Engenharia de Softwares Escaláveis

Domain-Driven Design (DDD) e Arquitetura de Softwares Escaláveis com Java

Agenda

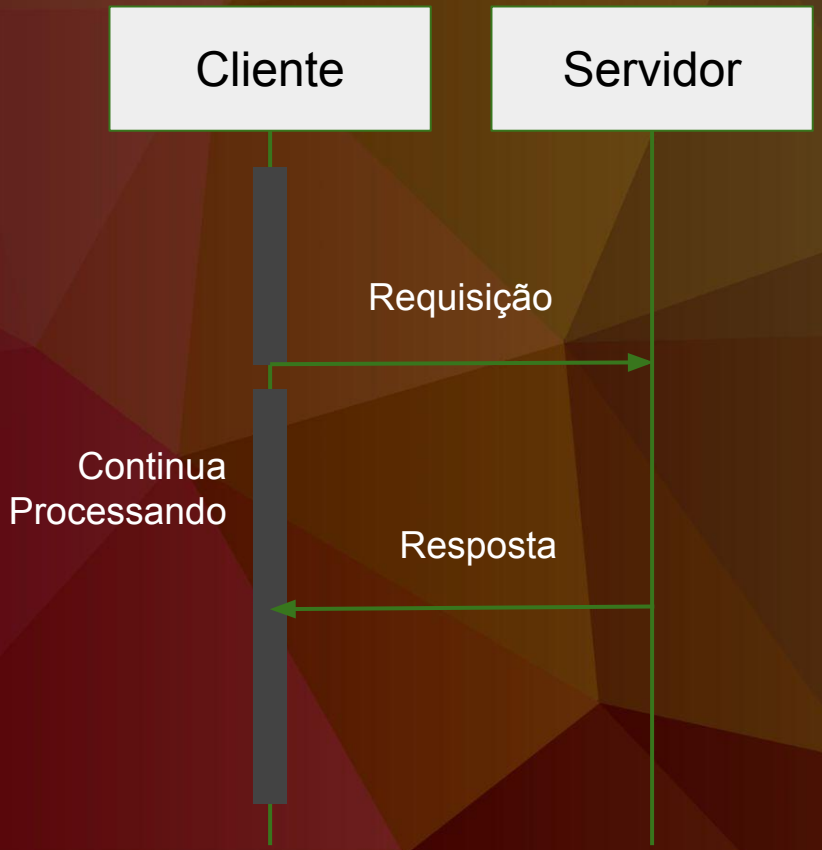
Etapa 3: Implementação e Gerenciamento de Domain Events.

- Comunicação de Microsserviços.
- Eventos de Domínio - Parte 2.

