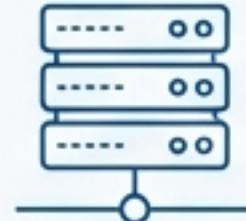


Desbravando a Mensageria Assíncrona • com CAP Node.js

Um guia para construir aplicações robustas, escaláveis e desacopladas.

A Nossa Jornada: Do Conceito à Infraestrutura



- 4. A Infraestrutura:** Conectando com Message Brokers.



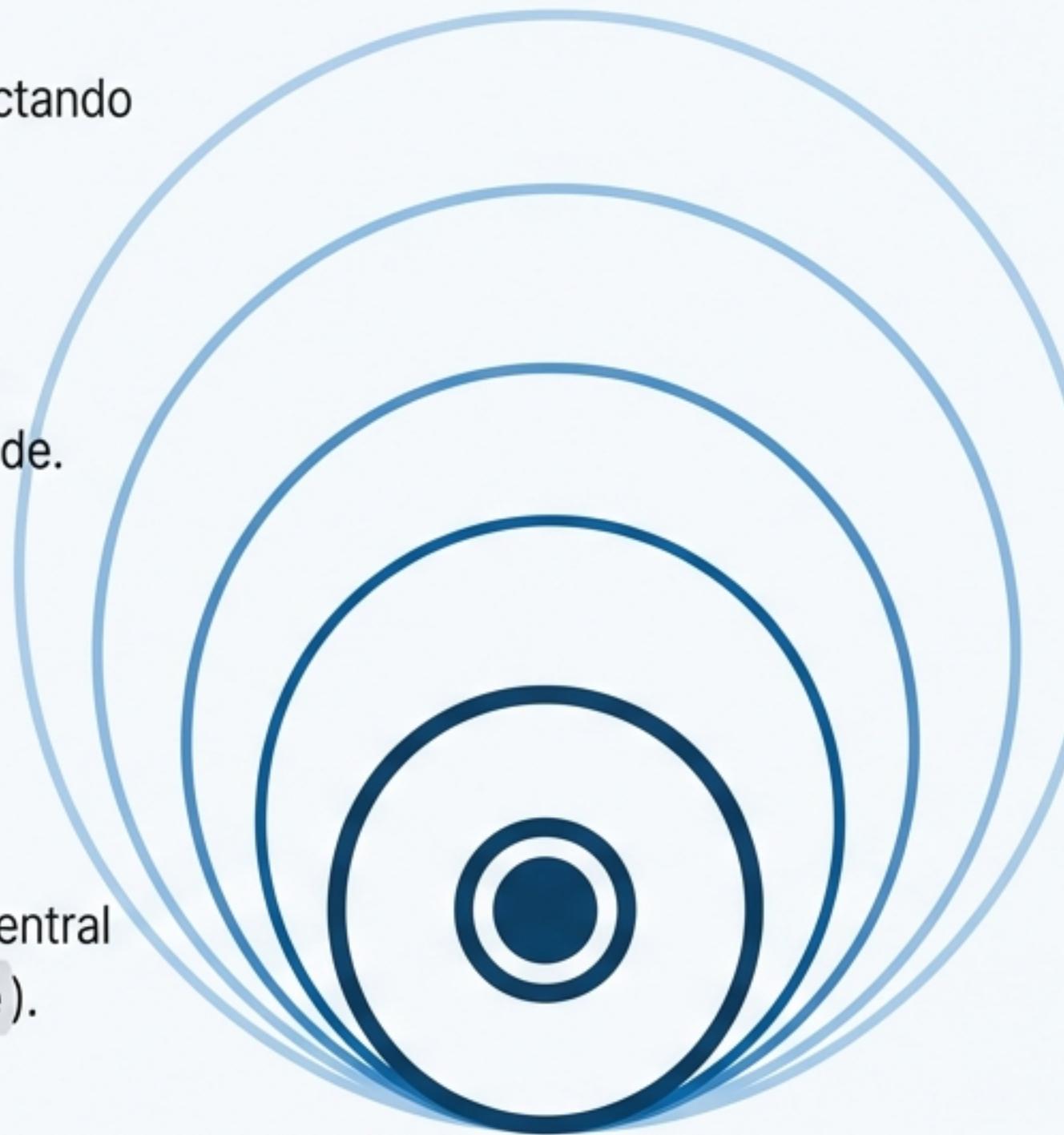
- 3. Padrões e Protocolos:** Robustez e interoperabilidade.



