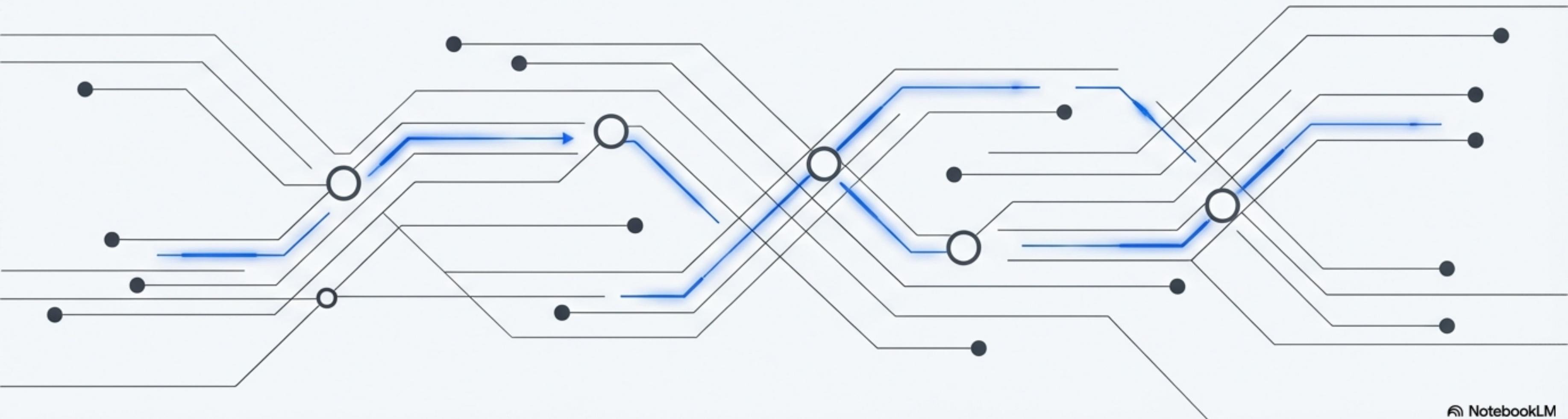




Dominando a Mensageria Assíncrona com SAP CAP

Um Guia Prático para Desenvolvedores Java

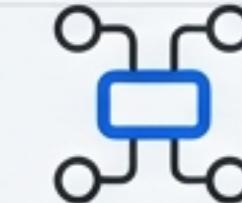
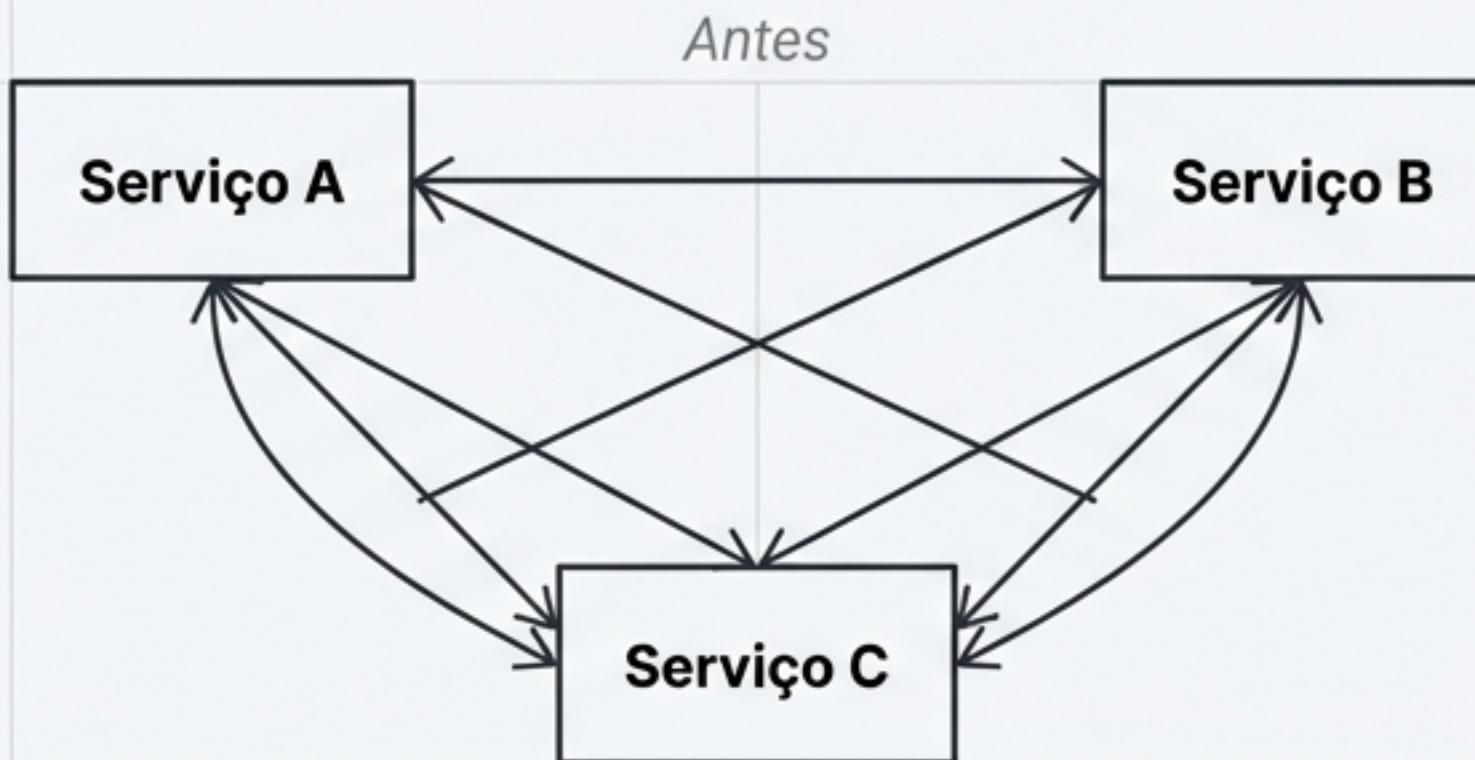