Comunicação de Microsserviços

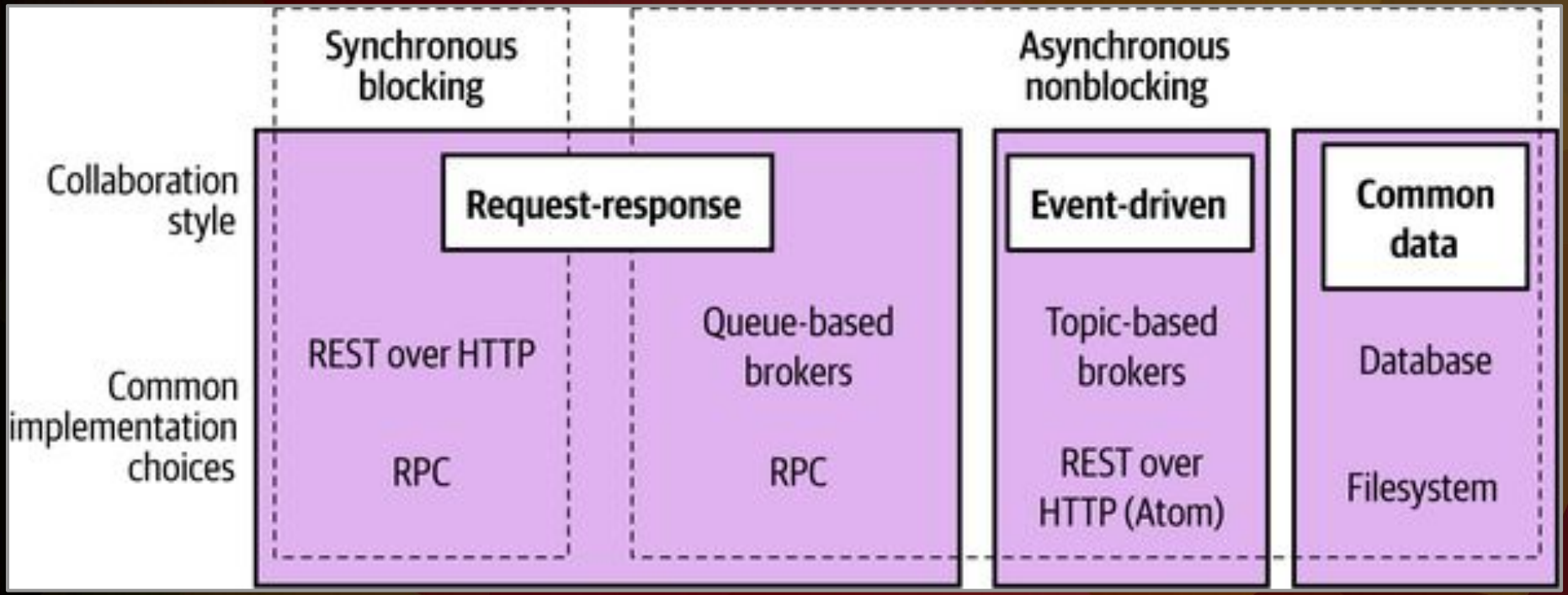
Síncrono

Assíncrono

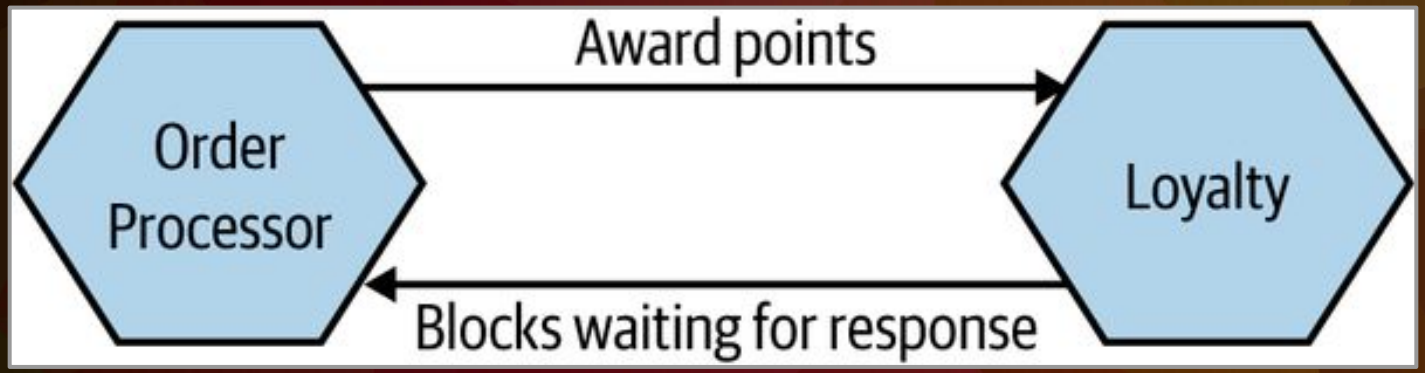


Comunicação	Descrição	Vantagens	Desvantagens
Síncrona	Um microsserviço envia uma solicitação e **bloqueia** a operação aguardando a resposta para saber o resultado. **Exemplos:** APIs REST (HTTP), gRPC.	**Simplicidade:** O fluxo é fácil de entender, similar a uma chamada de função local. **Feedback Imediato:** O chamador sabe imediatamente se a operação teve sucesso ou falhou. **Consistência:** É mais fácil garantir a consistência dos dados em uma única transação de negócios.	**Acoplamento Temporal:** O serviço chamado *precisa* estar online e respondendo. Se ele falhar, o chamador também falha. **Baixa Resiliência:** Pode levar a falhas em cascata. **Desempenho:** O chamador fica ocioso (bloqueado) enquanto espera, reduzindo o *throughput* (vazão) geral do sistema.
Assíncrona	Microsserviços emitem eventos. O emissor **não bloqueia** e continua o processamento. Outros serviços consomem e reagem aos eventos sem que o emissor saiba quem são. **Exemplos:** Filas e Tópicos (RabbitMQ, SQS) ou Logs (Kafka).	**Desacoplamento:** O produtor e o consumidor não precisam se conhecer nem estar online ao mesmo tempo. **Alta Resiliência:** Se um consumidor falhar, o produtor não é afetado. O evento pode ser reprocessado. **Escalabilidade:** Vários consumidores podem processar eventos em paralelo. **Não Bloqueante:** O emissor tem alta performance, pois apenas dispara o evento e segue em frente.	**Complexidade:** Requer infraestrutura adicional (um *message broker*). **Consistência Eventual:** Os dados não são atualizados instantaneamente. Leva um tempo para o evento ser processado. **Depuração Difícil:** Rastrear o fluxo de uma solicitação por múltiplos serviços assíncronos é complexo (exige *distributed tracing*).