- 2. A Mecânica:** O ciclo de vida do evento.



- 1. O Núcleo:** A abstração central (`cds.MessagingService`).



Vamos construir seu entendimento sobre a mensageria em CAP camada por camada, garantindo uma base sólida antes de avançarmos para a implementação.

Camada 1: O Núcleo

A Abstração Central: `cds.MessagingService`

CAP trata a mensageria como um serviço técnico, unificando a forma como sua aplicação interage com diferentes canais de comunicação assíncrona, independentemente do broker por trás.

`cds.MessagingService`, que estende `cds.Service`, é a classe fundamental representando canais de mensagem. Ele atua como um adaptador universal, desacoplando a lógica de negócios da aplicação da infraestrutura de mensageria específica.

```
// Em package.json
{
  "cds": {
    "requires": {
      "messaging": {
        "kind": "enterprise-messaging" // Exemplo de broker
      }
    }
  }
}
```

Ao ativar a seção `messaging`, todos os eventos modelados no seu CDS são automaticamente habilitados para mensageria.

Camada 2: A Mecânica

O Ciclo de Vida do Evento (1/3): Declarar

Defina a estrutura e o tópico dos seus eventos diretamente no seu modelo de domínio (CDS), mantendo a coesão do seu código.

```
// Em srv/own.cds
service OwnService {
    event OwnEvent {
        ID: UUID;
        name: String;
    }
}
```

```
// Customizando o tópico
service OwnService {
    @topic: 'my.custom.topic'
    event OwnEvent { ID: UUID; name: String; }
}
```



Se a anotação @topic não for fornecida, o tópico será, por padrão, o nome totalmente qualificado do evento.

O Ciclo de Vida do Evento (2/3): Emitir

Utilize o método `emit` para publicar eventos de forma segura e consistente, atrelados ao sucesso da transação de negócio.

```
// Em srv/own.js
const messaging = await cds.connect.to('messaging');

this.after('CREATE', 'Reviews', async (_, req) => {
  const { subject } = req.data;
  await messaging.emit('review.created', { subject });
});

await messaging.emit({
  event: 'review.created',
  data: { subject },
  headers: { 'X-Correlation-ID': req.headers['X-Correlation-ID'] }
});
```



Garantia de consistência: As mensagens são enviadas ao broker somente *após* o sucesso da transação do banco de dados.

