INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Publish–Subscribe

Grupo 2 & Grupo 3

2023

INSTITUTO DE INFORMÁTICA UFG

UFG UNIVERSIDADE FEDERAL DE GOIÁS

Sumário

Introdução

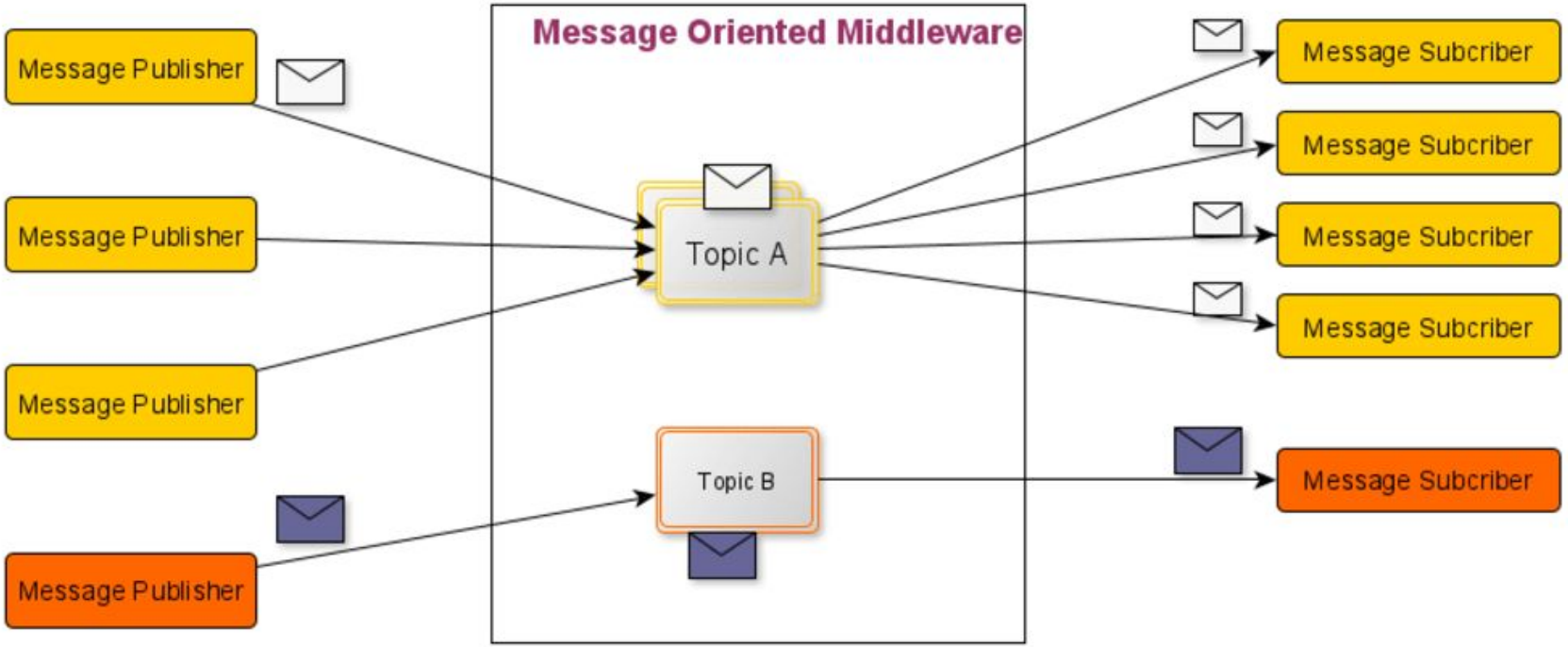
Publish-Subscribe: Visão geral

- O padrão Publish-Subscribe, também conhecido como Pub-Sub, é um modelo de comunicação amplamente utilizado em sistemas distribuídos.
- Ele oferece uma maneira flexível de trocar informações entre diferentes componentes do sistema, onde os participantes podem se comunicar sem conhecer uns aos outros diretamente.
- No Pub-Sub, os eventos ou mensagens são publicados em tópicos específicos, e os componentes interessados em receber essas mensagens se inscrevem nesses tópicos.

Publish-Subscribe: Visão geral

- Componentes essenciais do padrão Publish-Subscribe:
  - **Publicadores (publishers)**: responsáveis por enviar mensagens ou eventos para o sistema de Pub-Sub. Eles identificam o tópico relevante e publicam suas mensagens nele.
  - **Assinantes (subscribers)**: registram nos tópicos de interesse, passando a receber todas as mensagens enviadas para esse tópico específico.
  - **Broker de mensagens**: atua como um intermediário entre os publicadores e os assinantes. Quando um publicador envia uma mensagem para um tópico, o broker de mensagens recebe essa mensagem e a distribui para todos os assinantes registrados neste tópico.

