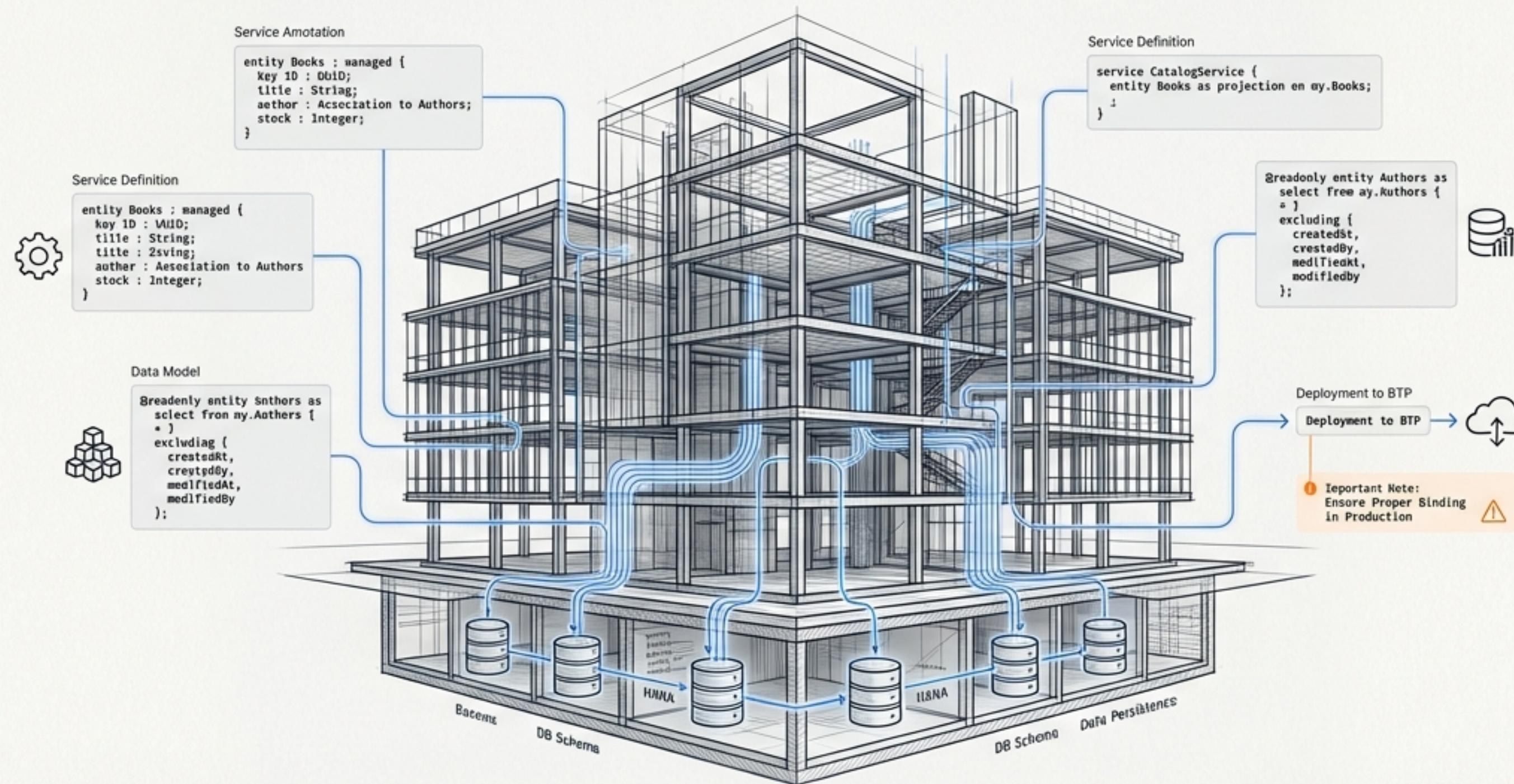
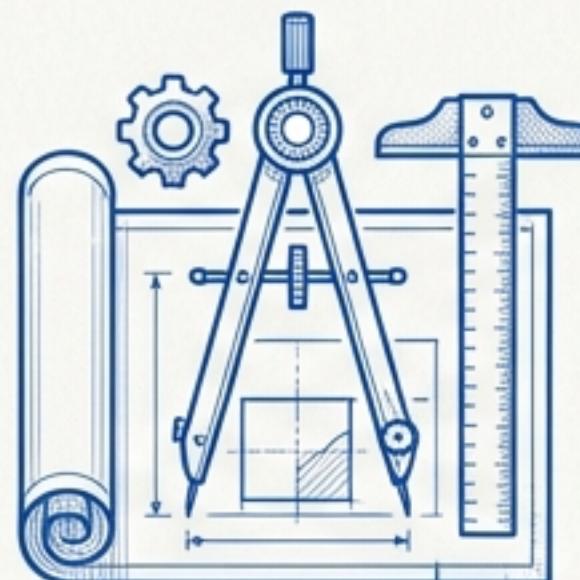