Camada 2: A Mecânica

O Ciclo de Vida do Evento (3/3): Receber

Registre `handlers` com o método `on` para processar mensagens recebidas. O CAP gerencia a criação das subscrições no broker implicitamente.

```
// Em srv/receiver.js
const messaging = await cds.connect.to('messaging');

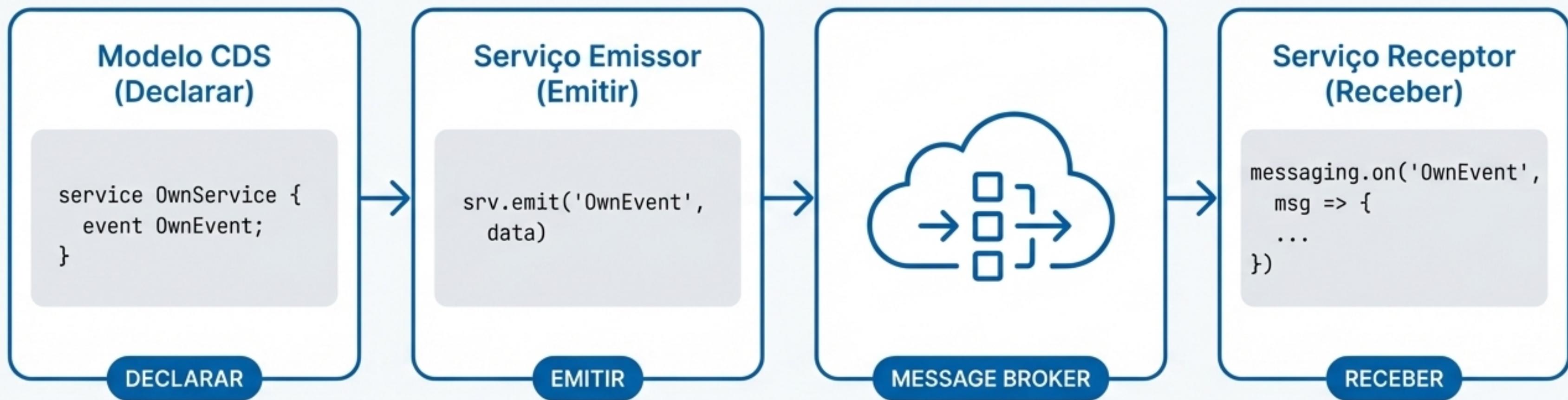
messaging.on('review.created', msg => {
  const { subject } = msg.data;
  // Lógica de processamento...
});
```

Para cenários como dead-letter queues, é possível ouvir todos os eventos com `messaging.on('*', ...)`.



Handlers de mensagens operam com um usuário técnico (`cds.User.privileged`). As checagens de autorização necessárias devem ser implementadas manualmente dentro do seu handler.

O Ciclo de Vida em Ação: O Fluxo Completo



Este fluxo de três etapas é o padrão fundamental para toda a comunicação assíncrona no CAP.

