

A photograph of a person's hands holding a smartphone, overlaid with a semi-transparent blue filter. The phone's screen shows a keyboard and some text. The words 'publish' and 'subscribe' are prominently displayed in white text on a blue rectangular background.

# **publish subscribe**

Bianca da Rocha Bartolomei  
Carolina Vasques Moreira  
Luís Otávio Malta Conceição

# índice

1. [Introdução](#) - Comunicação indireta
2. Sistema [publicar/assinar](#) - O que é
3. Modelos de [inscrição](#)
4. [Questões](#) de implementação
5. Arquitetura do [sistema](#)



# comunicação indireta

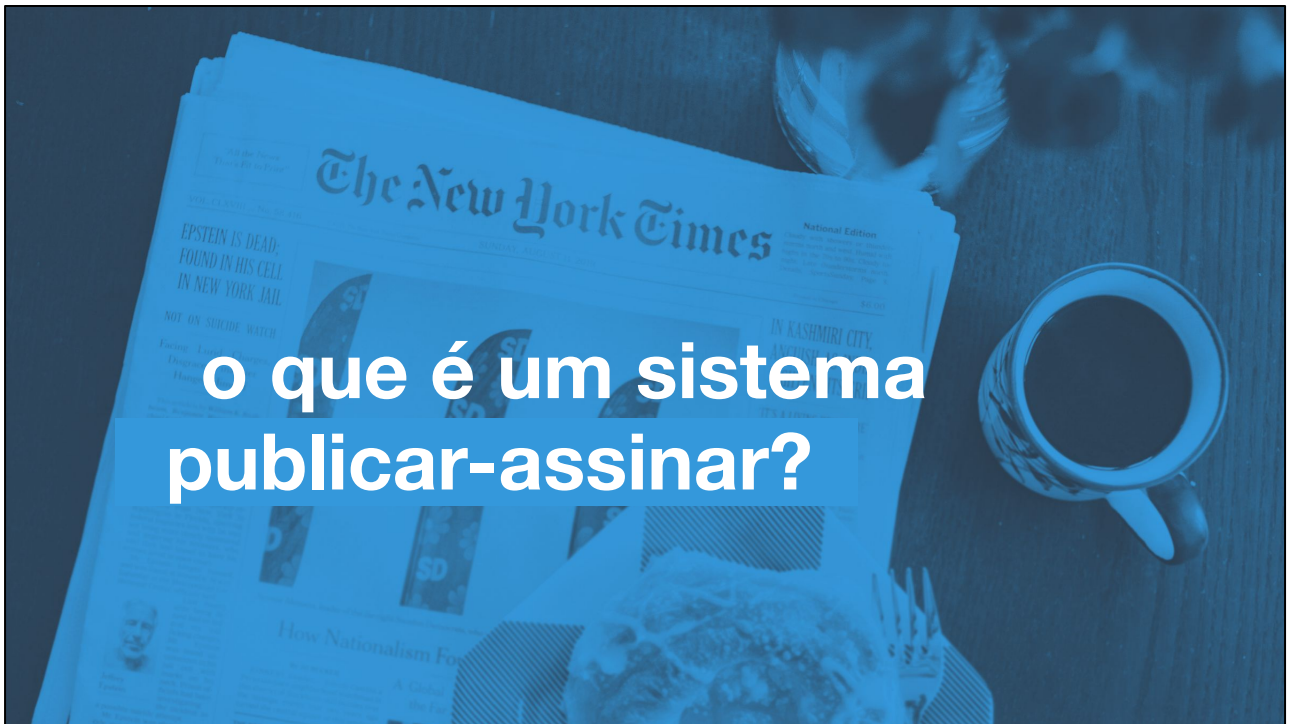
**Introdução - o que é comunicação indireta:** tipo de comunicação em que não há ligação direta entre emissores e receptores; em vez disso, existe uma **entidade intermediária (broker)** que é responsável pela comunicação entre eles.