# Dominando o Banco de Dados no CAP: A Jornada para Aplicações Enterprise-Ready

Da modelagem de dados à implantação em produção: um guia prático e arquitetural.

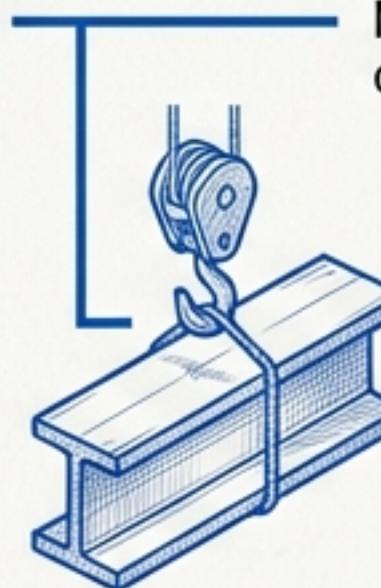


# A Jornada do Desenvolvedor CAP



## 1. O Alicerce (A Fundação)

Modelagem de Domínio com CDS. Foco na captura da intenção de negócio e na criação de um modelo de dados limpo e conciso.



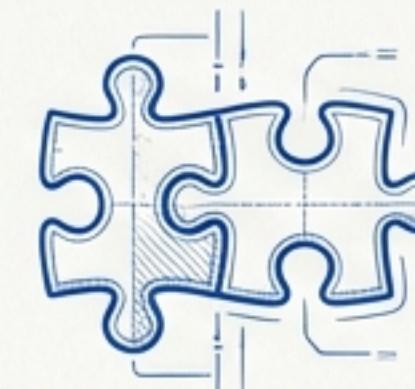
## 2. A Estrutura (Os Pilares)

Escolhendo o Banco de Dados Certo. Análise das opções para desenvolvimento (SQLite), produção (PostgreSQL, SAP HANA Cloud) e integração nativa.



## 3. O Reforço (As Qualidades)

Adicionando Capacidades Enterprise. Implementação de internacionalização (i18n), segurança robusta e otimização de performance.



## 4. O Ecossistema (As Conexões)

Integração e Consumo de Serviços. Utilizando dados de fontes externas para enriquecer a aplicação.



## 5. O Selo de Qualidade (A Inspeção Final)

Testando a Aplicação. Garantindo a robustez e a confiabilidade da construção com testes automatizados.

# Fase 1: O Alicerce - Capturar Intenção, Não Implementação

“O CDS foca na modelagem conceitual: queremos capturar a intenção, **não implementações imperativas**. — ou seja: **O quê, não Como.**”

## O REQUISITO

Queremos uma livraria onde usuários possam navegar por **Livros** e **Autores**, e navegar de **Livros** para **Autores** e vice-versa. **Livros** são classificados por **Gênero**.