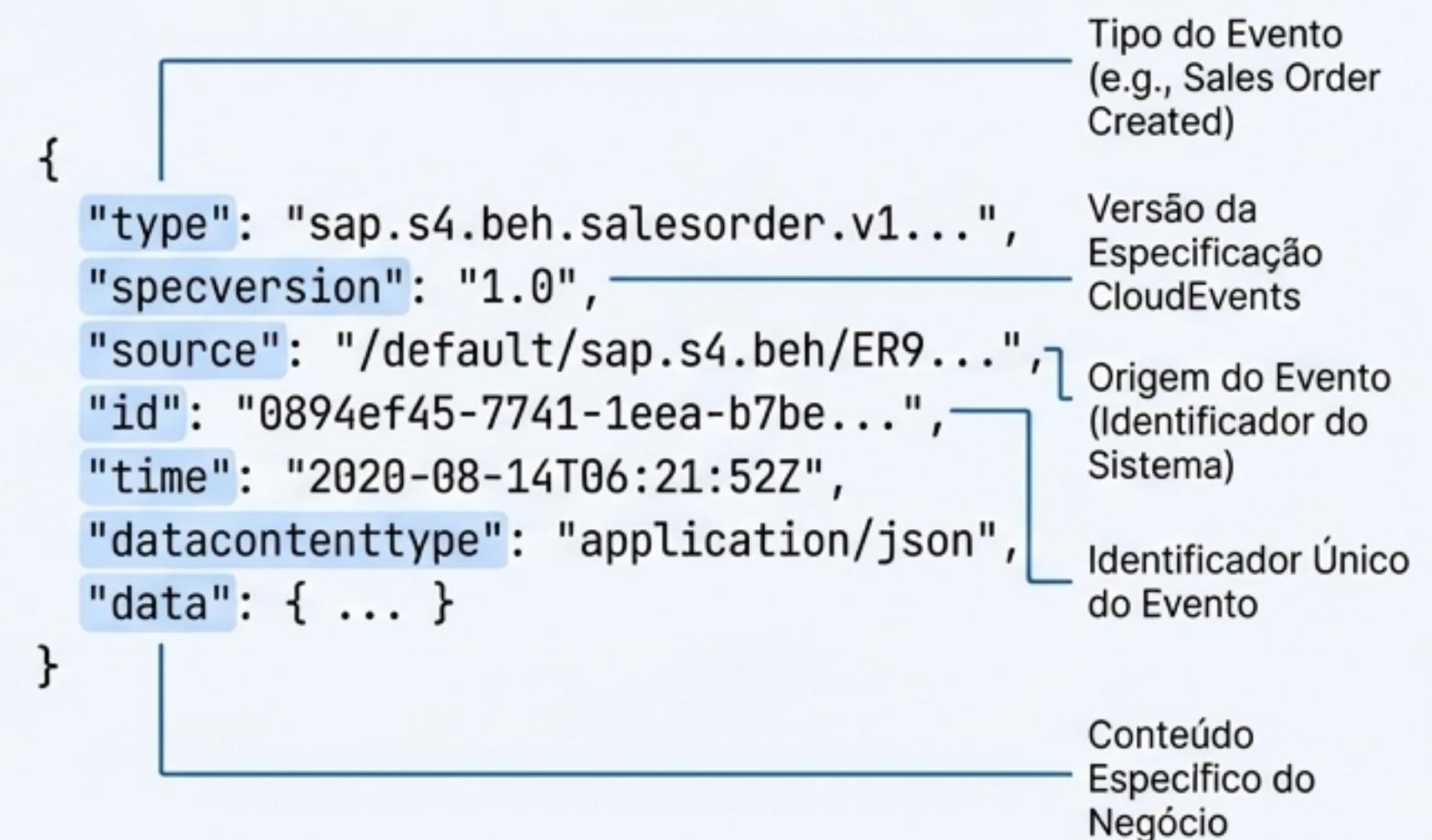
Camada 3: Padrões e Protocolos

Padronizando a Comunicação com CloudEvents

Adote o padrão da indústria para interoperabilidade com uma simples chave de configuração, sem alterar seu código de negócio.

CloudEvents é uma especificação que descreve dados de eventos em um formato comum para melhorar a interoperabilidade entre serviços, plataformas e sistemas.

```
{"messaging": {  
    "kind": "enterprise-messaging-shared",  
    "format": "cloudevents"  
}}
```



Organizando o Fluxo com Prefixos de Tópicos

Utilize `publishPrefix` e `subscribePrefix` para gerenciar namespaces e organizar suas filas de mensagens de forma limpa e escalável.

```
{  
  "messaging": {  
    "kind": "enterprise-messaging-shared",  
    "publishPrefix": "default/sap.cap/books/",  
    "subscribePrefix": "default/sap.cap/reviews/"  
}
```

Manipulação de Tópicos com CloudEvents

Ao usar `format: 'cloudevents'`, prefixos padrão são aplicados (`\$namespace/ce/` para publicar) e o framework manipula os tópicos para conformidade.



Barras (`/`) no nome do evento são substituídas por pontos (`.`).