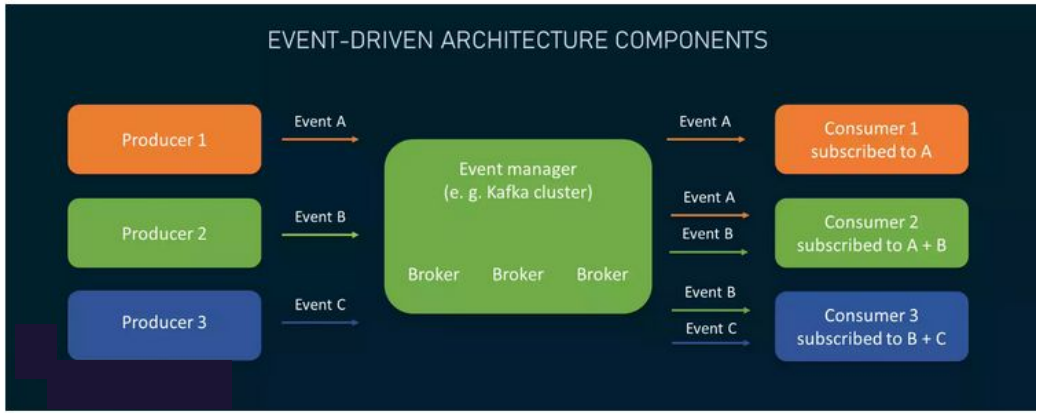
Publish-Subscribe: Visão geral



Arquitetura

Arquitetura Pub/Sub

- Arquitetura visa a escalabilidade horizontal
- Segue a estrutura básica de sistemas baseados em eventos
- Componentes: Publicadores, Assinantes e Broker



Arquitetura Pub/Sub

- Para escalabilidade:
  - Plano de dados: gerencia a movimentação de mensagens entre publicadores e assinantes
  - Plano de controle: gerencia a atribuição de editores e assinantes a servidores no plano de dados
- Esse separação garante que qualquer upgrade na rede não afeta quem já participa dela
- *Heterogeneidade*: promove a cooperação de componentes do sistema distribuído que inicialmente não foram projetados para funcionarem juntos

Vantagens

Vantagens

- Desacoplamento
  - O Pub-Sub permite um acoplamento fraco entre os componentes do sistema.
  - Os publicadores não precisam ter conhecimento prévio dos inscritos e vice-versa.
  - Isso facilita a escalabilidade e a manutenção do sistema.
- Assincronismo
  - Os publicadores podem enviar mensagens a qualquer momento, e os inscritos as recebem assim que estiverem disponíveis.
  - Útil em situações em que os componentes têm velocidades de processamento diferentes.

Vantagens

- Escalabilidade
  - O Pub-Sub permite adicionar ou remover facilmente novos publicadores e inscritos conforme necessário, permitindo que o sistema seja distribuído em vários servidores ou nós.
- Flexibilidade
  - Os publicadores publicam mensagens em tópicos ou canais específicos, não comunicando diretamente com os Subscribers, os Subscribers, por sua vez, podem se inscrever em tópicos de seu interesse e receber as mensagens relevantes. Essa abordagem desacoplada permite que os componentes do sistema sejam independentes uns dos outros, facilitando a manutenção e evolução do sistema.

Casos de uso

Casos de uso

- Semáforos inteligente com Pub-Sub
  - Pub-Sub permite que os semáforos inteligentes compartilhem informações com os sistemas de controle de tráfego de maneira **eficiente e assíncrona**.
  - Os semáforos inteligentes podem publicar informações relevantes sobre o tráfego e, assim, os sistemas de controle de tráfego se inscrevem.
- Coordenação de Cruzamentos para Veículos Autônomos
  - O Pub-Sub possibilita a **comunicação assíncrona** entre os semáforos inteligentes e os veículos autônomos.
  - Similar ao primeiro exemplo, mas quem se inscreve são os veículos autônomos.

Casos de uso

- Eventos de jogos com notificação em tempo real

O modelo Pub-Sub é bastante utilizado para fornecer notificações em tempo real aos jogadores durante eventos de jogos. Quando um evento significativo ocorre no jogo, o servidor do jogo publica a mensagem relevante para um broker centralizado, que depois, encaminha a mensagem para todos os jogadores subscritos que estão interessados nesse tipo específico de evento.