## comunicação indireta

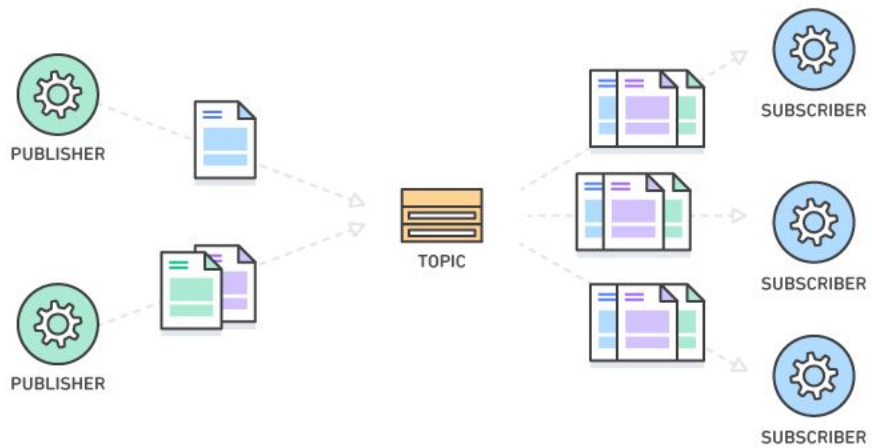
Existe uma entidade intermediária (**broker**) que é responsável pela comunicação entre o remetente e o destinatário; isto permite um alto grau de **desacoplamento** entre eles.

- Desacoplamento espacial
- Desacoplamento temporal

Isto permite um alto grau de desacoplamento entre os remetentes e os destinatários, uma vez que: 1) o remetente não precisa saber para quem está enviando, e vice-versa (**desacoplamento espacial**); o remetente e o destinatário não precisam estar conectados ao mesmo tempo, pois o broker é quem irá guardar e transmitir a mensagem (**desacoplamento temporal**) - *diferentemente do cliente-servidor tradicional, em que não é possível enviar e receber mensagem sem que ambos existam ao mesmo tempo*. Este tipo de comunicação é muito útil em sistemas que os destinatários podem ser **desconhecidos e mudar o tempo todo** (por exemplo, usuários que se conectam e desconectam rapidamente). Uma das principais técnicas de comunicação indireta é o **publish/subscribe**.



Também conhecidos como sistemas baseados em eventos distribuídos, sistemas publish/subscribe seguem um padrão em que **publicadores disseminam eventos para assinantes** que se inscrevem para recebê-los. O **broker** é responsável por fazer com que as mensagens dos publicadores cheguem para os assinantes corretos, realizando uma **filtragem** de acordo com o **modelo de inscrição** utilizado. O padrão publish/subscribe possui duas características principais: é **assíncrono** (o publisher e o subscriber não ficam bloqueados aguardando a resposta do outro; ambos estão livres para continuar seu processamento, permitindo que diferentes usuários fiquem ativos em diferentes momentos sem comprometer o funcionamento do sistema) e **heterogêneo** (permite a comunicação até mesmo de componentes de um sistema distribuído que não foram projetados para trabalharem juntos, podendo ser utilizado para aplicações em IoT). Um exemplo de aplicação do padrão publish/subscribe é o **feed RSS**, que permite que os usuários se inscrevam em blogs ou sites de notícias para serem notificados de novas publicações.



Fonte da imagem: [aws.amazon.com/pt/pub-sub-messaging/](https://aws.amazon.com/pt/pub-sub-messaging/)



# modelos de inscrição

Existem várias formas de direcionar uma mensagem publicada ao assinantes interessados, essas formas são conhecidas como modelos de inscrição. Abaixo são listados alguns dos modelos mais populares e destacados suas características mais importantes.