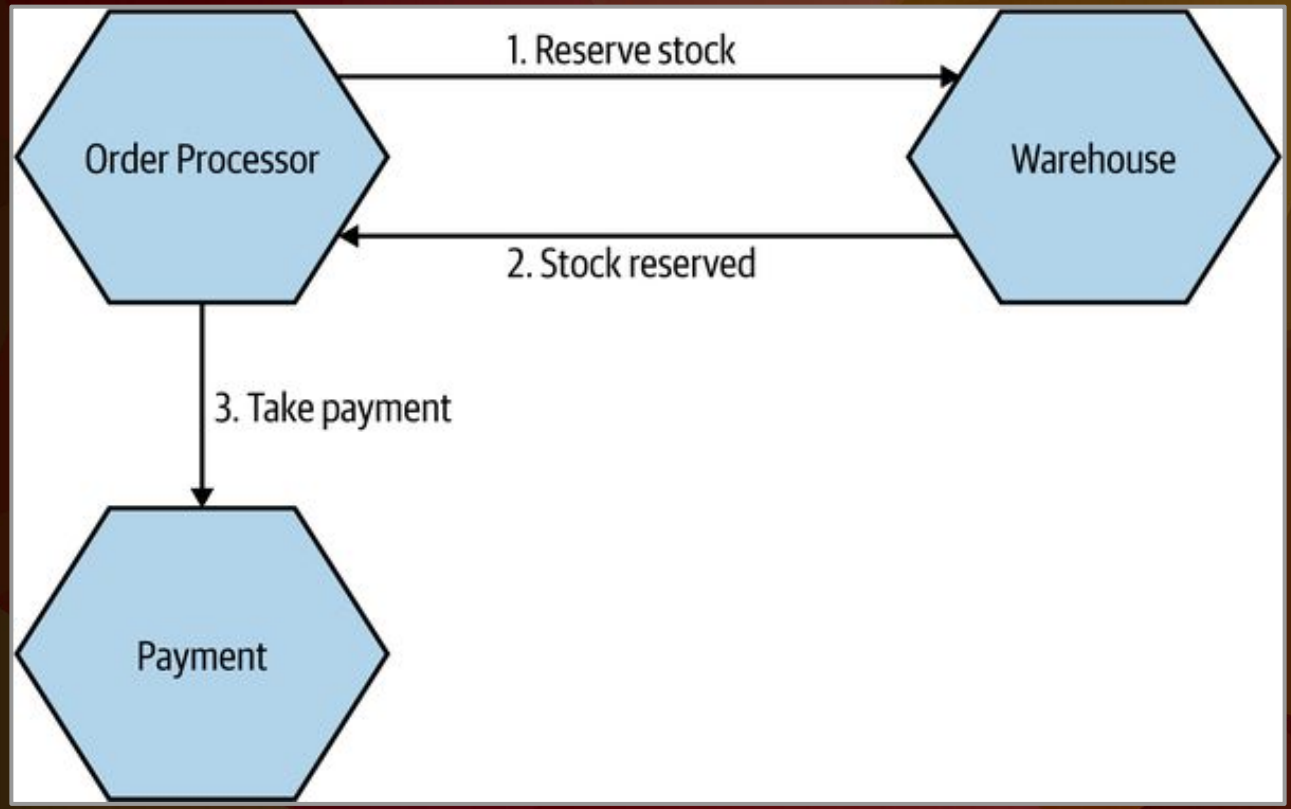
Comunicação de Microsserviços



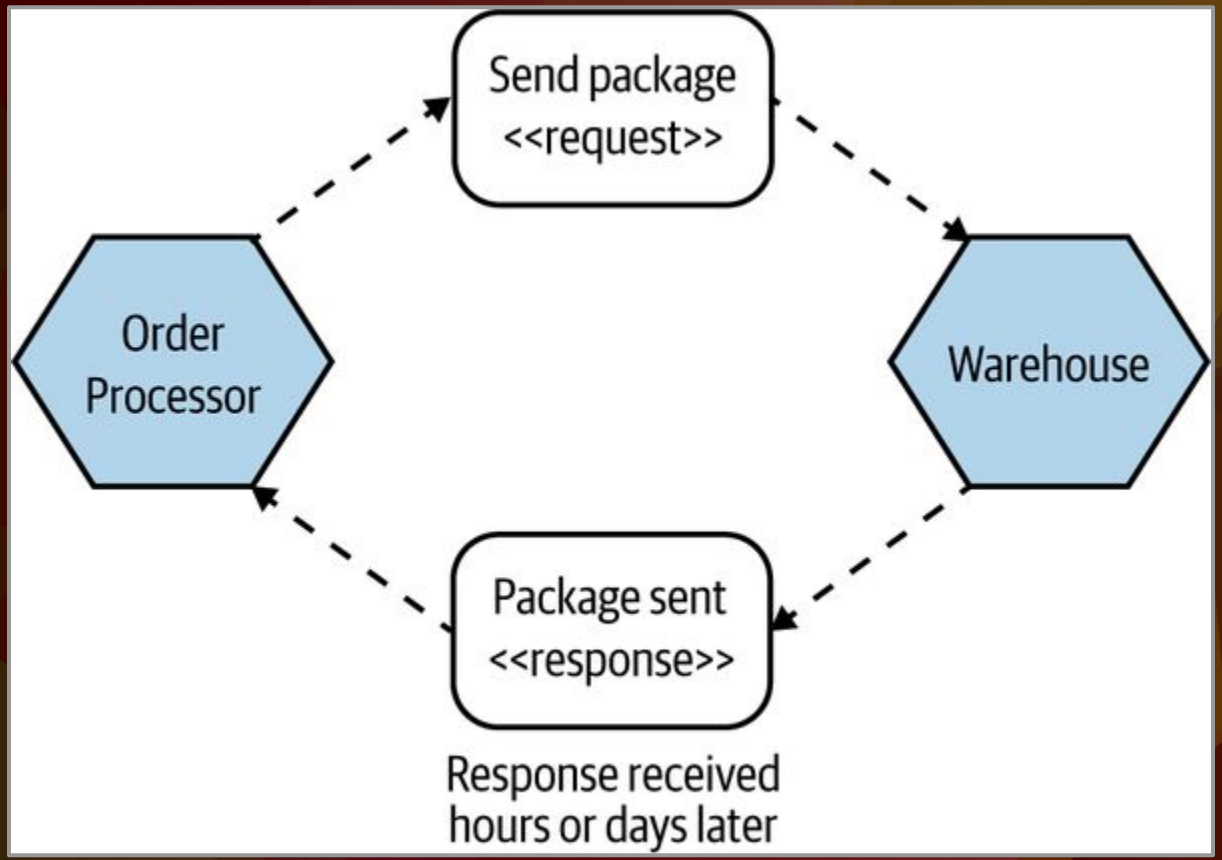
Síncrono Bloqueante



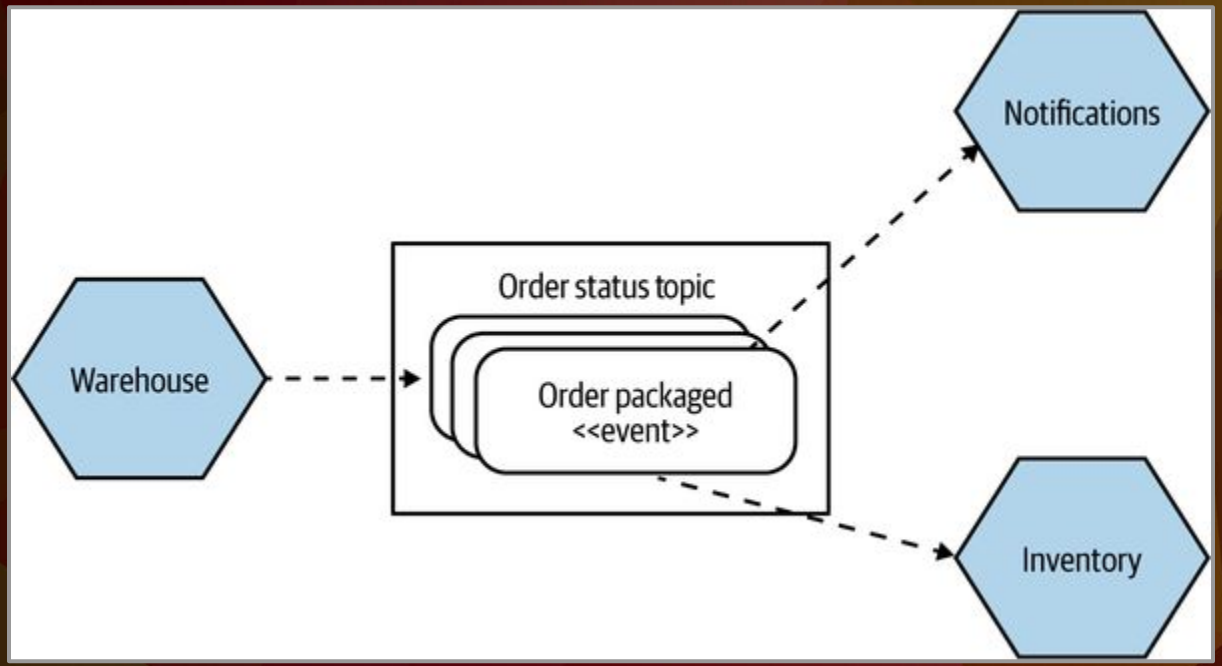
Solicitação-Resposta



Assíncrono Não Bloqueante



Orientado por Eventos



Eventos de Domínio - Parte 2

Implementação de Eventos de Domínio

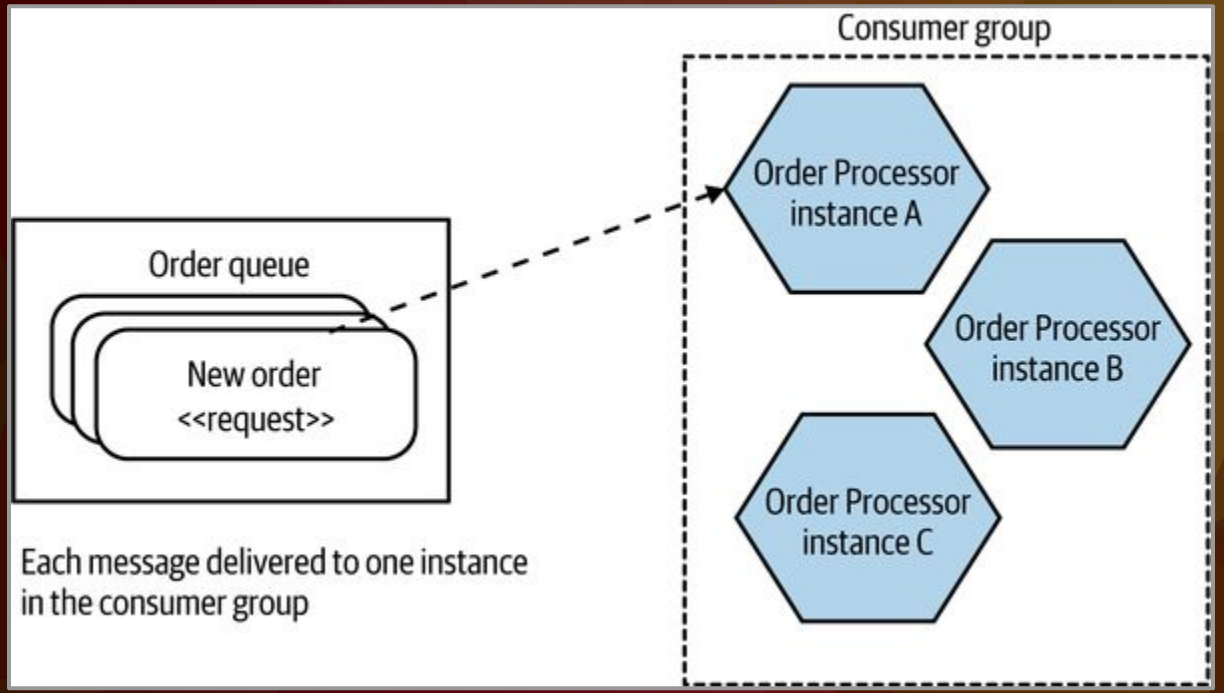
Message Brokers são intermediários, frequentemente chamados de middleware, que ficam entre processos para gerenciar a comunicação entre eles.