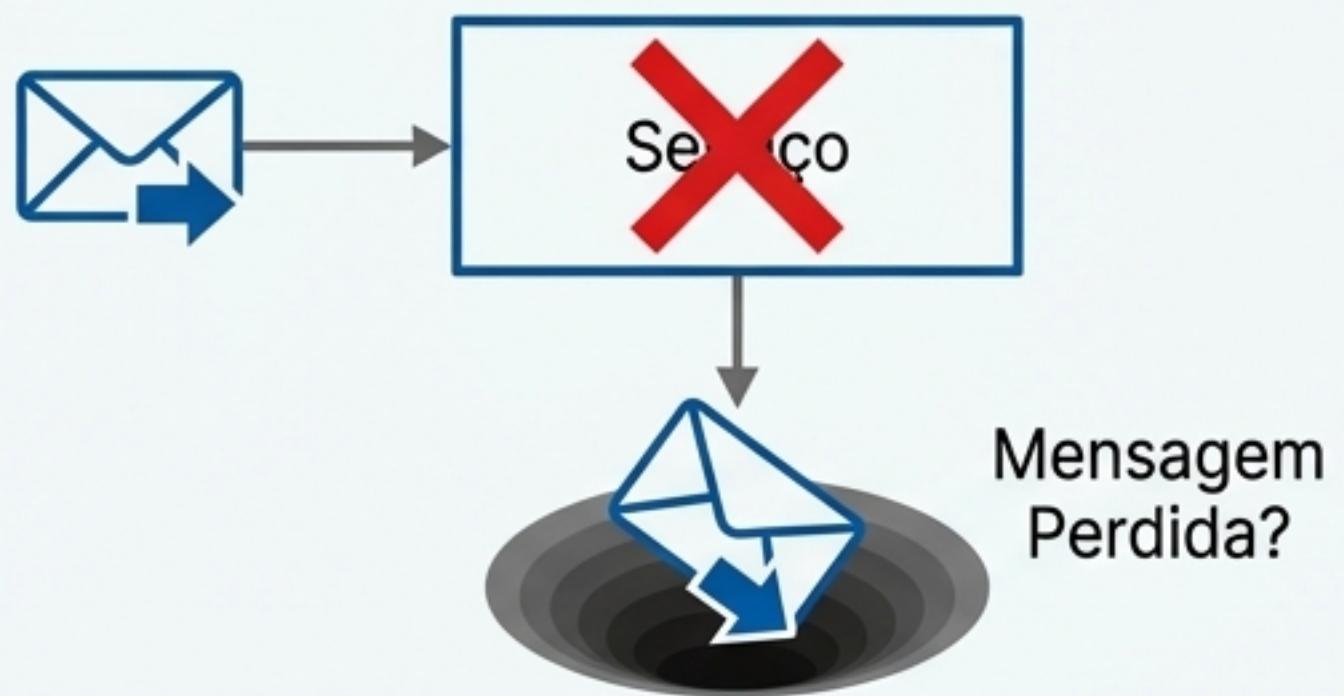
`publishPrefix: my/own/namespace/ce/` →
cabeçalho `source: /my/own/namespace`

Camada 3: Padrões e Protocolos

Garantindo o Processamento Confiável com a Inbox (beta)

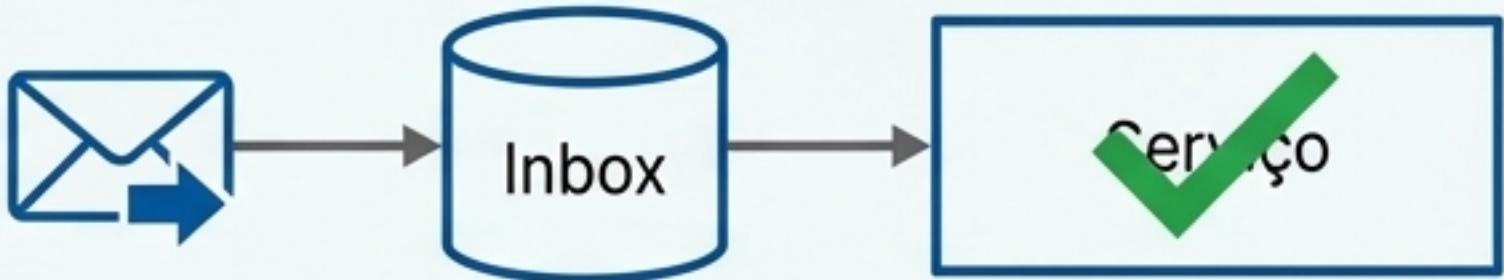
Habilite uma caixa de entrada para armazenar mensagens recebidas antes do processamento, aumentando a resiliência da sua aplicação contra falhas.

O Problema



O que acontece se seu serviço falhar ao processar uma mensagem? Sem um mecanismo de proteção, a mensagem poderia ser perdida ou entrar em um ciclo de reprocessamento infinito.

