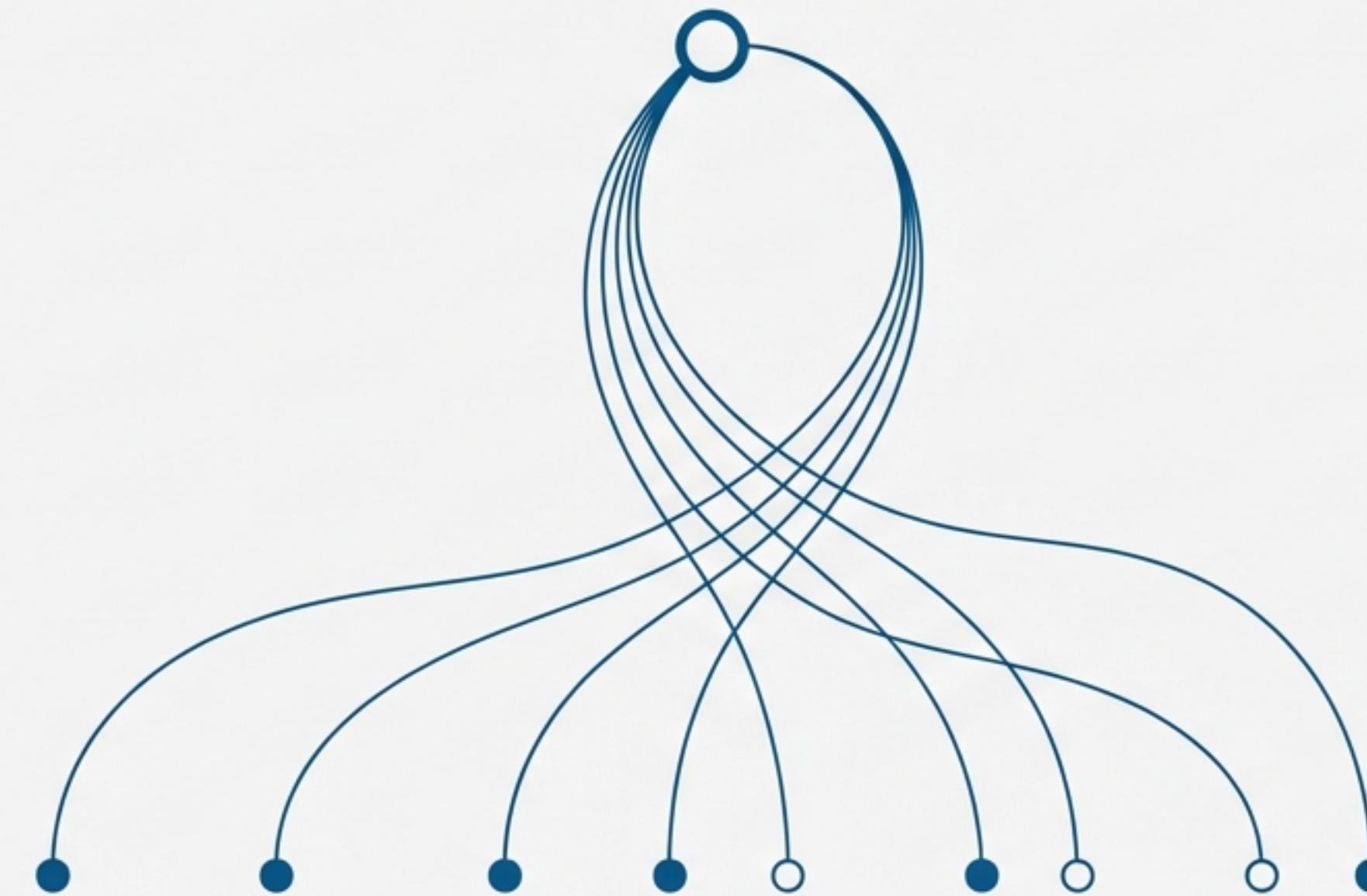


A Jornada do Desenvolvedor: Do Conceito à Conexão

Dominando o Consumo de Serviços com SAP Cloud Application Programming Model (CAP)