Por Que a Mensageria Assíncrona é Essencial?



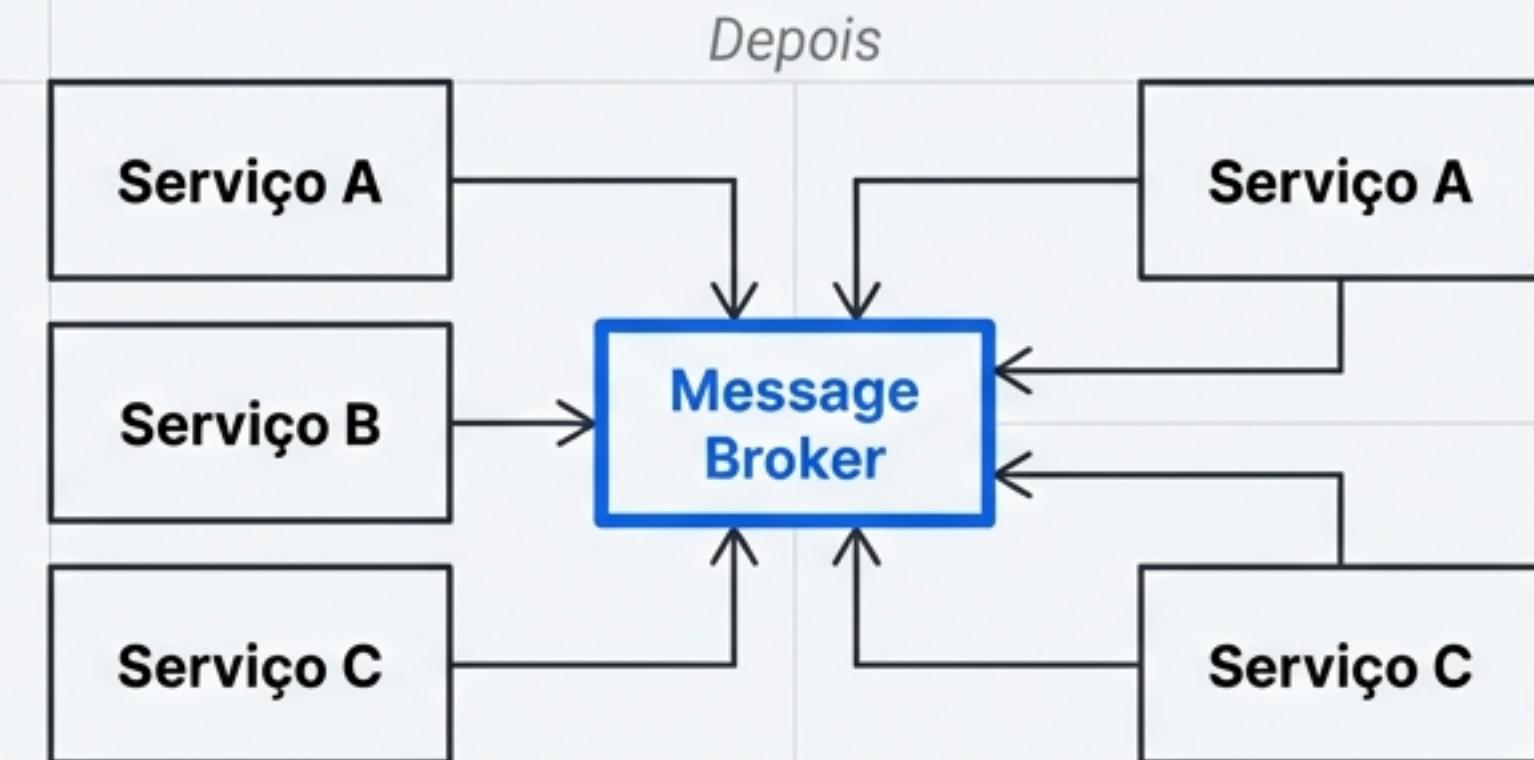
Comunicação Síncrona (O Desafio)

O remetente e o destinatário precisam estar disponíveis simultaneamente. Cria um forte acoplamento, onde o remetente precisa conhecer e chamar o destinatário diretamente. A comunicação é tipicamente ponto a ponto, dificultando a escalabilidade.