A Solução CAP



Habilitar a 'inbox' garante que a mensagem seja armazenada de forma persistente e processada de forma assíncrona e confiável.

```
"messaging": {  
    "kind": "enterprise-messaging",  
    "boxed": true  
}
```

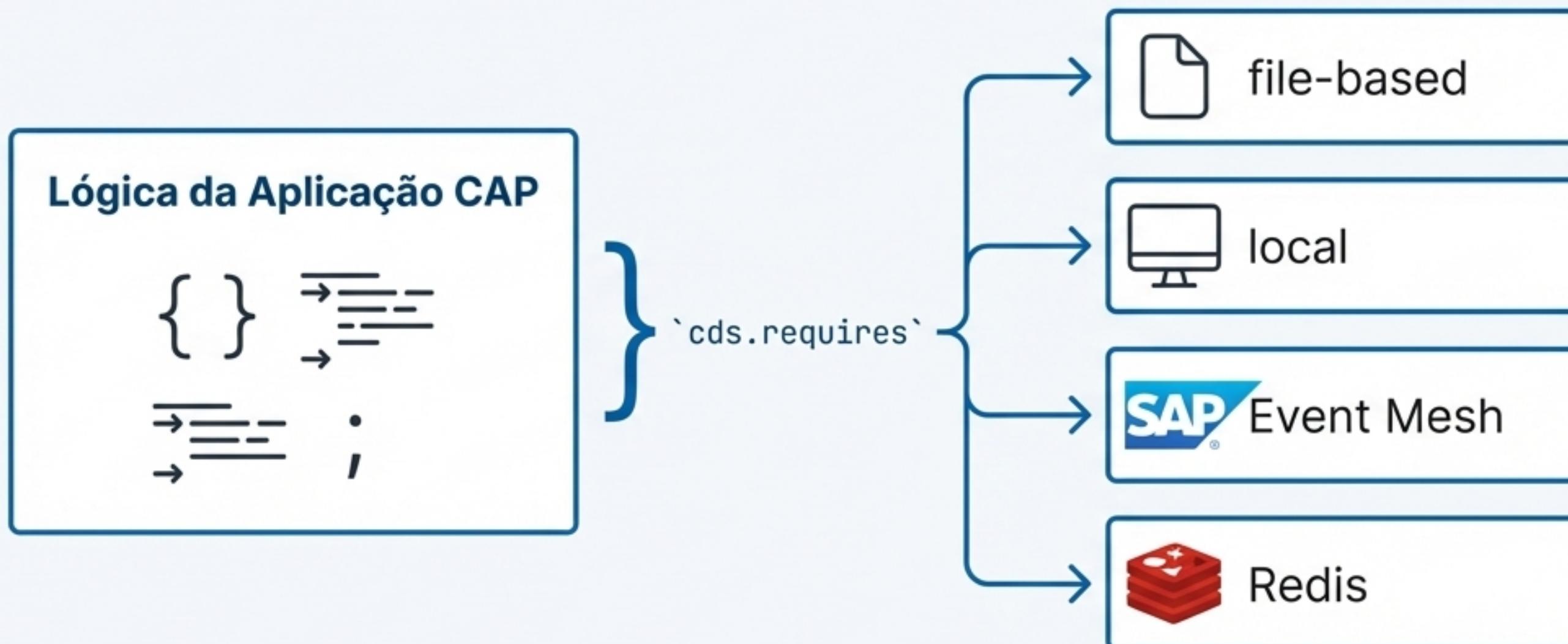
Por baixo dos panos, a 'inbox' utiliza o mecanismo de task queue do CAP para um processamento robusto.



Camada 4: A Infraestrutura

Onde as Mensagens Vivem: Message Brokers

A filosofia “Grow as you go” do CAP permite começar localmente e evoluir para soluções de mensageria de nível enterprise sem alterar a lógica do seu código.



Opções de Broker: Simplicidade para o Ambiente Local

`local-messaging`

Caso de Uso: Comunicação dentro do mesmo processo Node.js.

Ideal para: Testes automatizados e cenários muito simples.

```
"messaging": {  
  "kind": "local-messaging"  
}
```

`file-based-messaging`

Caso de Uso: Usa um arquivo local como um broker simples.