Uma mensagem é um conceito genérico que define “a coisa” que um **Message Broker** envia.

Uma mensagem pode conter uma solicitação, uma resposta ou um evento.

Em vez de um microsserviço se comunicar diretamente com outro microsserviço, o microsserviço envia uma mensagem a um **Message Broker**, com informações sobre como a mensagem deve ser repassada adiante.

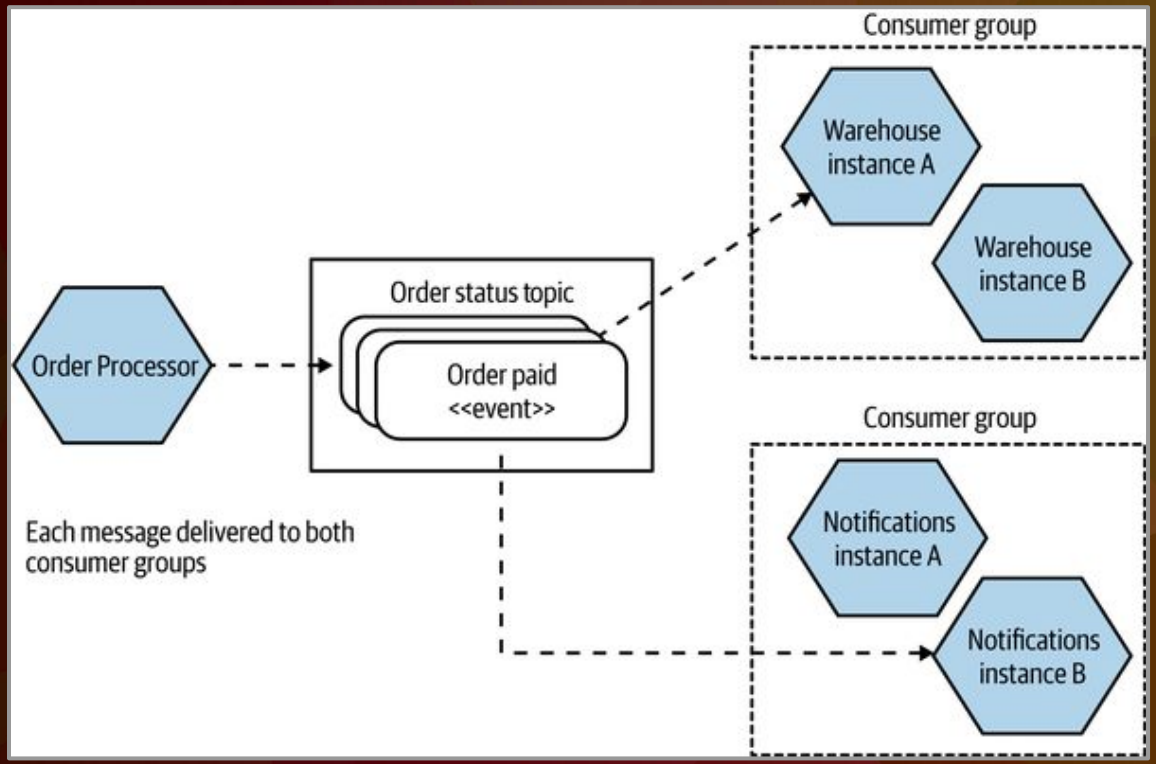
Filas e Tópicos



As **Filas** são tipicamente ponto a ponto.

Um remetente coloca uma mensagem em uma fila, e um consumidor lê a partir dessa fila.

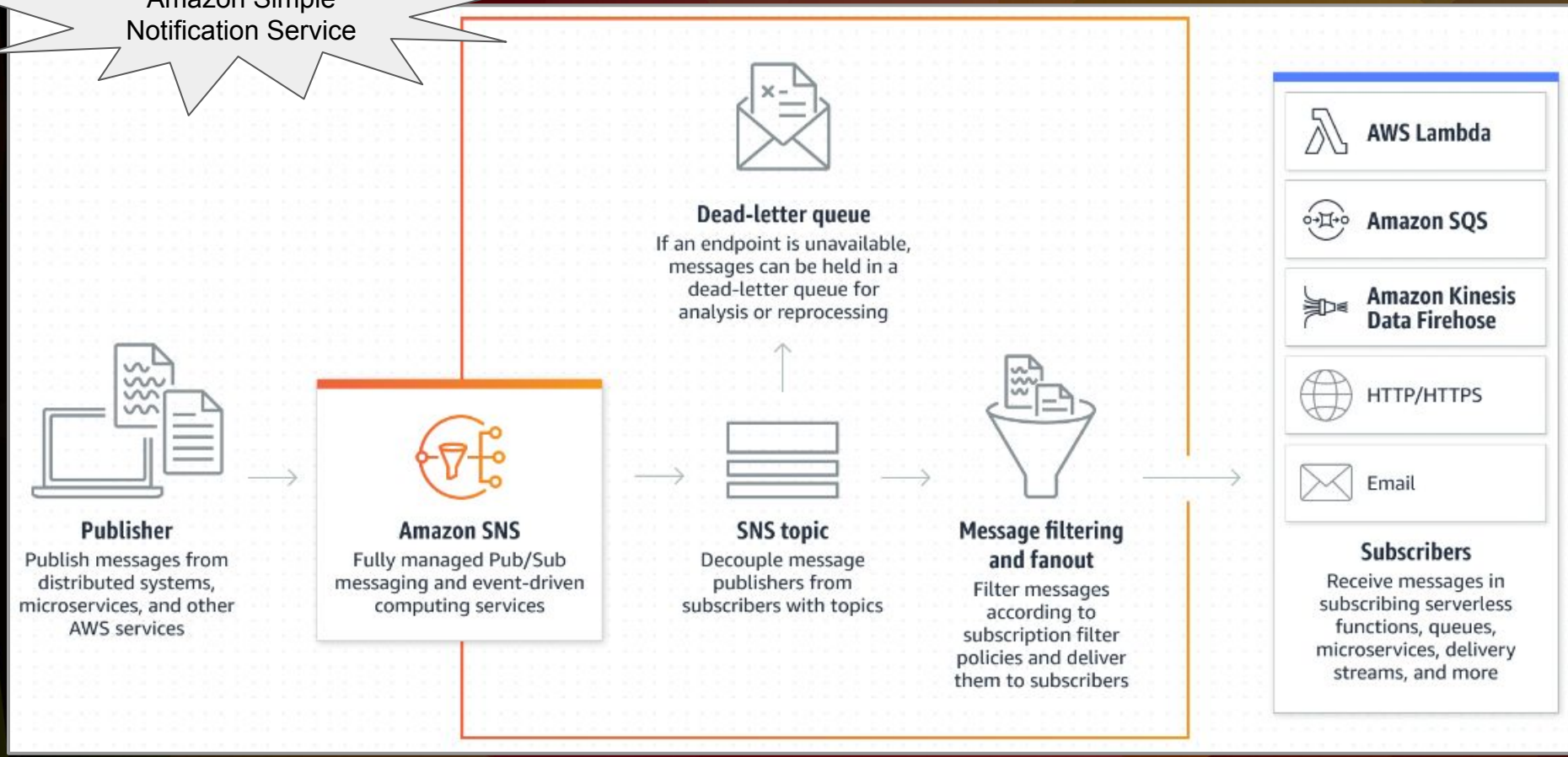
Quando uma mensagem é colocada na fila, apenas um membro do grupo de consumidores receberá essa mensagem.