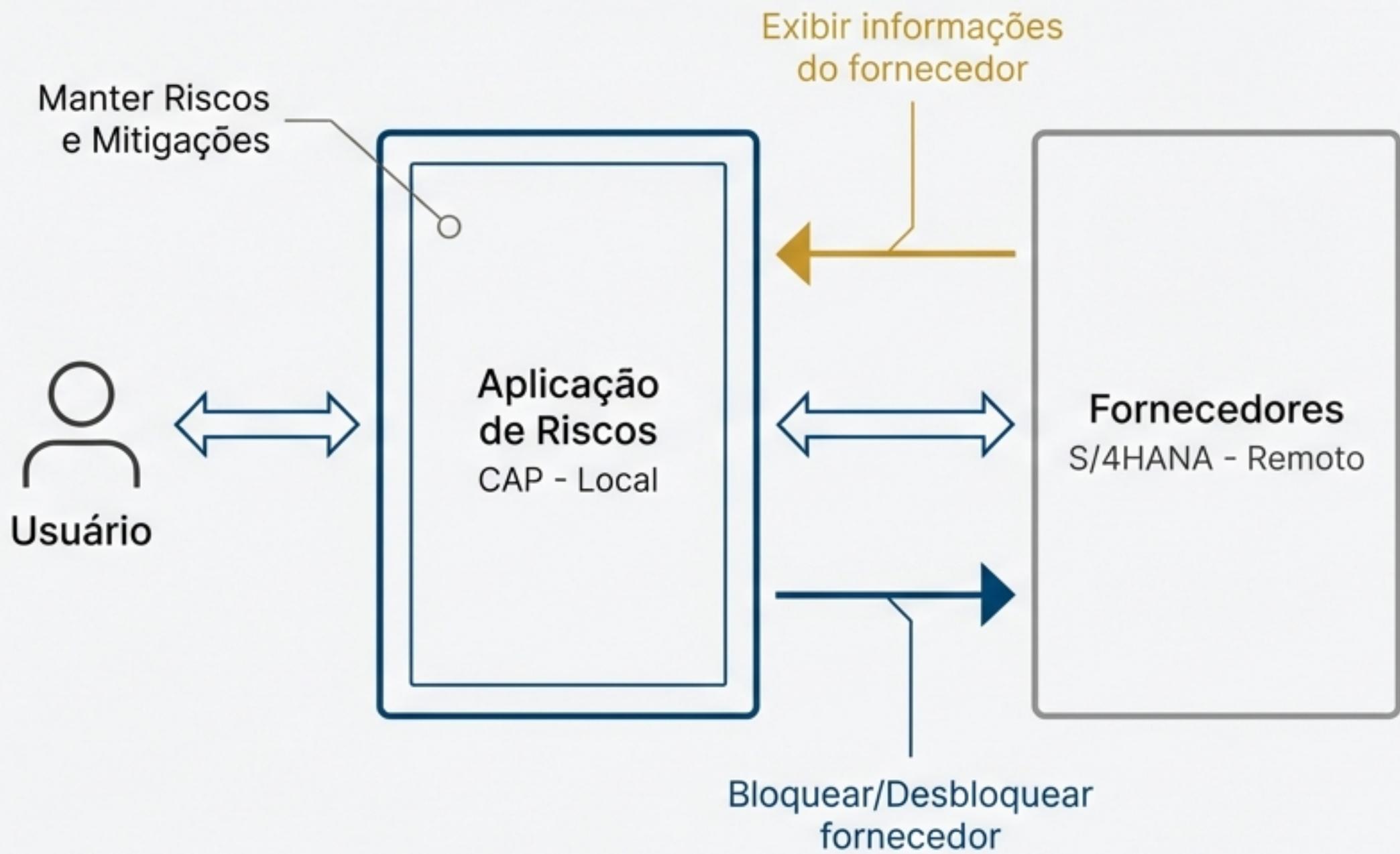
Aplicações modernas prosperam na interconexão. No universo dos microsserviços, a habilidade de consumir dados de fontes externas de forma eficiente e robusta não é apenas uma necessidade – é uma arte. Esta apresentação é o seu guia para dominar essa arte com CAP, transformando integrações complexas em soluções elegantes e resilientes.

A Missão: Estendendo o S/4HANA com Inteligência de Risco

Uma empresa precisa garantir que suas compras venham apenas de fornecedores com riscos aceitáveis.

A missão: criar uma aplicação que permita a um analista gerenciar riscos e mitigações (entidades locais), associando-os a fornecedores (dados remotos do S/4HANA Cloud).