Ideal para: Simular o desacoplamento entre produtores e consumidores localmente, sem a complexidade de um broker real.

```
"messaging": {  
  "kind": "file-based-messaging",  
  "file": "../msg-box" // Opcional  
}
```



Atenção: Estas opções são destinadas exclusivamente para desenvolvimento e teste. Não utilizar em produção.

Opções de Broker: Escalando com SAP Event Mesh

`kind: enterprise-messaging-shared`

Utiliza o protocolo AMQP. É a forma recomendada e mais simples para iniciar.

 Em cenários multitenant, apenas a conta do provedor terá um event bus. Não há isolamento de tenant.

`kind: enterprise-messaging`

Utiliza HTTP. A comunicação de entrada é feita via webhooks.

 Requer configuração de `scopes` no `xs-security.json` para permitir o callback do Event Mesh. Não funciona no plano *dev*.

```
// Adicionar ao array "scopes"
{
  "name": "$XSAPPNAME.emcallback",
  "description": "Event Mesh Callback Access",
  "grant-as-authority-to-apps": ["$XSSERVICENAME(...)"
```

Opções de Broker: O Ecossistema Estendido

Redis PubSub (beta)



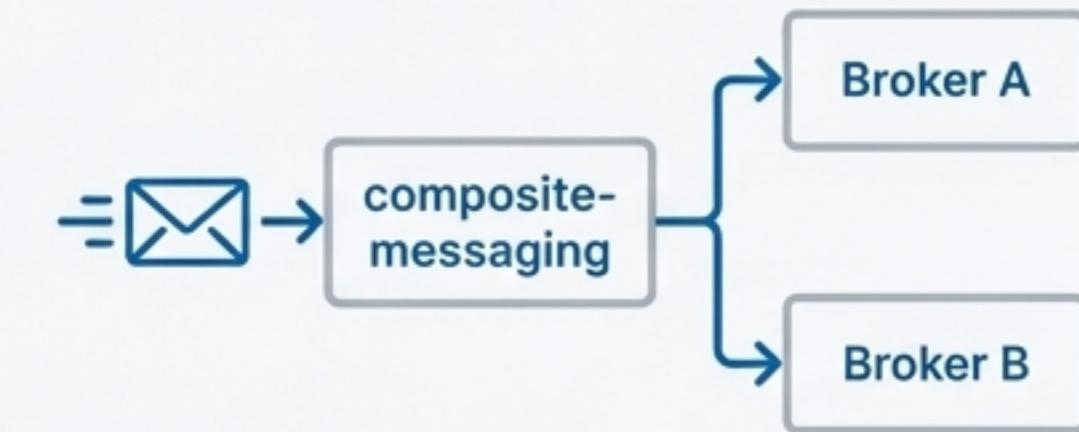
Utiliza o Redis Pub/Sub como broker. Ideal para cenários de alta velocidade e baixa latência onde a persistência de filas não é um requisito (mensagens são perdidas se os consumidores não estiverem disponíveis).

SAP Advanced Event Mesh (beta)



Para necessidades avançadas de roteamento, governança de eventos e arquiteturas complexas. Requer o plugin `@cap-js/advanced-event-mesh`.

Composite-Messaging



Uma 'meta-solução' que atua como um roteador. Permite definir rotas para que eventos, baseados em padrões de tópicos, sejam direcionados para diferentes brokers configurados.

Mensageria com CAP: Simples, Poderosa e Flexível

Pontos Chave (Key Takeaways)



- **Abstração Unificada:** O `cds.MessagingService` simplifica o código, independentemente do broker.



- **Ciclo de Vida Claro:** O padrão "Declarar, Emitir, Receber" é intuitivo e robusto.



- **Padrões Integrados:** Suporte nativo para CloudEvents e Inbox aumenta a interoperabilidade e resiliência.



- **Ecossistema Plágavel:** A filosofia "Grow as you go" é suportada por um ecossistema de brokers que cresce com a sua necessidade.



Próximos Passos (Call to Action)

Explore a documentação oficial para exemplos detalhados e tópicos avançados.

cap.cloud.sap/docs/node.js/messaging