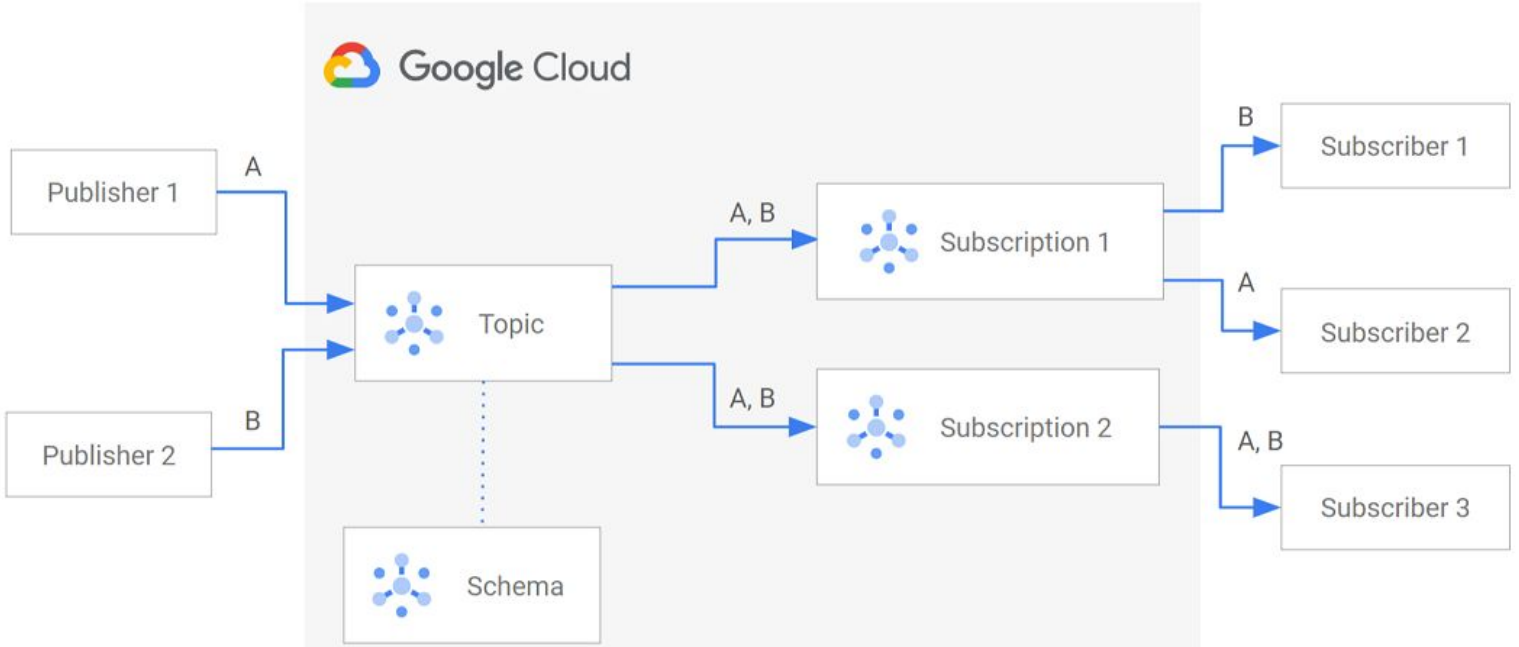


Com **Tópicos**, você pode ter vários grupos de consumidores.