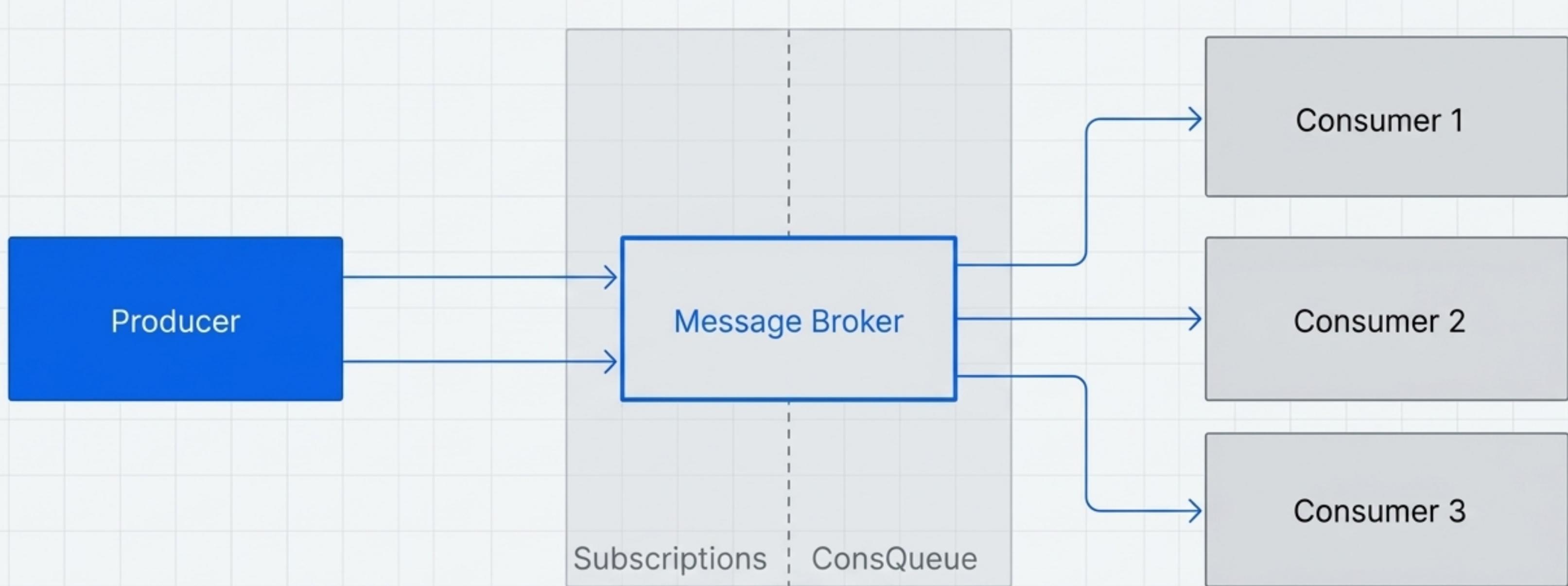
Padrão Publish-Subscribe (A Solução)

Remetentes (publishers) enviam mensagens para um "broker" sem conhecer os destinatários (subscribers). Os destinatários se inscrevem em tópicos de interesse. O broker gerencia a entrega, garantindo que as mensagens não se percam mesmo se o destinatário estiver offline.



A Abstração do SAP CAP: Foco no Negócio, Não na Infraestrutura

O CAP Messaging fornece uma API agnóstica de broker, cuidando de toda a mecânica de baixo nível: gerenciamento de conexões, protocolos, criação de filas e inscrições. A API se integra perfeitamente ao modelo de eventos do CAP, permitindo enviar mensagens com `emit` e receber com a anotação `@On`.



Enviando a Primeira Mensagem: A API emit

Passo 1: Habilitando o Broker Local

Para desenvolvimento e testes locais, o file-based-messaging simula um broker real, persistindo mensagens em um arquivo. Nenhuma instalação de broker é necessária.

```
cds:  
  messaging:  
    services:  
      - name: "messaging-name"  
        kind: "file-based-messaging"  
        binding: "/any/path/to/file.txt"
```

Passo 2: Utilizando emit

Você pode enviar mensagens diretamente pela API técnica ou de forma mais estruturada, usando um EventContext.

```
@Autowired  
MessagingService messagingService;  
  
// 1. Via API técnica: rápido e direto  
messagingService.emit("My/Topic", Map.of("message", "  
  
// 2. Via EventContext: padrão do CAP, mais flexível  
TopicMessageEventContext context = TopicMessageEventContext.  
context.setDataMap(Map.of("message", "hello world"));  
messagingService.emit(context);
```



Dica: Mensagens são enviadas apenas quando a transação é concluída com sucesso, usando um outbox em memória por padrão.