## O MODELO CDS

```
using { cuid } from '@sap/cds/common';

entity Books : cuid {
    title : String;
    genre : Genre;
    author : Association to Authors;
}

entity Authors : cuid {
    name : String;
    books : Association to many Books on books.author = $self;
}

type Genre : String enum {
    Mystery; Fiction; Drama;
}
```



O uso de `cuid` e `Association` demonstra como o CAP captura a intenção (uma chave única, um relacionamento) e deixa os detalhes da implementação (geração de UUID, chaves estrangeiras) para o framework.

# As Vigas Mestras: Chaves, Aspectos e Relacionamentos

## Chaves Primárias Universais (`cuid`)

**Por quê UUIDs?** "UUIDs são universais... permitem sementes distribuídas... e são preenchidos automaticamente."

Evite validações desnecessárias de formatos de UUID. A única suposição permitida é que são únicos.

```
entity Books : cuid { ... }
```

é a forma concisa de

```
entity Books {  
    key ID : UUID;  
    ...  
}
```

## Dados Gerenciados (`managed`)

**O que faz?** Adiciona automaticamente quatro campos essenciais para auditoria: `createdAt`, `createdBy`, `modifiedAt`, `modifiedBy`.

**Como funciona?** O framework preenche esses campos usando as pseudovariáveis `\$now` e `\$user`, garantindo consistência.

```
entity Foo : managed {  
    ...  
}
```

## Associações vs. Composições

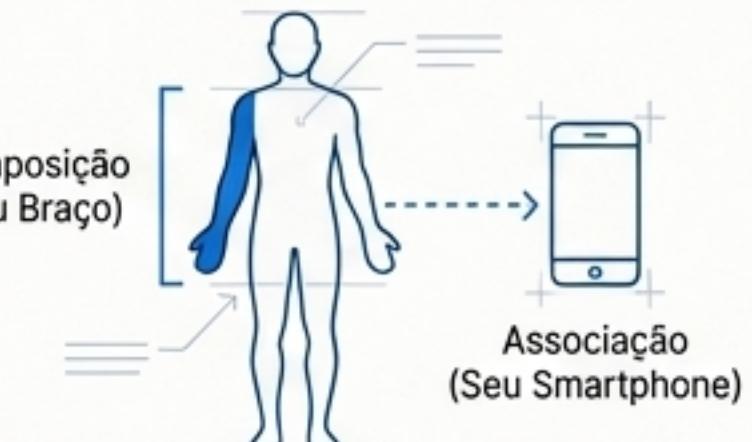
**Associação:** Relacionamento entre entidades independentes.

Ex: `author : Association to Authors;`

**Composição:** Relacionamento pai-filho com ciclos de vida acoplados.

Ex: `Items : Composition of many OrderItems;`

**Regra de Ouro:** "Seu braço é uma composição do seu corpo; seu smartphone é uma associação a você, pois amanhã pode pertencer a outra pessoa."



# Fase 2: A Estrutura - O Banco de Dados Certo para Cada Etapa

Com o modelo de dados definido, o CAP pode gerar os artefatos de banco de dados (DDL) para diferentes sistemas. A escolha depende do ambiente e do caso de uso.

## 1. Desenvolvimento Rápido: SQLite

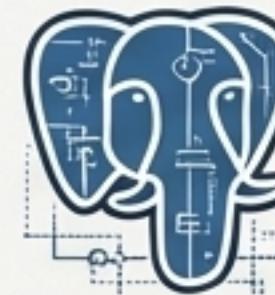


O padrão para desenvolvimento local. É leve, baseado em arquivo e não requer instalação de servidor (in-memory). Ideal para prototipagem e testes rápidos.

**Comando Chave:** `cds deploy --to sqlite`

Note que o SQLite não suporta **locales** como o **SAP HANA**.

## 2. Alternativa Robusta: PostgreSQL



Uma opção de código aberto popular e poderosa para desenvolvimento e produção. Oferece mais funcionalidades que o SQLite.

**Configuração:** Requer a instalação do driver (`npm i pg`) e configuração no `package.json` para direcionar o `cds.requires.db` para `hanaPg`.