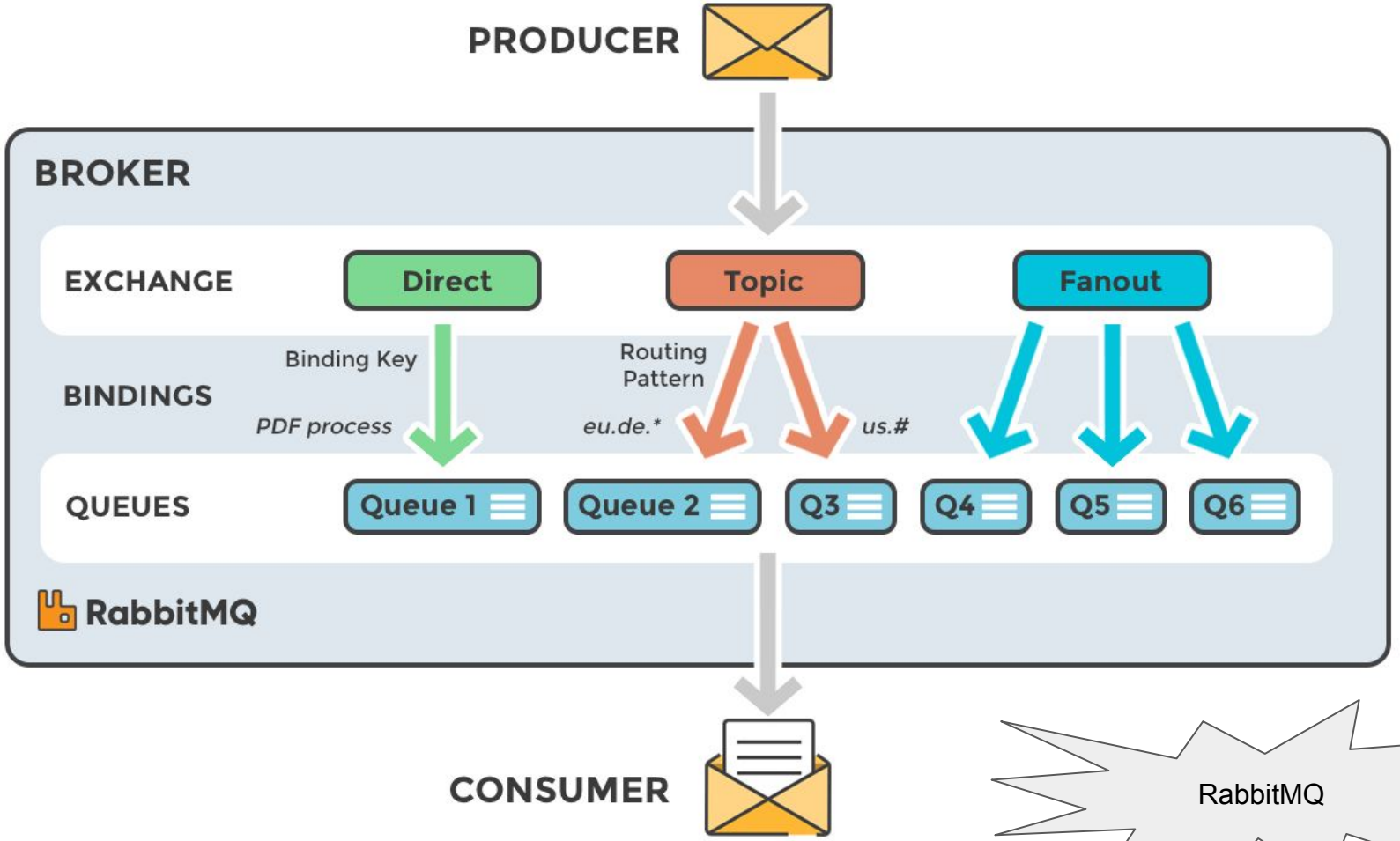
Uma cópia da mensagem é recebida por consumidores diferentes que estão em grupos de separados. Apenas uma instância de cada grupo de consumidores verá esse evento.