Recebendo Mensagens: O Poder do @On



Dica

Handlers @On para mensageria são completados automaticamente pelo CAP para permitir processamento paralelo por múltiplos componentes. Não chame `context.setComplete d()` manualmente.

Para receber mensagens, basta declarar um handler com a anotação **@On**, especificando o nome do serviço e o tópico (evento).

```
@On(service = "messaging-name", event = "My/Topic")
public void receiveMyTopic(TopicMessageEventContext context) {
    // Acessa o ID e o payload da mensagem
    String msgId = context.getMessageId();
    Map<String, Object> payload = context.getDataMap();

    // ... sua lógica de negócio aqui
}
```



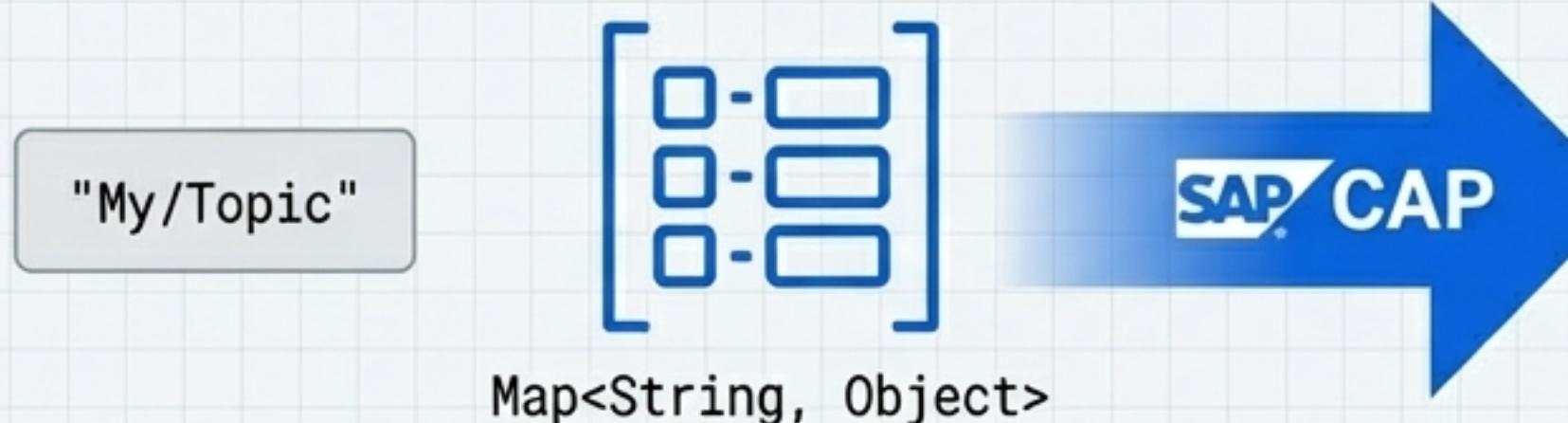
Aviso

Qualquer exceção não tratada no seu handler fará com que a mensagem **não** seja reconhecida ('acknowledged'). O broker tentará reenviá-la, podendo gerar um loop. O tratamento de erros é crucial.

Modelando Eventos com CDS: Da String ao Contrato

Em vez de usar tópicos como strings, declare eventos diretamente no seu modelo de serviço CDS. O CAP irá gerar automaticamente interfaces Java para acessar a mensagem e seu payload de forma type-safe, criando um contrato claro entre serviços.

Método Antigo



Método CDS

```
service ReviewService {  
    // ...  
    event reviewed : {  
        subject: String;  
        rating: Decimal(2,1);  
    }  
    // ...  
}
```

Benefícios

- ✓ **Type Safety:** Reduz erros em tempo de execução.
- ✓ **Clareza:** O modelo de dados define a API de eventos.
- ✓ **Desacoplamento:** Serviços publicam eventos de negócio, não mensagens técnicas.

Publicando Eventos de Domínio com Type Safety

O ReviewService agora emite um contexto de evento fortemente tipado. Note que o serviço em si (`reviewService.emit`) dispara o evento; ele não interage diretamente com um MessagingService técnico.

```
// ...dentro de um handler do ReviewService
Reviewed event = Reviewed.create();
event.setSubject(review.getSubject());
event.setRating(avg);
```

```
ReviewedContext evContext = ReviewedContext.create();
evContext.setData(event);
```

```
// Emite o evento de negócio
reviewService.emit(evContext);
```

Customizando o Tópico

Por padrão, o CAP usa o Fully Qualified Name (FQN) do evento como nome do tópico. Para definir um nome explícito e versionado, use a anotação `@topic`.

```
@topic:
'sap.cap.reviews.v1.ReviewService.changed
event reviewed : { ... }
```