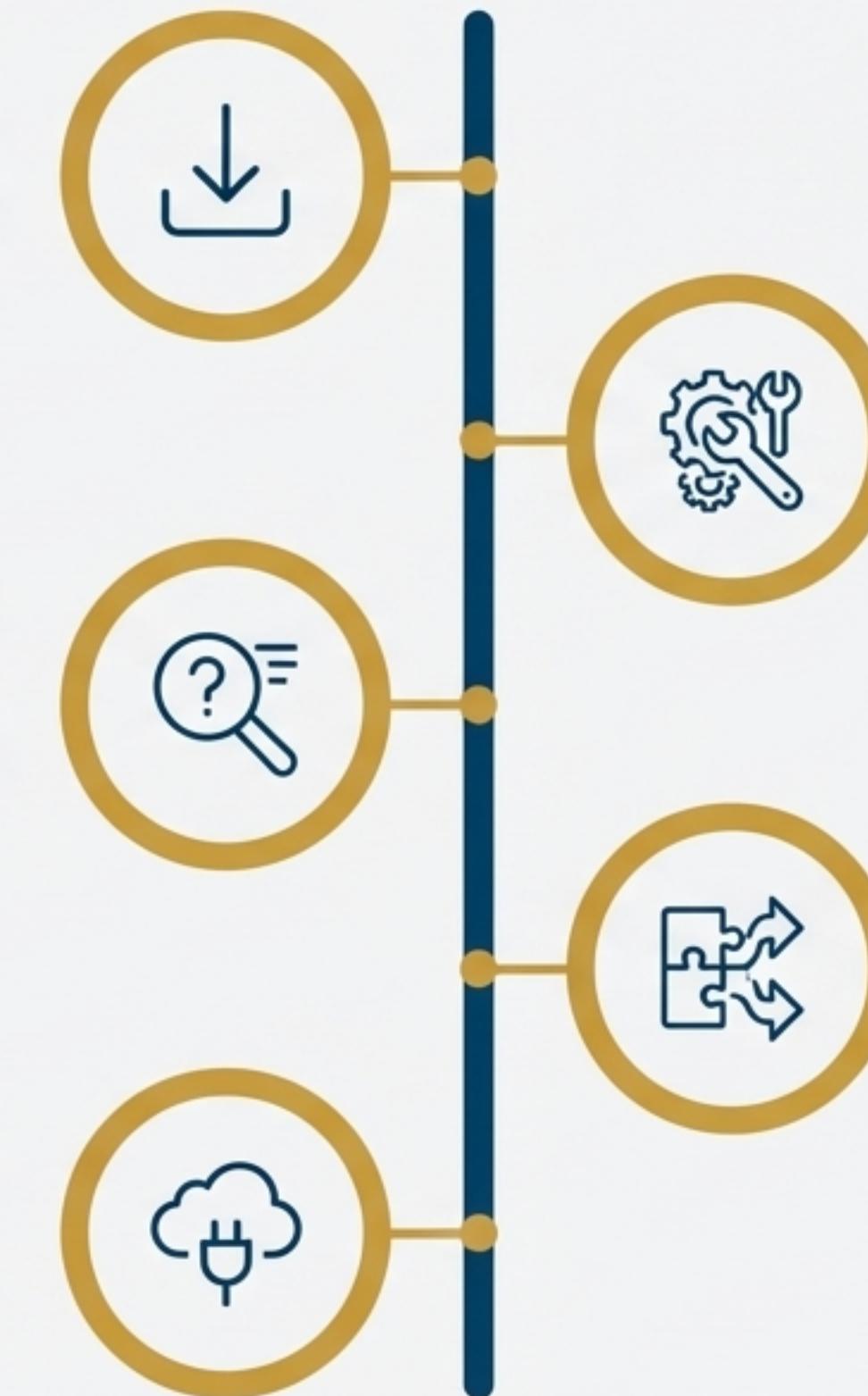
A aplicação deve exibir informações do fornecedor e ser capaz de bloqueá-lo automaticamente com base no nível de risco.



O Mapa da Jornada em 5 Etapas

Etapa 1: Importar o Mundo Exterior

Obter e importar a definição da API do serviço remoto para o nosso projeto CAP.



Etapa 3: A Arte da Conversa

Aprender a consultar o serviço remoto usando la API de querying do CAP e as projeções de modelo.

Etapa 2: Construir um Simulacro Local

Usar o poderoso mocking do CAP para desenvolver de forma ágil e independente, sem depender do serviço real.

Etapa 4: A Grande Fusão (Mashups)

Combinar dados locais e remotos em um único serviço coeso, criando uma experiência de usuário unificada.

Etapa 5: Conectar ao Mundo Real

Configurar a conexão para produção usando Destinations e implantar a aplicação.

Etapa 1: Importando o Mundo Exterior com `cds import`

Para que o CAP possa interagir com um serviço remoto, ele primeiro precisa entender sua ‘linguagem’ – sua definição de API. O primeiro passo é importar essa definição.

Fontes da API

- **SAP Business Accelerator Hub:** Baixe o arquivo EDMX da API desejada (ex: Business Partner API do S/4HANA Cloud).
- **Outro Serviço CAP:** Gere a definição com o comando `cds compile srv --to edmx`.