Amazon Simple Notification Service

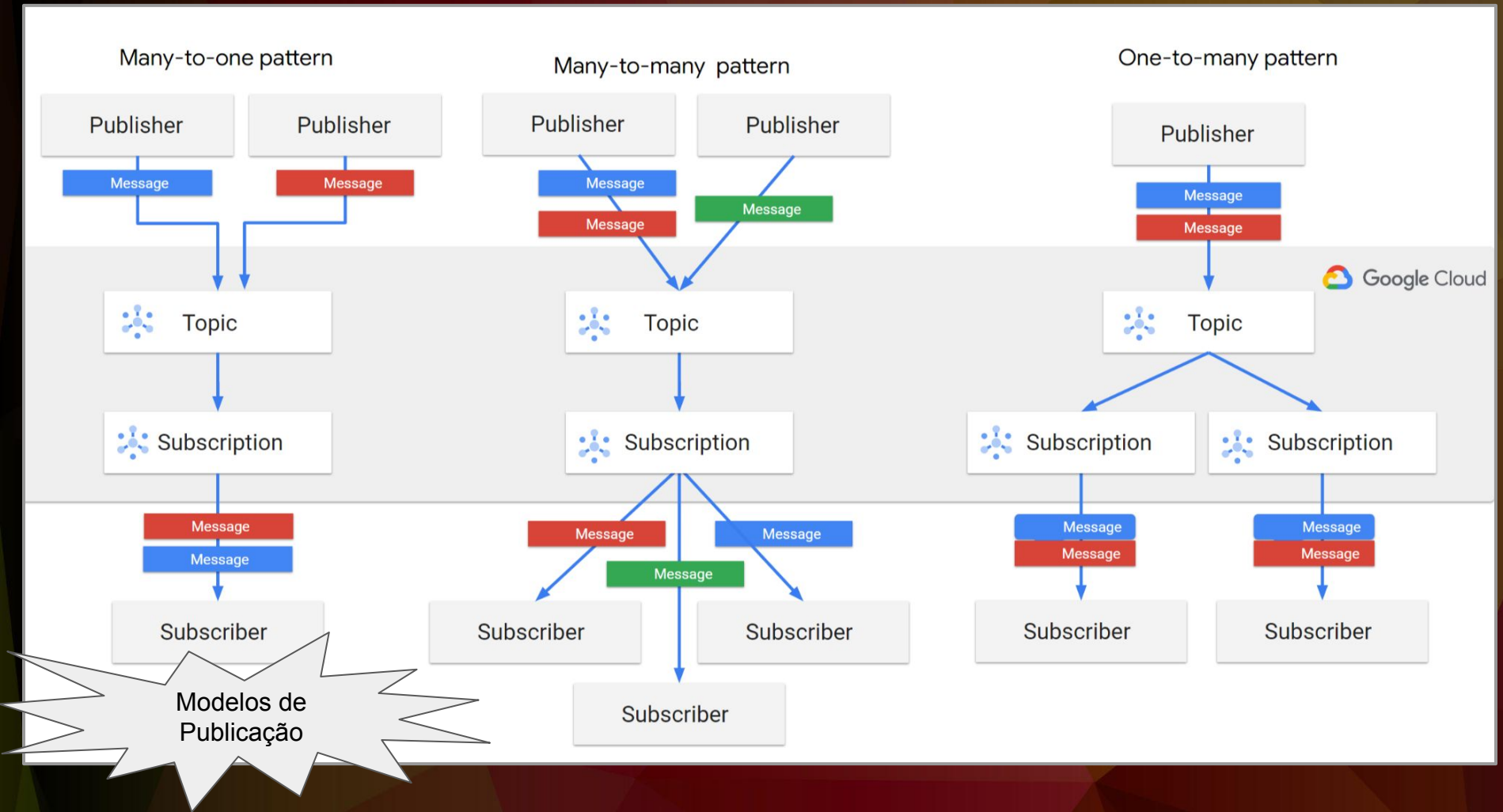


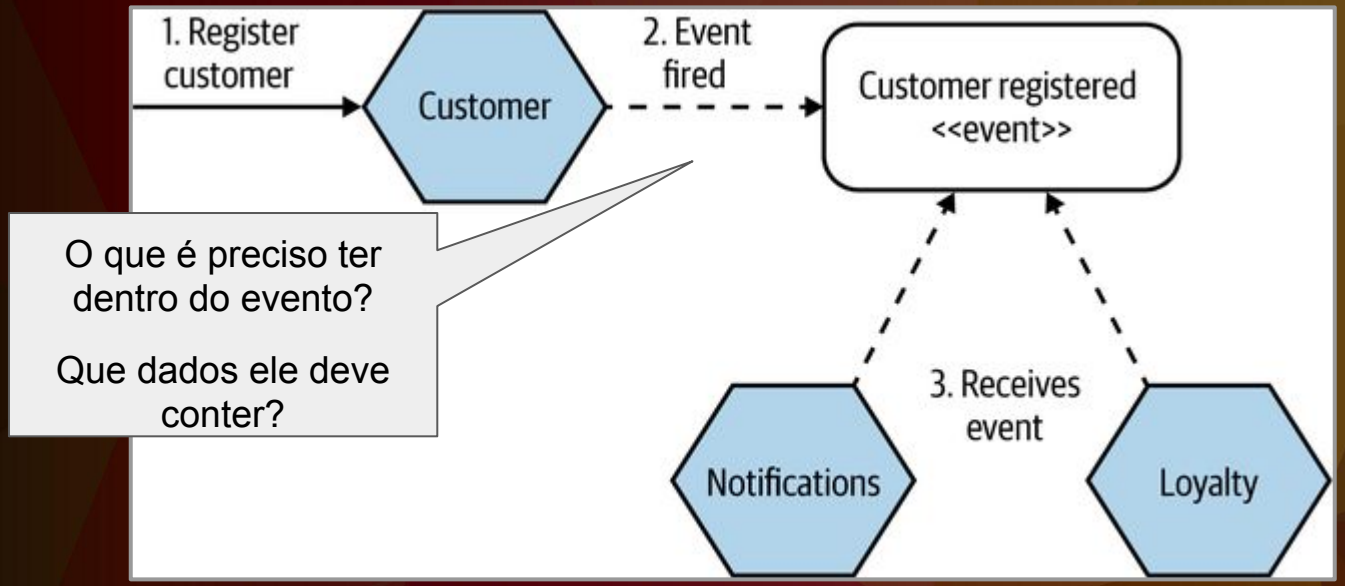


Google Pub/Sub


PRODUCER
BROKER
EXCHANGE
Direct
Topic
Fanout
Binding Key
Routing Pattern
BINDINGS
PDF process
eu.de.*
us.#
QUEUES
Queue 1
Queue 2
Q3
Q4
Q5
Q6
RabbitMQ
CONSUMER
RabbitMQ

Característica	Fila (Queue)	Tópico (Topic)	Fanout (Exchange Type)
Padrão	**Ponto-a-Ponto**	**Publicar/Assinar**	**Publicar/Assinar**
Lógica de Entrega	A mensagem é entregue a apenas um consumidor, mesmo que vários estejam escutando a fila.	A mensagem é entregue a todos os assinantes que estão registrados no tópico.	A mensagem é enviada para todas as filas que estão ligadas a ele.
Relação	Um-para-Um	Um-para-Muitos	Um-para-Muitos
Conceito Principal	A fila "guarda" a mensagem até que um consumidor a "puxe".	O tópico "publica" uma cópia da mensagem para cada assinante independente.	É um roteador. Ele não armazena mensagens; apenas as "copia" e as encaminha para todas as filas ligadas a ele.
Filtragem de Mensagens	Não aplicável. O consumidor escuta a fila inteira.	Sim. Permite filtragem sofisticada. Assinantes podem usar padrões para receber apenas o que interessa.	Não. O Fanout ignora qualquer filtro. Sua única regra é: "enviar para todos".
Principal Caso de Uso	Processar um pedido, enviar um e-mail. A tarefa só deve ser feita uma vez.	Notificações e Eventos: Um evento (ex: "Pedido Criado") precisa ser consumido por múltiplos microsserviços (Estoque, Faturamento, Notificação).	Broadcast Global: Enviar uma mensagem que todos os serviços precisam receber, como "limpar cache" ou "atualização de configuração".

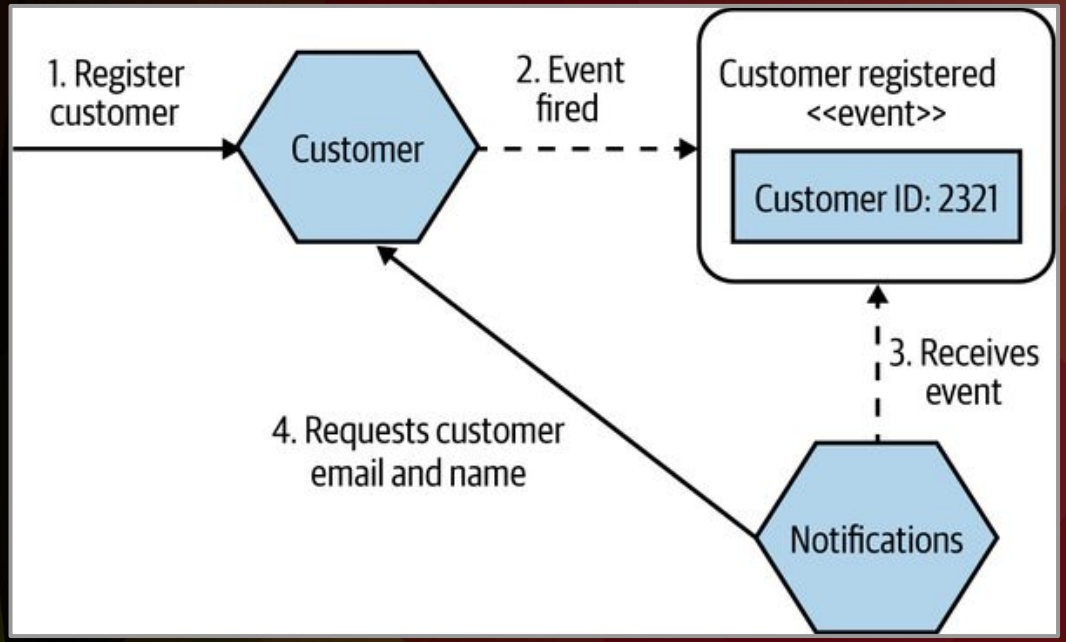

Many-to-one pattern
Publisher
Publisher
Message
Message
Topic
Subscription
Message
Message
Subscriber
Many-to-many pattern
Publisher
Publisher
Message
Message
Message
Topic
Subscription
Message
Message
Message
Subscriber
Subscriber
Subscriber
One-to-many pattern
Publisher
Message
Message
Topic
Subscription
Subscription
Message
Message
Message
Message
Subscriber
Subscriber
Google Cloud
Modelos de Publicação



Um evento sendo transmitido de Customer, informando as partes interessadas que um novo cliente se registrou no sistema.

Dois dos microsserviços downstream, Loyalty e Notifications, se importam com esse evento.