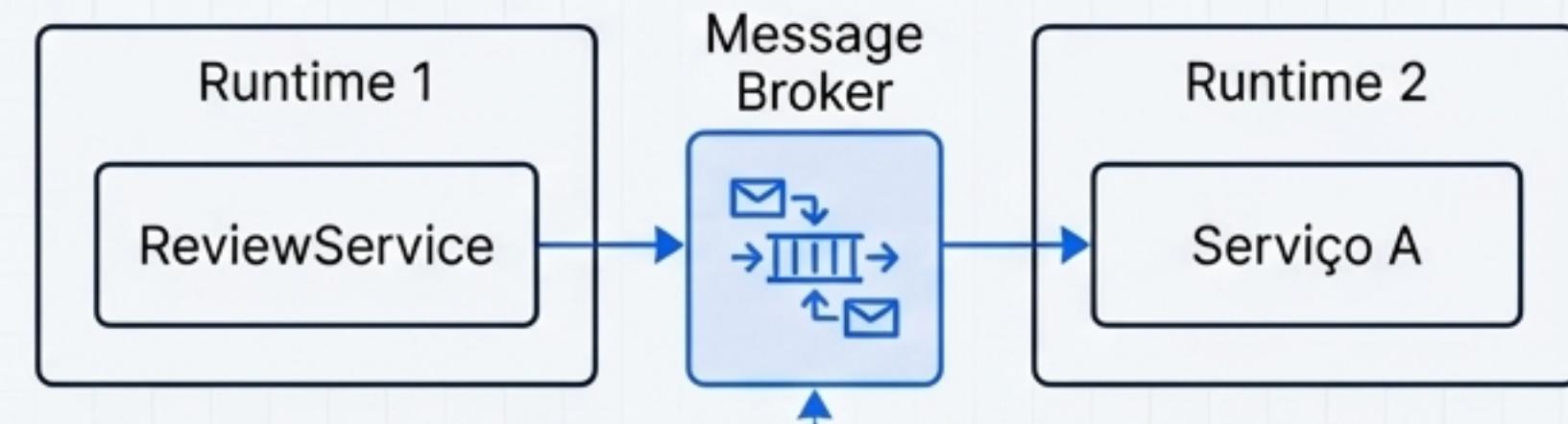
Consumindo Eventos de Domínio: A Magia da Configuração

Cenário 1: Comunicação Local



Se o `ReviewService` está no mesmo runtime, o evento é despachado internamente, sem passar por um broker. É uma chamada de método eficiente.

Cenário 2: Comunicação Remota



Quando o `ReviewService` é configurado como um serviço remoto, o CAP automaticamente cria uma inscrição no broker. A mesma lógica de handler funciona, agora recebendo a mensagem da nuvem.

Código e Configuração, demonstram que code, não same: o configuração triggerá a remote scenario.

Handler do Consumidor

```
@On(service = ReviewService_.CDS_NAME)
private void ratingChanged(ReviewedContext context) {
    Reviewed event = context.getData();
    System.out.println("Rating changed for: " + event.getSubject());
}
```

Configuração Remota

```
cds:
  remote.services:
    - name: ReviewService
messaging.services:
  messaging-em:
    kind: enterprise-messaging
```

Configurando o "Backbone": Do Teste Local à Nuvem



Testes Locais

Simplicidade e Rapidez

`local-messaging`:

Para testes automatizados. O `emit` bloqueia até que todos os receivers processem. Ideal para testes de unidade/integração.

`file-based-messaging`:

Simula um broker real usando um arquivo. Permite a comunicação entre processos na mesma máquina e a injeção manual de mensagens para depuração.



Brokers Reais

Conectando-se ao Mundo Real

O suporte a brokers como SAP Event Mesh, Redis, etc., é adicionado via dependências Maven. Você só inclui o que precisa.

```
<dependency>
    <groupId>com.sap.cds</groupId>
    <artifactId>cds-feature-enterprise-messaging</artifactId>
    <scope>runtime</scope>
</dependency>
```



Integração com Cloud Foundry

Deploy em Produção

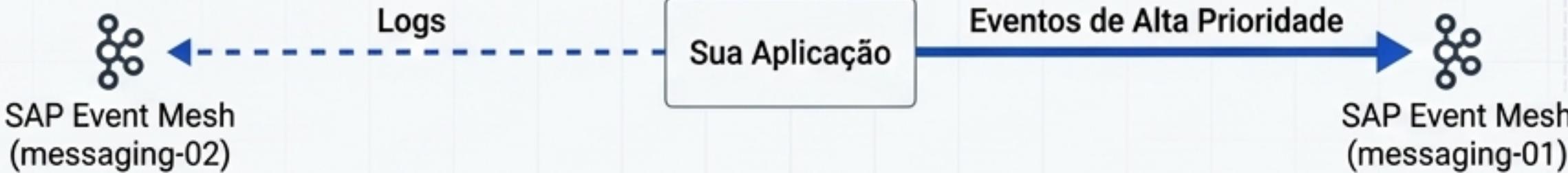
Em um ambiente Cloud Foundry, o CAP detecta automaticamente os serviços de mensageria vinculados ([bound services](#)).

Para desenvolver localmente usando serviços da nuvem, você pode simular as variáveis de ambiente `'VCAP_SERVICES'` e `'VCAP_APPLICATION'` através de um arquivo `'default-env.json'`.