O Comando Mágico

```
cds import srv/external/API_BUSINESS_PARTNER.edmx --as cds
```

O Resultado

O comando `cds import` realiza duas ações cruciais:

1. ***Cria um arquivo `*.cds`:** Adiciona uma versão CDS da API em `srv/external/`.
2. ***Atualiza o `package.json`:** Registra o serviço como uma dependência externa (`requires`), preparando o terreno para a conexão futura.

```
"cds": {  
  "requires": {  
    "API_BUSINESS_PARTNER": {  
      "kind": "odata",  
      "model": "srv/external/API_BUSINESS_PARTNER"  
    }  
  }  
}
```

Etapa 2: Desenvolvendo em um Ambiente Controlado com Mocking

O “Porquê” do Mocking

Desenvolver dependendo de um sistema remoto real é lento e frágil. O CAP resolve isso com um servidor de mock integrado que é ativado automaticamente, permitindo um ciclo de desenvolvimento rápido e independente.

Passo 1: Fornecer Dados de Amostra

Crie um arquivo `*.csv` no diretório `srv/external/data/`. O nome do arquivo deve seguir o padrão: `<NomeDoServiço>-<NomeDaEntidade>.csv`.

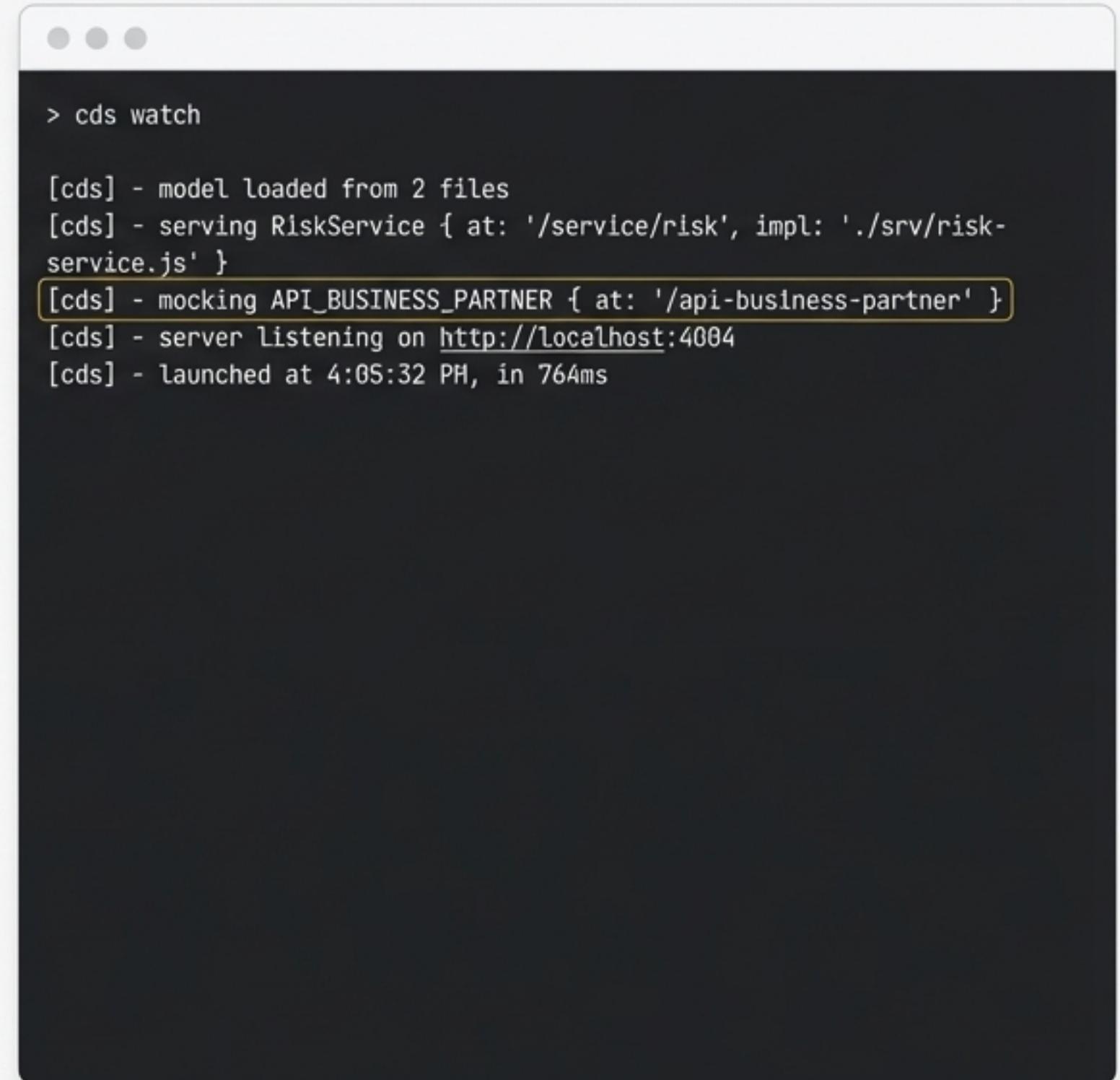
Exemplo de Código (CSV)

```
// File: srv/external/data/API_BUSINESS_PARTNER-A_BusinessPartner.csv
BusinessPartner,BusinessPartnerFullName,BusinessPartnerIsBlocked
1004155,"Williams Electric Drives",false
1004161,"Smith Batteries Ltd",false
1004100,"Johnson Automotive Supplies",true
```