- Salvamento de jogos

O servidor do jogo age como um publicador, enviando eventos de salvamento contendo os dados relevantes para o broker. Quando um jogador deseja salvar seu progresso no jogo ou sincronizá-lo em outro dispositivo, o cliente envia uma solicitação para o servidor, que em resposta publica um evento de salvamento no broker

Implementação

Publisher - .NET

```
using RabbitMQ.Client;

var factory = new ConnectionFactory()
{
        HostName = "localhost"
};

using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

var exchangeName = "Publish-Subscribe";

channel.ExchangeDeclare(exchange: exchangeName, type: ExchangeType.Fanout);
...
```

Publisher - .NET

```
...
while (true)
{
        var message = "";
        do
        {
                Console.Write("Publique uma mensagem: ");
                message = Console.ReadLine();
        }
        while (string.IsNullOrWhiteSpace(message));

        var body = System.Text.Encoding.UTF8.GetBytes(message);
        channel.BasicPublish(exchange: exchangeName, routingKey: "", basicProperties:
null, body: body);
        Console.WriteLine("Mensagem publicada!");
}
```

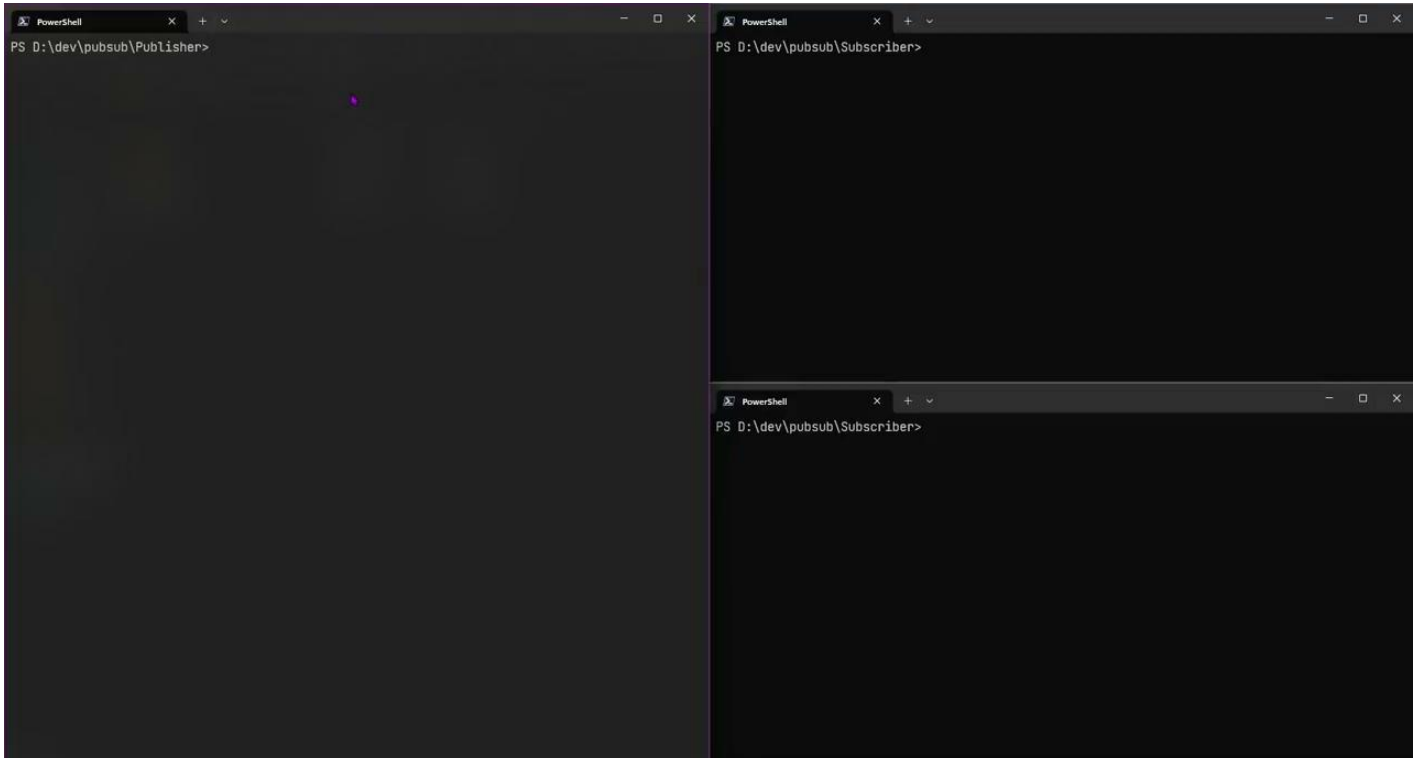
Subscriber - .NET

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

var factory = new ConnectionFactory()
{
        HostName = "localhost"
};

using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

var exchangeName = "Publish-Subscribe";
var queueName = channel.QueueDeclare().QueueName;

channel.ExchangeDeclare(exchange: exchangeName, type: ExchangeType.Fanout);
…
```

Subscriber - .NET

```
...
channel.QueueBind(queue: queueName, exchange: exchangeName, routingKey: "");

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (sender, eventArgs) =>
{
        var message = System.Text.Encoding.UTF8.GetString(eventArgs.Body.ToArray());
        Console.WriteLine($"Mensagem recebida: {message}");
};

channel.BasicConsume(queue: queueName, autoAck: true, consumer: consumer);

Console.WriteLine("Subscriber " + queueName + " está aguardando por mensagens...");
Console.ReadLine();
```

Execução - .NET



Obrigado!

Dúvidas ou sugestões?