Gerenciando Múltiplas Conexões de Broker com Precisão

O Cenário



Sua aplicação precisa se conectar a duas instâncias diferentes do SAP Event Mesh: `messaging-01` (para eventos de alta prioridade) e `messaging-02` (para logs).

Configuração (YAML)

Opção 1: Detecção Automática

O CAP cria beans `MessagingService` com os mesmos nomes das instâncias vinculadas no Cloud Foundry.

```
cds:  
  messaging:  
    services:  
      - name: "messaging-01"  
      - name: "messaging-02"
```

Opção 2: Nomes Lógicos (Abstração)

Use a propriedade `binding` para mapear nomes de serviço lógicos no seu código para nomes de instâncias físicas na plataforma. Isso mantém seu código estável mesmo que os nomes na nuvem mudem.

```
cds:  
  messaging:  
    services:  
      - name: "priority-events"  
        binding: "messaging-01"  
      - name: "log-events"  
        binding: "messaging-02"
```

Implementação (Java)

Injetando o Serviço Correto

Quando múltiplos serviços estão presentes, use a anotação `@Qualifier` para injetar a instância exata que você precisa, evitando ambiguidades.

```
@Autowired  
 @Qualifier("log-events")  
 MessagingService loggingMessagingService;
```

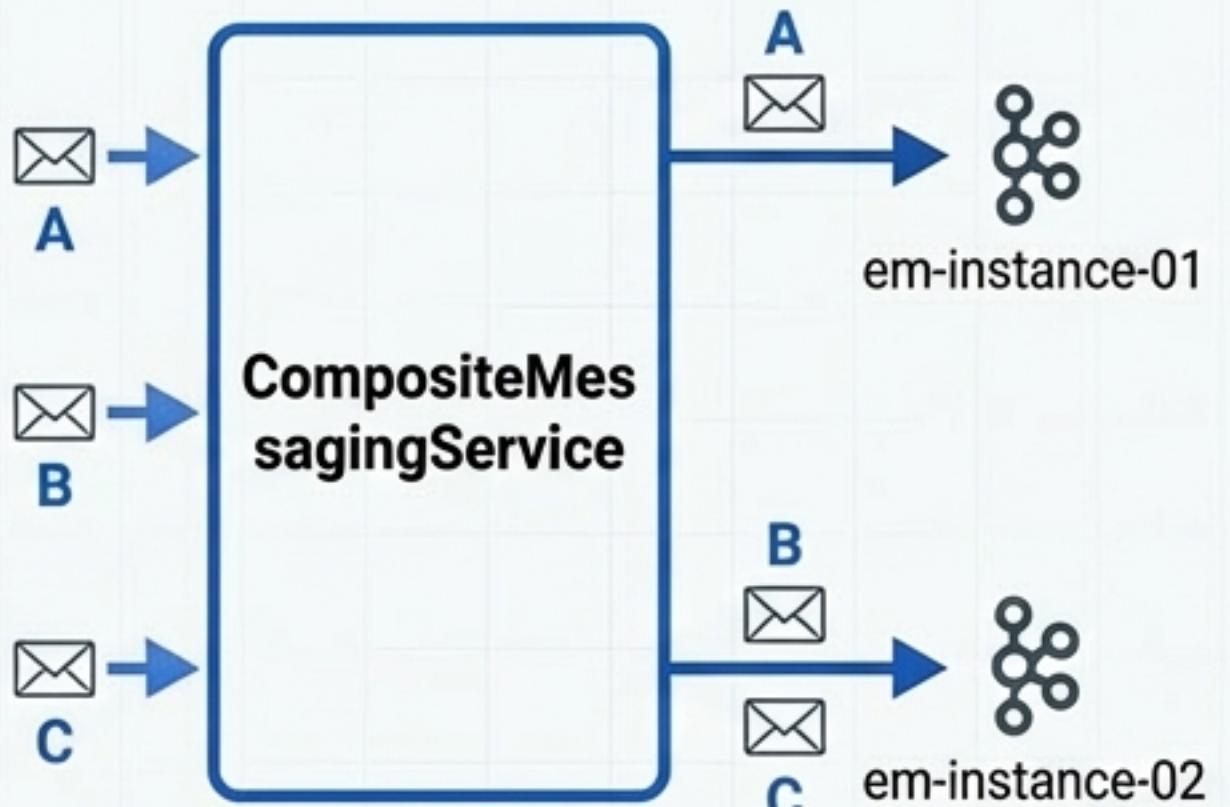
Roteamento Avançado: O Composite Messaging Service

A Solução: Configure o Roteamento

Use o `CompositeMessagingService`. Ele atua como uma fachada única.