Passo 2: Iniciar o Servidor

Execute o comando: `cds watch`

****O que acontece?**** O CAP detecta a definição do serviço externo, encontra os dados CSV e automaticamente inicia um endpoint mockado para ele.



```
> cds watch
[cds] - model loaded from 2 files
[cds] - serving RiskService { at: '/service/risk', impl: './srv/risk-service.js' }
[cds] - mocking API_BUSINESS_PARTNER { at: '/api-business-partner' }
[cds] - server listening on http://localhost:4004
[cds] - launched at 4:05:32 PM, in 764ms
```

Aprofundando o Mocking: Associações e Processos Separados

Cenário 1: Mockando Associações

O Desafio: Por padrão, as associações em serviços importados não são resolvidas, pois a condição de junção (`on`) está ausente.

A Solução: Modifique o arquivo `*.cds` importado para adicionar a condição `on` manualmente. Isso ensina ao mock como conectar as entidades.

Antes

```
// Em API_BUSINESS_PARTNER.cds
entity A_BusinessPartner {
  ...
  to_BusinessPartnerAddress : Association to many A_BusinessPartnerAddress { };
}
```

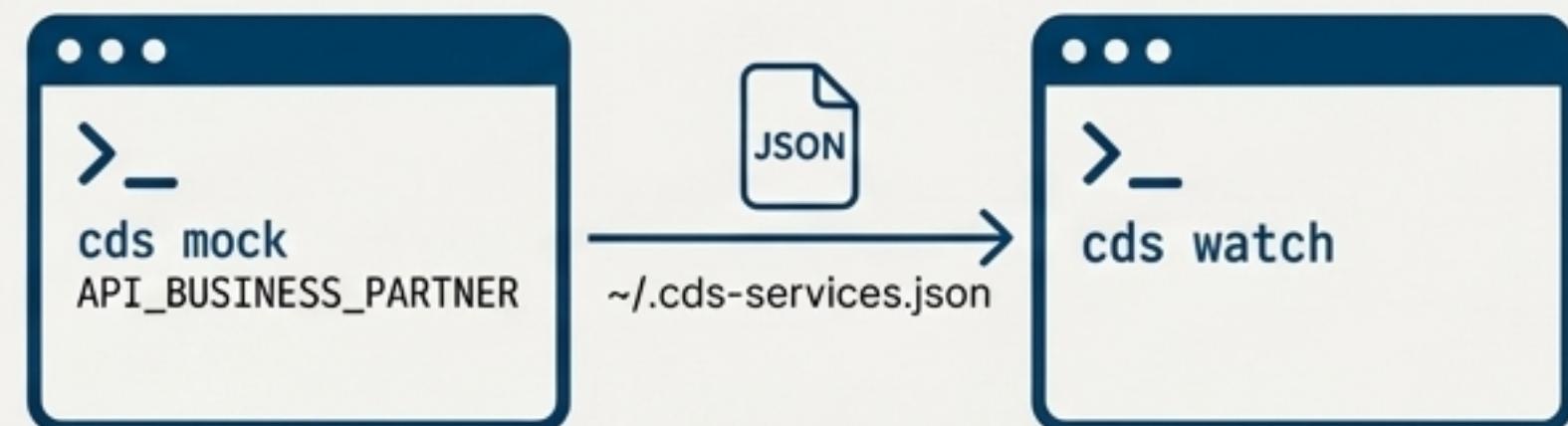
Depois

```
// Com a condição 'on' adicionada
entity A_BusinessPartner {
  ...
  to_BusinessPartnerAddress : Association to many A_BusinessPartnerAddress
    on to_BusinessPartnerAddress.BusinessPartner = BusinessPartner;
}
```

Depois

Cenário 2: Mock Realista como um Serviço OData

A Vantagem: Para simular a comunicação de rede real entre microsserviços, execute o mock em um processo separado. `cds watch` detecta o serviço rodando localmente e se conecta a ele, simulando uma chamada de API real.



Terminal 1: Inicie o serviço de mock

Terminal 2: Inicie sua aplicação

`cds watch` detecta o serviço rodando localmente (via `~/.cds-services.json`) e se conecta a ele, simulando uma chamada de API real.