## 3. Produção Enterprise: SAP HANA Cloud

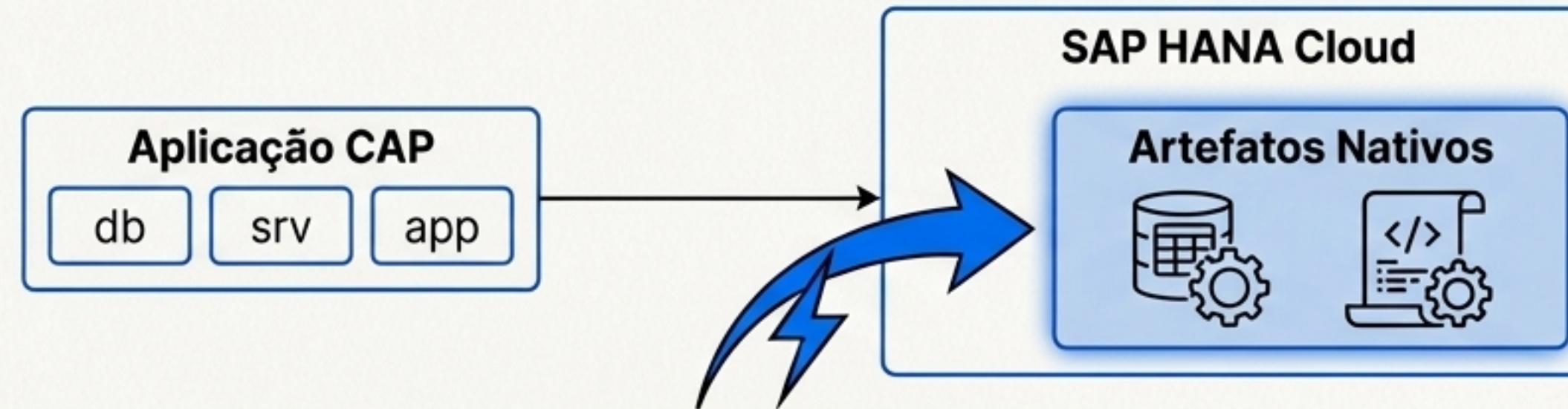


A solução recomendada para produção. Oferece alta performance, escalabilidade e integração nativa com o ecossistema SAP BTP.

**Conceito Chave:** Utiliza HDI Containers para garantir isolamento de dados por tenant em cenários multi-tenant.

**Integração Nativa:** Permite o uso de artefatos nativos do HANA (Calculation Views, Stored Procedures) para otimizações avançadas.

# Maximizando o Potencial com SAP HANA Cloud



## A Abordagem Padrão

- O CAP gera DDL a partir de modelos CDS.
- O runtime constrói consultas SQL a partir de CQN de forma agnóstica ao banco.
- Ideal para a maioria dos cenários, garantindo portabilidade.

## A Abordagem Nativa

### Quando usar?

Para cenários que exigem performance extrema ou lógica complexa que é mais bem implementada diretamente no banco de dados.

### Como funciona?

```
...
// srv/service.cds
@cds.persistence.exists
entity MyHanaView as projection on
  VDM.MyCalcView;
...
```

1. **Definir um Proxy CDS:** Crie uma entidade de serviço CDS que atua como um proxy para um artefato HANA nativo (ex: uma Calculation View).
2. **Implementar o Handler:** No handler do serviço, em vez de deixar o CAP gerar o SQL, você chama o artefato nativo.

```
// srv/service.js
srv.on('READ', 'MyHanaView', async (req) => {
  // Lógica para chamar a view nativa do HANA
  // ...
});
```

Combina a facilidade de modelagem do CAP com o poder bruto do processamento in-memory do HANA.