Você define as regras de roteamento (qual tópico vai para qual broker) externamente no `application.yaml`.

```
cds:  
  messaging:  
    routes:  
      - service: "em-instance-01"  
        events:  
          - "My/Topic/A"  
      - service: "em-instance-02"  
        events:  
          - "My/Topic/B"  
          - "My/Topic/C*"
```



Código Limpo e Unificado

No código, injete e use apenas o serviço composto. Ele fará o roteamento automaticamente com base na configuração.

```
// Enviando  
@Autowired  
@Qualifier("MessagingService.COMPOSITE_NAME")  
MessagingService messagingService;  
messagingService.emit("My/Topic/A", ...); // → em-instance-01  
messagingService.emit("My/Topic/A", ...);  
messagingService.emit("My/Topic/B", ...); // → em-instance-02  
  
// Recebendo  
@On(service = "MessagingService.COMPOSITE_NAME", event = "My/Topic/A")  
public void receiveA(...) { ... }
```

Controle Fino de Filas e Conexões

Configuração de Filas

Nomeação Explícita

Defina um nome de fila explícito em vez de um autogerado.

```
queue:  
  name: "my-custom-queue"
```

Parâmetros do Broker

Passe configurações customizadas para a fila no momento da criação.

```
queue:  
  config:  
    accessType: "EXCLUSIVE" # Exemplo para SAP Event Mesh
```



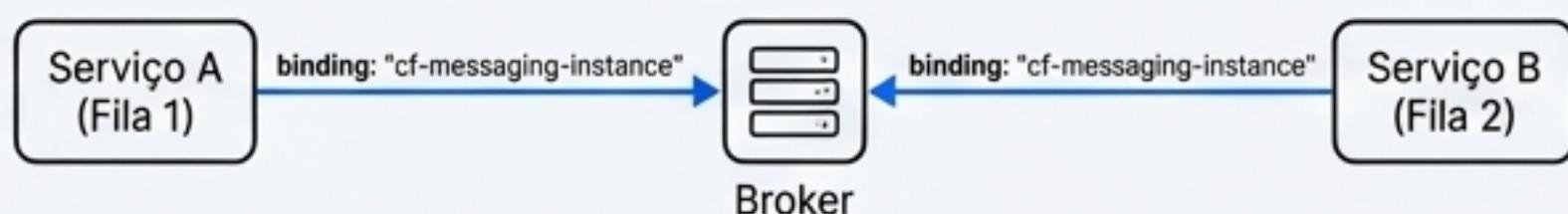
Aviso Box

Filas nunca são deletadas automaticamente pelo CAP. A limpeza de filas não utilizadas deve ser feita manualmente via console ou API do broker.

Arquiteturas Avançadas

Múltiplas Filas, Um Broker

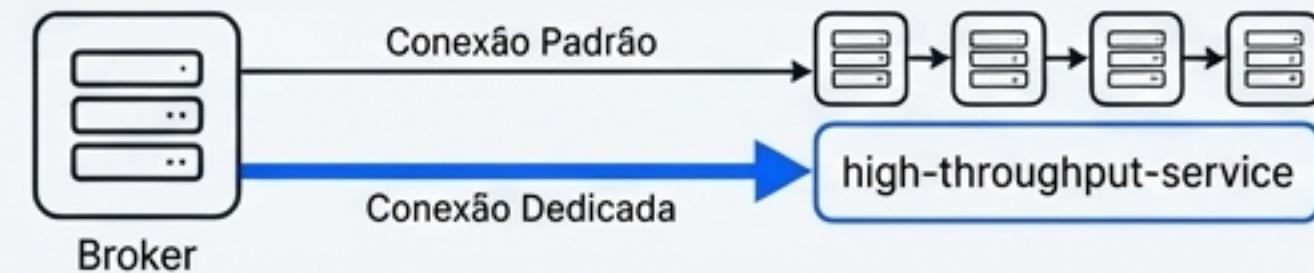
Para separar fluxos de mensagens (ex: alta e baixa prioridade), configure múltiplos serviços CAP apontando para a mesma instância de broker ('binding'), cada um com sua própria configuração de fila.



Conexões Dedicadas

Para cenários de alto throughput, use `connection: dedicated: true` para forçar um serviço a usar uma conexão física separada com o broker, em vez de compartilhar a conexão padrão.

```
- name: "high-throughput-service"  
  binding: "cf-messaging-instance"  
  connection:  
    dedicated: true  
  queue:
```



Tratamento de Erros Robusto e `Acknowledgement`

O Comportamento Padrão

Uma exceção em um handler `@On` impede o `acknowledgement`. O broker irá reenviar a mensagem, o que é bom para falhas transitórias, mas perigoso para erros permanentes.