Etapa 3: A Arte da Conversa com a Querying API

O CAP oferece uma API de consulta fluente e unificada (CQN) para interagir com qualquer serviço, local ou remoto. A sintaxe é a mesma, abstraindo a complexidade da fonte de dados.

O Processo em Código (Node.js)

Passo 1: Conectar ao Serviço

Primeiro, obtenha uma referência ao serviço remoto.



```
const bupa = await cds.connect.to('API_BUSINESS_PARTNER');
```

Passo 2: Executar a Consulta

Use a sintaxe `SELECT` para construir  e executar sua query.

```
// Desestruturar a entidade para facilitar o uso
const { A_BusinessPartner } = bupa.entities;

// Executar a consulta para buscar os 100 primeiros parceiros
const result = await bupa.run(
  SELECT.from(A_BusinessPartner).limit(100)
);
```

Passo 3: Resolvendo Associações

A API permite navegar por associações de forma intuitiva.

```
const result = await bupa.run(SELECT.from(A_BusinessPartner, bp => {
  bp('BusinessPartner'),
  bp.to_BusinessPartnerAddress(addr => { addr('*') })
}));
```

Esta API é a base para todas as interações de dados. Dominá-la é fundamental.

Criando sua Própria Lente: A Importância das Projeções

O “Porquê”

Acoplar sua lógica de negócios diretamente à estrutura de um serviço remoto é arriscado. A API externa pode mudar. A melhor prática é criar uma ‘fachada’ ou ‘interface’ para o serviço externo usando projeções.

Benefícios



Desacoplamento: Isola sua aplicação de mudanças no serviço remoto.



Clareza: Define explicitamente quais campos são relevantes para sua aplicação.



Eficiência: Solicita apenas os dados que você realmente precisa.

Como Fazer

Defina uma nova entidade em seu serviço local como uma projeção (`projection on`) da entidade remota. Use aliases para renomear campos, criando uma interface mais limpa e alinhada ao seu domínio.

```
using { API_BUSINESS_PARTNER as bupa } from '../srv/externa'  
  
// Projeção local 'Suppliers' sobre a entidade remota 'A_Bus:  
entity Suppliers as projection on bupa.A_BusinessPartner {  
    key BusinessPartner as ID, // Renomeia 'BusinessPartner'  
    BusinessPartnerFullName as fullName,  
    BusinessPartnerIsBlocked as isBlocked  
}
```

Consultando Através da Lente: A Magia do Mapeamento Automático

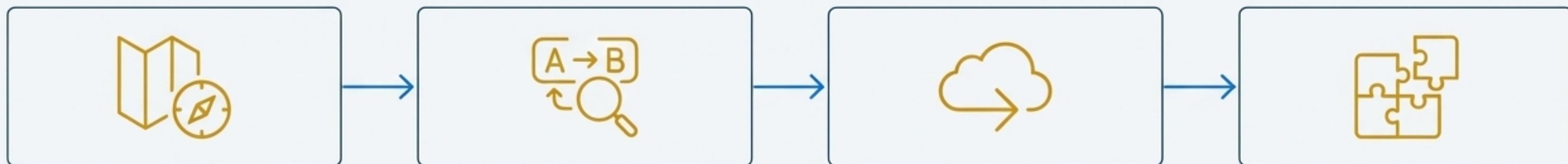
Uma vez que sua projeção está definida, usá-la é tão simples quanto consultar uma entidade local. O CAP cuida de toda a tradução para você.

Código de Consulta Simplificado

```
// Conecta ao serviço remoto
const bupa = await cds.connect.to('API_BUSINESS_PARTNER');

// Executa a consulta na NOSSA projeção 'Suppliers'
const suppliers = await bupa.run(SELECT.from(Suppliers).where({ ID: '1004155' }));
```

O que o CAP Faz nos Bastidores



Resolve a Projeção

Entende que `Suppliers` é uma projeção de `API_BUSINESS_PARTNER.A_BusinessPartner`.

Mapeia a Cláusula `where`

Traduz a condição `where({ ID: ... })` para `where({ BusinessPartner: ... })`.

Executa a Query Remota

Envia a consulta traduzida para o serviço externo.

Mapeia o Resultado

Converte a resposta, mapeando `BusinessPartner` de volta para `ID`, `BusinessPartnerFullName` para `fullName`, etc.

Etapa 4: A Grande Fusão - Criando Mashups de Serviços

O Objetivo

O verdadeiro poder emerge quando combinamos dados de diferentes fontes. Um ‘mashup’ de serviço integra dados locais e remotos, apresentando-os como uma única API coesa para o cliente.

