é uma plataforma de streaming distribuída que atua como um sistema de mensagens de alta performance. Na arquitetura de microsserviços, ele serve como um backbone de comunicação, permitindo que diferentes microsserviços troquem informações em tempo real de forma escalável e resiliente. Kafka é ideal para cenários em que o fluxo contínuo de dados é essencial, como processamento de eventos, integração de sistemas e monitoramento.



# Diferenças

## RabbitMQ

- Tipo: Message broker tradicional;
- Arquitetura: RabbitMQ utiliza um modelo de fila, onde as mensagens são enviadas para filas e, em seguida, consumidas pelos consumidores. Ele suporta vários padrões de mensagens, como filas ponto a ponto, pub/sub, e filas com roteamento;
- Persistência: As mensagens podem ser persistentes ou transitórias. O foco principal é a entrega garantida de mensagens, com suporte para confirmação de entrega;
- Casos de Uso: RabbitMQ é geralmente usado para tarefas como filas de trabalho, integração de sistemas com diferentes tecnologias, comunicação entre microserviços, e distribuição de mensagens.

## Apache Kafka

- Tipo: Plataforma de streaming distribuída;
- Arquitetura: Kafka é baseado em um modelo de publicação e assinatura (pub/sub) onde os dados são armazenados em tópicos. Os produtores escrevem dados em tópicos, e os consumidores lêem desses tópicos. O Kafka é altamente escalável e foi projetado para lidar com grandes volumes de dados e alta taxa de transferência;
- Persistência: Os dados são armazenados de forma persistente em disco e podem ser retidos por longos períodos, permitindo o reprocessamento das mensagens;
- Casos de Uso: Kafka é ideal para o processamento de eventos em tempo real, pipelines de dados, integração de dados entre sistemas e sistemas de streaming.



## E no universo do Delphi? O que posso usar pra botar em prática?

Frameworks que podem ser usados

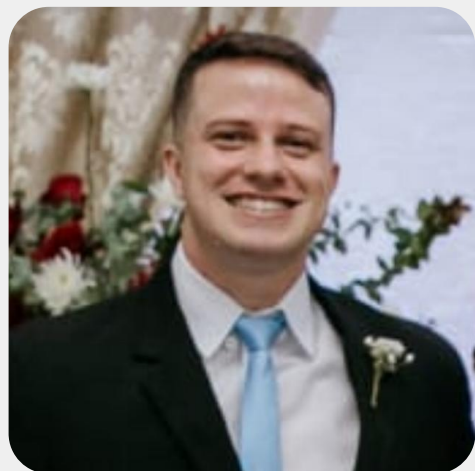
- StompClient – Conexão com o RabbitMQ
- Horse – Construção de API REST
- DelphiKafkaClient – Conexão com Apache Kafka



# Stomp



# Agradecimentos



# Embarcadero Conference 2024

Inovação faz parte do nosso DNA!





Quer me ver na  
**#ECON25?**  
Acesse o QRCode  
e avalie minha palestra!




**Arthur Steinbach**

 [@arthurst.dev](https://github.com/arthurst.dev)

 [arthur-steinbach](https://www.linkedin.com/in/arthur-steinbach)

 [stein.arth13@gmail.com](mailto:stein.arth13@gmail.com)

 [\(48\) 9 8497 0952](tel:(48)984970952)