A Solução: Error Handler Customizado

Registre um handler de erro no serviço de mensageria para interceptar *qualquer* exceção ocorrida durante o processamento da mensagem. Dentro do handler, você decide se a mensagem deve ser confirmada (`acknowledge`) ou não.

```
@On(service = "messaging-name")
private void handleeError(MessagingErrorHandlerContext ctx) {
    String errorCode = ctx.getException().getErrorHandlerStatus().getCodeString();

    if (errorCode.equals(CdsErrorHandlerStatuses.INVALID_DATA_FORMAT.getCodeString())) {
        // Erro de infraestrutura, não adianta tentar de novo.
        // Logar o erro e confirmar a mensagem para removê-la da fila.
        ctx.setResult(true); // Acknowledge ←
    } else if (errorCode.equals(CdsErrorHandlerStatuses.TENANT_NOT_EXISTS.getCodeString())) {
        // Inquilino ainda não provisionado, tentar novamente mais tarde.
        ctx.setResult(false); // Do NOT acknowledge ←
    } else {
        // Erro de aplicação. Depende da sua lógica.
        // Talvez enviar para uma Dead Letter Queue?
        ctx.setResult(true); // Acknowledge por padrão para evitar loops ←
    }
}
```

✓ Confirma a mensagem, removendo-a da fila.

✗ Não confirma. O broker tentará reenviar a mensagem.

✓ Acknowledge por padrão para evitar loops.

Padrões Avançados e Capacidades do Broker

1.

Suporte a 'Acknowledgement'

Inter SemiBold

Nem todos os brokers suportam a confirmação de mensagens. A lógica do seu `handleError` depende do suporte do broker.

Messaging Broker	Suporte a 'Acknowledgement'	Causa (se não confirmado)
SAP Event Mesh	✓	Mensagem é reenviada
File Based	n/a	-
Redis PubSub (beta)	✗	Mensagem é perdida

2.

Prefixos de Tópicos

Inter SemiBold

Use `publishPrefix` e `subscribePrefix` para adicionar prefixos aos tópicos, essencial para organização e para atender a regras de nomenclatura do broker (ex: namespaces no SAP Event Mesh).

JetBrains Mono Regular

```
cds:  
  messaging:  
    services:  
      messaging-em:  
        subscribePrefix:  
          '$namespace/'
```

3.

CloudEvents

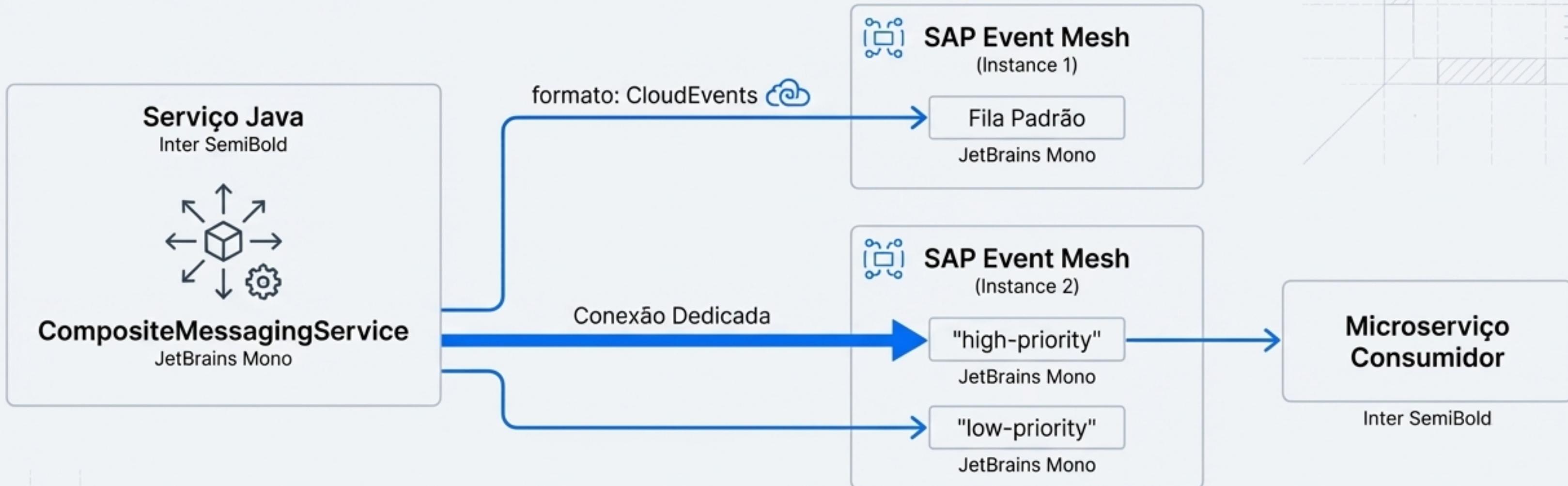
Inter SemiBold

Ative o formato padrão da indústria CloudEvents com `format: 'cloudevents'`. O CAP irá preencher automaticamente os headers padrão e estruturar o payload.

JetBrains Mono Regular

```
// Acesso type-safe a headers  
comuns  
String type =  
context.as(CloudEventMessage  
EventContext.class).getType();
```

Da Simplicidade à Maestria: Sua Arquitetura Evolutiva com CAP



Conclusão

A jornada com o SAP CAP Messaging começa com um simples 'emit' e se expande para arquiteturas de nuvem complexas, resilientes e orientadas a eventos. A abstração do CAP permite que sua lógica de negócio permaneça limpa e desacoplada, enquanto a configuração oferece o poder de adaptar, escalar e controlar com precisão o fluxo de mensagens em qualquer ambiente.