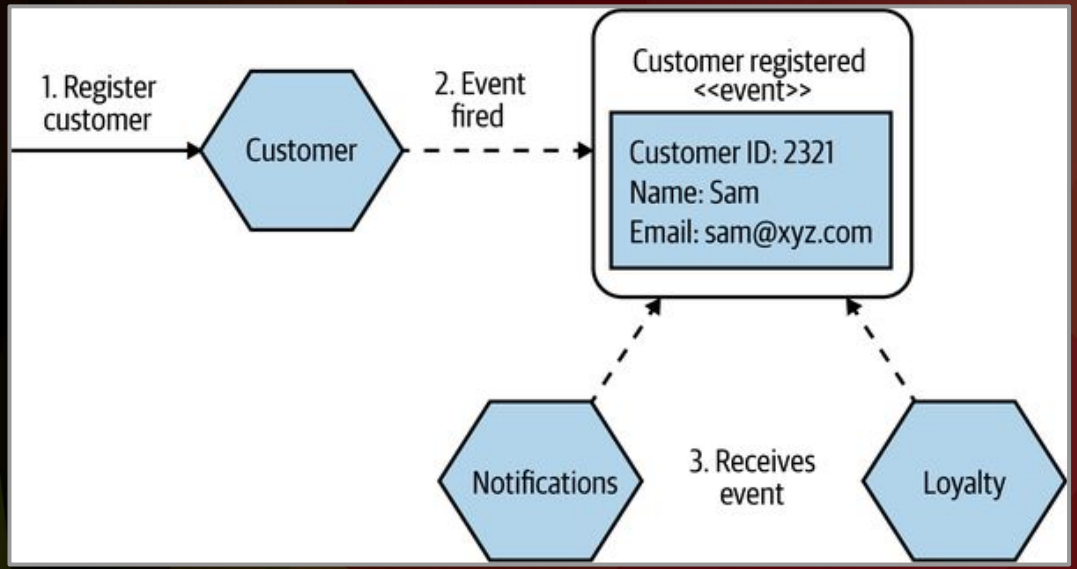
O Loyalty reage ao receber o evento configurando uma conta para o novo cliente para que ele possa começar a ganhar pontos, enquanto o Notifications envia um e-mail ao cliente recém-registrado dando-lhe as boas-vindas.



Uma opção é que o evento contenha apenas um identificador para o cliente recém-registrado.

O Loyalty precisa apenas desse identificador para criar a conta de fidelidade correspondente, então ele tem todas as informações de que precisa.

No entanto, embora o Notifications saiba que precisa enviar um e-mail de boas-vindas quando esse tipo de evento for recebido, ele precisará de informações adicionais para fazer seu trabalho.



A alternativa é colocar todos os dados necessários em um evento.

Se você deixasse Notifications pedir o endereço de e-mail e o nome de um determinado cliente, por que não colocar essas informações no evento em primeiro lugar?

Notifications agora é autossuficiente e capaz de fazer seu trabalho sem precisar se comunicar com o Customer.

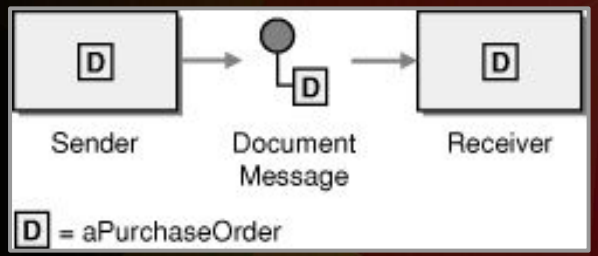
Na verdade, ele pode nunca precisar saber que o Customer existe.

Documento

Uma mensagem genérica que contém apenas dados.

O receptor decide como interpretá-la.

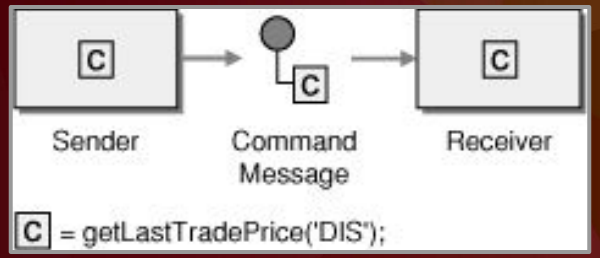
A resposta a um comando é um exemplo de uma mensagem de documento.



Command

Uma mensagem que é o equivalente a uma solicitação RPC.

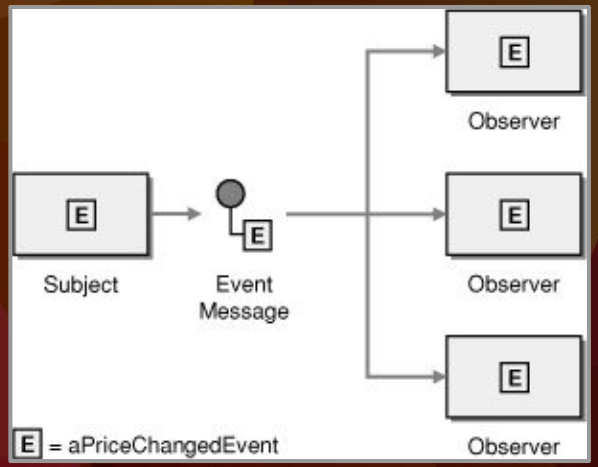
Ela especifica a operação a ser invocada e seus parâmetros.



Evento

Uma mensagem indicando que algo notável ocorreu no remetente.

Um evento é frequentemente um evento de domínio, que representa uma mudança de estado de um objeto de domínio.



Tipos de Mensagens

Característica	Command	Event	Document
Descrição	Equivalente a uma chamada RPC. Especifica uma operação e seus parâmetros.	Indica que uma mudança de estado ocorreu no remetente.	Uma mensagem genérica que contém apenas dados. O receptor decide como interpretá-la.
Propósito Principal	**Dizer** a um serviço para **fazer algo**.	**Informar** a outros serviços que **algo aconteceu**.	**Transferir** um conjunto de dados.
Intenção	Imperativa (uma ordem).	Reativa (um fato que ocorreu).	Informativa (os dados em si).
Foco da Nomenclatura	Verbo no imperativo.	Verbo no passado.	Substantivo (o próprio documento).
Relação	**1-para-1.** O remetente espera que *um* receptor específico execute a ação.	**1-para-Muitos.** O remetente (publicador) não sabe quem (ou quantos) são os receptores (assinantes).	**Variável.** Pode ser 1-para-1 (como resposta a um comando) ou 1-para-Muitos (como um relatório publicado).
Acoplamento	**Alto acoplamento.** O remetente *precisa* saber sobre o receptor e a operação que ele pode executar.	**Baixo acoplamento.** O publicador e os assinantes não precisam se conhecer.	**Acoplamento de dados.** O receptor precisa entender a *estrutura* (schema) dos dados para poder interpretá-los.
Exemplo	CriarPedidoCommand (enviado ao serviço de Pedidos).	PedidoCriadoEvent (publicado pelo serviço de Pedidos).	DetalhesDoPedido (enviado como resposta a um BuscarPedidoCommand).