Cenário de Partida: Expor um Serviço Remoto

Para começar, vamos simplesmente expor a lista de fornecedores remotos através do nosso `RiskService`.

```
// Em risk-service.cds
using { API_BUSINESS_PARTNER as bupa } from '../srv/external/API_BUSINESS_PARTNER';

service RiskService {
    // ... outras entidades locais ...
    entity BusinessPartners as projection on bupa.A_BusinessPartner;
}
```



Passo 2: Criar o Handler de Delegação (Node.js)



O Problema

O CAP tentará buscar `BusinessPartners` no banco de dados local e falhará.

A Solução

Precisamos interceptar a requisição ('on READ') e delegá-la manualmente para o serviço remoto.

```
// Em risk-service.js
module.exports = cds.service.impl(async function() {
    const bupa = await cds.connect.to('API_BUSINESS_PARTNER');

    this.on('READ', 'BusinessPartners', req => {
        // Simplesmente repassa a query recebida (req.query) para o serviço remoto
        return bupa.run(req.query);
    });
});
```



O Desafio da Fusão: Navegando entre o Local e o Remoto

Quando associações cruzam a fronteira entre o banco de dados local e um serviço remoto, o CAP precisa de ajuda. `Expands` e navegações entre essas fontes diferentes devem ser implementados manualmente em handlers customizados.

Matriz de Implementação para Mashups

Cenário da Requisição	Exemplo de URL	Implementação Necessária
Local para Local	/risks/Risks	Automático pelo CAP. Nenhuma ação necessária.
Remoto para Remoto	/risks/Suppliers?\$expand=addresses	Handler de delegação simples. O serviço remoto resolve o expand.
Local com `expand` para Remoto	/risks/Risks?\$expand=supplier	Handler customizado no `READ` de `Risks`. É preciso buscar os riscos, extrair os IDs dos fornecedores e fazer uma segunda chamada ao serviço remoto para buscar os dados do expand.
Navegação de Local para Remoto	/risks/Risks(ID)/supplier	Handler customizado no `READ` de `Suppliers`. É preciso detectar a navegação, extraír o ID do risco, e buscar o fornecedor correspondente.
Remoto com `expand` para Local	/risks/Suppliers?\$expand=risks	Handler customizado no `READ` de `Suppliers`. Similar ao `expand` local->remoto, mas na direção oposta.

Mergulho Profundo: Implementando um `expand` Local -> Remoto

Cenário

O usuário solicita uma lista de `Risks` (locais) e quer ver os detalhes do `supplier` (remoto) para cada um.

URL da Requisição:

`GET /service/risk/Risks?\$expand=supplier`

Algoritmo no Handler `on READ Risks`

1. **Detectar e Isolar o `expand`:** Verifique se `req.query.SELECT.columns` contém uma expansão para `supplier`. Se sim, remova-a temporariamente da query principal.
2. **Buscar os Dados Principais:** Execute a query modificada para buscar os dados da entidade `Risks` do banco de dados local.

```
const risks = await SELECT.from(Risks).where(...)
```

3. **Coletar as Chaves Estrangeiras:** A partir do resultado `risks`, crie um array com todos os `supplier_ID` únicos.
4. **Buscar os Dados do `expand`:** Faça uma única chamada ao serviço remoto, buscando todos os fornecedores cujos IDs estão no array.

```
const suppliers = await bupa.run(SELECT.from(Suppliers).where({ ID: { in: supplierIDs } }))
```

5. **Combinar os Resultados:** Anexe os dados de `suppliers` aos registros correspondentes de `risks` antes de retornar o resultado final.



Cuidado com `expands` em listas grandes. Isso pode gerar queries com muitos IDs na cláusula `IN`. Considere se o `expand` é realmente necessário na visualização de lista ou apenas na de detalhes.

Etapa 5: Conectando ao Mundo Real com Destinations

O Que São Destinations?

Em produção, não usamos URLs e credenciais fixas no código. Usamos **Destinations**, que são essencialmente URLs avançadas gerenciadas na BTP. Elas contêm o **endpoint**, o tipo de autenticação e outras informações de conexão de forma segura e configurável.

Tipos de Destination

- **BTP Destinations:** Gerenciadas centralmente na sua subconta BTP. O método preferencial.
- **Application-defined Destinations:** Propriedades de conexão definidas diretamente na configuração da aplicação (útil para cenários específicos).