# modelos de inscrição

Formas de mapear uma determinada mensagem de um **publisher** ao **subscriber** interessado:



CANAL



TÓPICO



CONTEÚDO



TIPO



CONTEXTO



CONCEITO

## Baseado em canal:

- Define um canal físico nomeado para cada categoria de evento publicado pelos Publishers, o qual um Subscriber pode se inscrever para obter as informações.
- Primitivo, sendo o único que define um canal físico
- Usado no COBRA Event Service

## Baseado em tópico:

- A mensagem publicada possui um campo que define a qual categoria de conteúdo ela pertence.
- Aceita hierarquia, separando tópicos e subtópicos que podem ser inscritos pelos Subscribers.
- Mais popular, mas com expressividade limitada

## Baseado em conteúdo:

- As mensagens dos Publisher tem seu conteúdo filtrado a partir de restrições dos Subscribers. Generalização dos tópicos.
- Os dados publicados são mais estruturados
- Filtra eventos inúteis aos assinantes

## Baseado em tipo:

- Ligado à ideia de objetos. A filtragem pode se dar desde o tipo do objeto até seus atributos ou métodos.
- Fácil de integrar com a linguagem de programação.
- Baixa granularidade permite consultas mais refinadas (métodos e atributos)

## Baseado em contexto:

- Leva em conta aspectos das circunstâncias físicas do ambiente onde o sistema está inserido.
- Ex: Filtragem do conteúdo de uma certa localização
- Importante para computação móvel e ubíqua.



**Baseado em conceito:**

- Leva em conta o significado da mensagem
- Filtros relacionados à semântica e sintaxe



```
32 self.fingerprints = set()
33 self.logdups = True
34 self.debug = debug
35 self.logger = logging.getLogger(__name__)
36 if path:
37     self.file = open(os.path.join(path, 'requests.log'), 'a')
38     self.file.seek(0)
39     self.fingerprints.update(requests_log)
40
41 @classmethod
42 def from_request(cls, request):
43     return cls(request)
44
45 def request_seen(self, request):
46     fp = self.request_fingerprint(request)
47     if fp in self.fingerprints:
48         return True
49     self.fingerprints.add(fp)
```

# questões de implementação

# questões de implementação

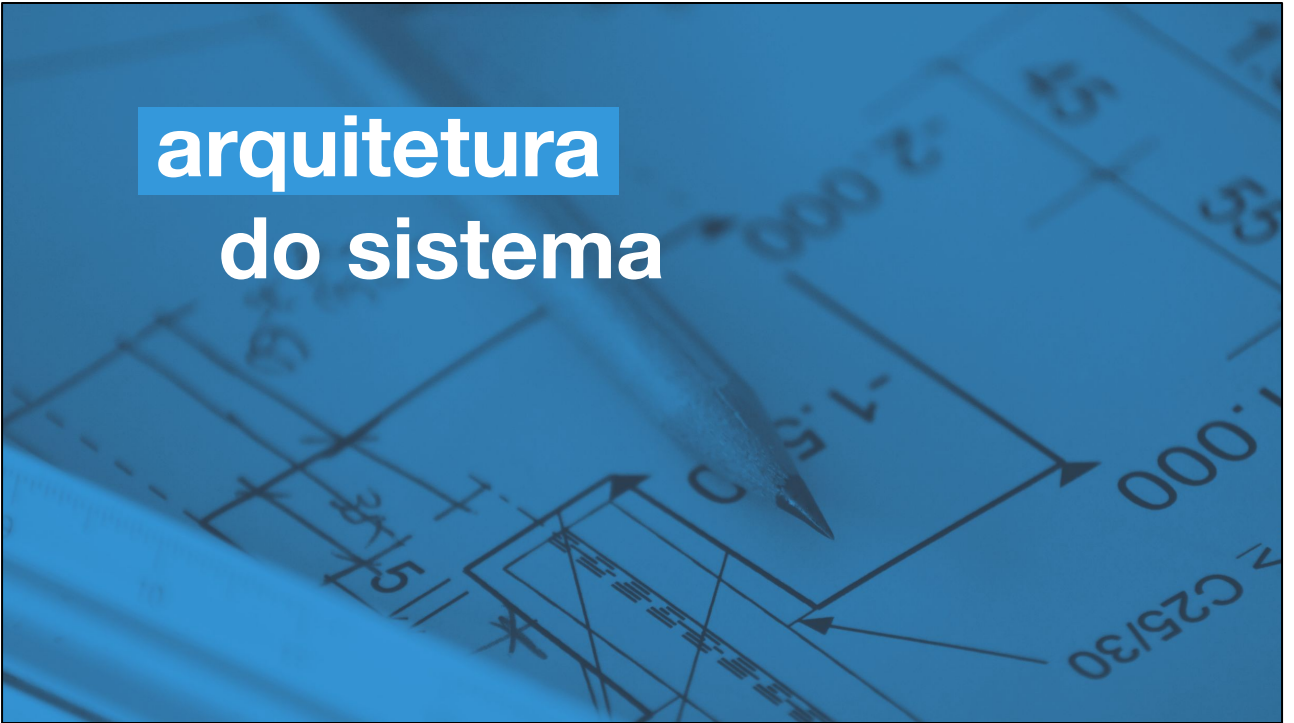
- Implementações centralizadas
  - Gargalo e ponto único de falha
- Implementações distribuídas
  - Rede de brokers ou P2P
  - Roteamento de mensagens:
    - Inundação
    - Filtragem
    - Rendez-vous
    - Anúncios