Configuração no `package.json`

A conexão é definida usando um **perfil de produção** (`[production]`). Isso garante que a configuração de destination só seja usada quando a aplicação for implantada, não durante o desenvolvimento local com mocks.

```
"cds": {  
  "requires": {  
    "API_BUSINESS_PARTNER": {  
      "kind": "odata",  
      "model": "srv/external/API_BUSINESS_PARTNER",  
      "[production)": {  
        "credentials": {  
          "destination": "S4HANA", // Nome do Destination na BTP  
          "path": "/sap/opu/odata/sap/API_BUSINESS_PARTNER"  
        }  
      }  
    }  
  }  
}
```

A Conexão Final: Teste Local e Implantação

Validando a Conexão Localmente (Hybrid Testing)

Você não precisa implantar para testar a conexão com um sistema real. O comando `cds bind` permite que sua aplicação local use serviços da BTP.

Fluxo de Teste Híbrido

1. Crie instâncias dos serviços `xsuaa` e `destination` na BTP.
2. Execute `cds bind -2 cpapp-xsuaa, cpapp-destination`.
3. Configure um perfil [hybrid] no seu `.cdsrc-private.json` para apontar para o `destination`.
4. Inicie sua aplicação. Ela agora se comunicará com o sistema remoto real.

Requisitos para Implantação (MTA)

Para que sua aplicação funcione na nuvem, ela precisa de ‘bindings’ para os serviços essenciais da plataforma. Adicione os serviços necessários ao seu projeto com um único comando:

```
cds add xsuaa,destination,connectivity
```



xsuaa: Para gerenciamento de autenticação e autorização.

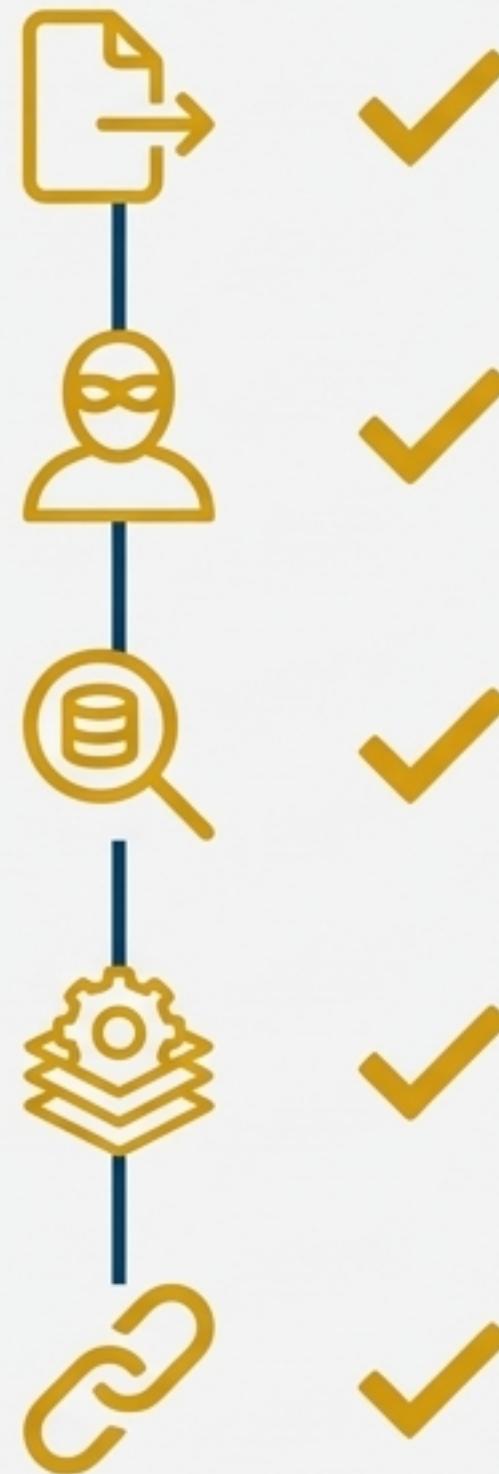


destination: Para encontrar e usar os destinations configurados.



connectivity: Necessário para se conectar a sistemas on-premise através do Cloud Connector.

A Jornada Completa: De um Desafio a uma Solução Conectada



Recapitulando a Conquista

Partimos de um cenário de negócio claro e, passo a passo, construímos uma solução robusta.

- **Importamos** a definição de um serviço externo, criando uma base sólida.
- **Mockamos** o serviço para um desenvolvimento local rápido e independente.
- **Consultamos** os dados de forma elegante, usando o poder das projeções para criar uma interface limpa e desacoplada.
- **Integramos** dados locais e remotos em mashups complexos, com controle total sobre o fluxo de dados.
- **Conectamos** nossa aplicação ao mundo real de forma segura e configurável para produção.

Mensagem Final

O SAP Cloud Application Programming Model transforma a complexa tarefa de consumir serviços em uma jornada estruturada e produtiva. Com as ferramentas e padrões certos, você está equipado para construir a próxima geração de aplicações empresariais conectadas.