Existem várias arquiteturas para a implementação sistemas publicar-assinar. A mais simples é a implementação centralizada em um único nó, em que o servidor desse nó atuar como broker (intermediário de evento). O problema dessa estratégia é que ela não é flexível nem escalável, já que o broker pode representar um ponto de único de falha e gargalo para desempenho. Como alternativa, existem as implementações distribuídas, que podem ser um rede de intermediários ou peer to peer. Para o caso P2P, não há distinção entre publicadores, assinantes e intermediários, todos os nós atuam como intermediários, implementando cooperativamente a funcionalidade de roteamento de evento exigida. O roteamento em uma implementação distribuída quando baseada em conteúdo pode ser feita de quatro formas:

- Inundação: é a estratégia mais simples. Consiste em enviar uma notificação de evento para todos os nós da rede e, então, realizar a correspondência apropriada na extremidade assinante.
- Filtragem: os intermediários encaminham as notificações pela rede somente onde há um caminho para um assinante válido. Isso é obtido pela propagação de informações de assinatura para os publicadores em potencial, seguido do armazenamento do estado associado em cada intermediário.
- Anúncios: a estratégia baseada em filtragem pode gerar muito tráfego, devido à propagação de assinaturas, com as assinaturas basicamente usando uma estratégia de inundação de volta para todos os possíveis publicadores. Nos sistemas com anúncios, essa carga pode ser reduzida por se propagar os anúncios para os assinantes de maneira semelhante à propagação de assinaturas.

- Rendez-vous: primeiro, recebe determinada assinatura, s, e retorna um ou mais nós, os quais assumem a responsabilidade por essa assinatura. Cada nó de rendez-vous mantém uma lista de assinaturas e encaminha todos os eventos correspondentes para o conjunto de nós assinantes. Segundo, quando um evento é publicado, retorna um ou mais nós, desta vez responsável por corresponder e às assinaturas do sistema.

# arquitetura do sistema









## arquitetura da implementação

Implementação é baseada em **tópico** e **centralizada**. Seus nós são publicadores e assinantes ao mesmo tempo. Uso do **banco de dados**:

- **ps.messages**(id intpkey, corpo varchar, idpub intfkey, idsub intfkey, idtopicfkey, createdat timestamp)
- **ps.nodes**(id intpkey, ip varchar, porta varchar)
- **ps.topics**(id intpkey, name varchar)
- **ps.topicsnodes**(idtopic intpkey fkey, idsub intpkey fkey)
- **trigger** para remoção de mensagens antigas

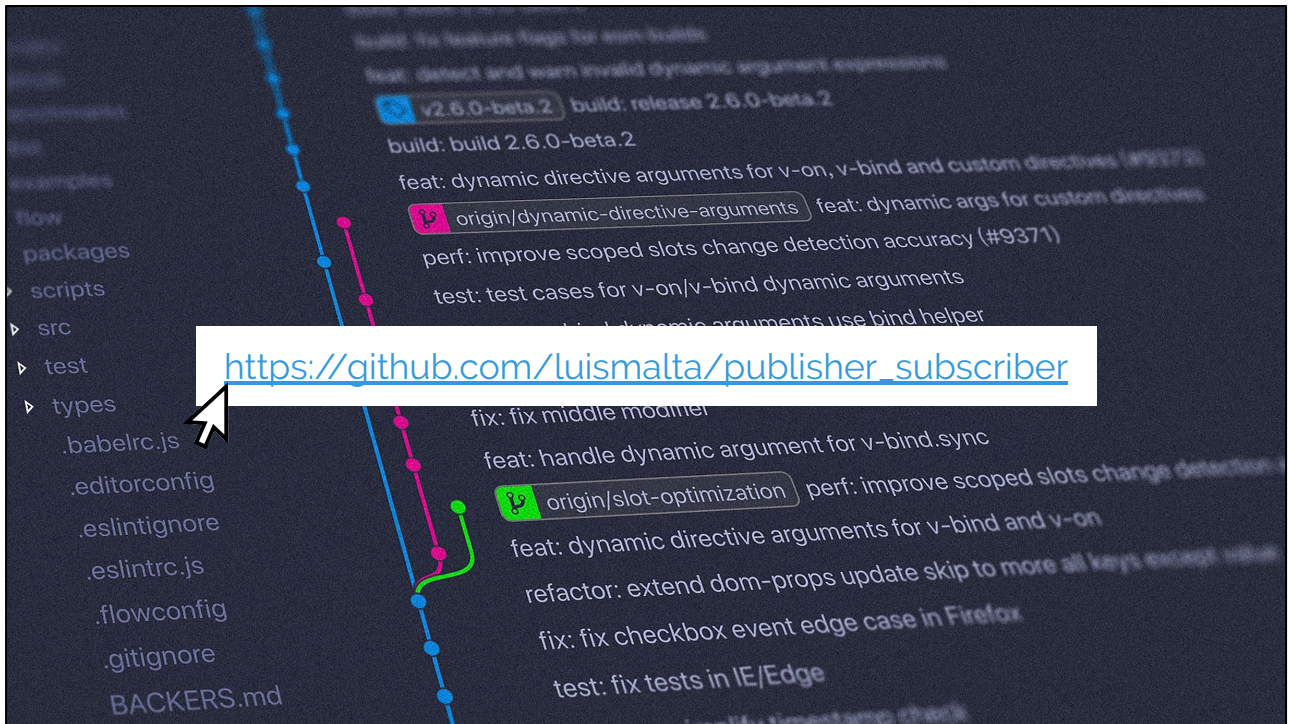
A implementação realizada foi a baseada em tópico. Os nós dessa arquitetura são ao mesmo tempo publicadores e assinantes e se conectam a um único broker centralizado. Uma das questões relevantes é como garantir que uma mensagem iria chegar a um nó assinante mesmo que sua conexão fosse interrompida. Para resolvê-la, escolheu-se o uso do banco de dados relacional postgresSQL. O banco possui um trigger que remove após uma nova inserção na tabela ps.messages todas as mensagens com 30 dias ou mais de criação.

## funcionalidades do sistema

-  Publicar mensagem;
-  Criar tópicos;
-  Listar tópicos disponíveis;
-  Listar tópicos inscritos pelo nó;
-  Assinar um tópico;
-  Cancelar uma assinatura.



- Publicar mensagem: O nó pode escolher um tópico e publicar uma mensagem de texto nele.
- Criar tópicos: Ao publicar uma mensagem em um tópico não existente o broker irá criá-lo.
- Listar tópicos disponíveis: O nó pode solicitar a lista de tópicos disponíveis.
- Listar tópicos escritos pelo nó: O nó pode solicitar a lista de tópicos que ele assinou
- Assinar um tópico: O nó pode de assinar em um tópico existente
- Cancelar uma assinatura: O nó pode cancelar sua assinatura em um tópico.



[https://github.com/luismalta/publisher\\_subscriber](https://github.com/luismalta/publisher_subscriber)