# Fase 3: O Reforço - Dados Localizados para um Mundo Global

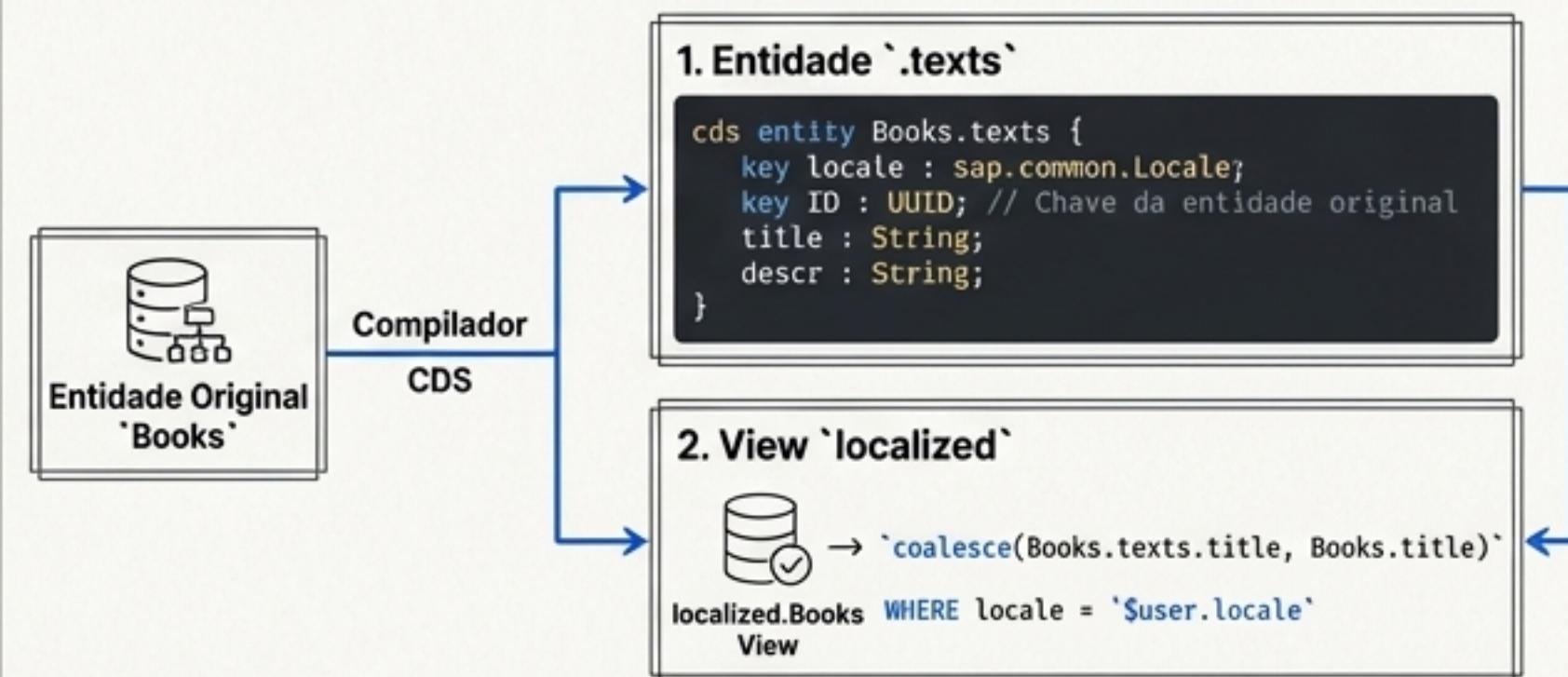
**O Desafio:** Aplicações enterprise precisam exibir dados no idioma preferido do usuário. Gerenciar traduções manualmente é complexo e propenso a erros.

**A Solução CAP:** O modificador `localized`.

## A Simplicidade no Modelo

```
// db/schema.cds
entity Books : cuid, managed {
    title : localized String;
    descr : localized String;
    ...
}
```

## A Mágica por Trás das Cenas



**Resultado:** O desenvolvedor simplesmente consulta `localized.Books` e o CAP resolve a tradução correta automaticamente com base no contexto do usuário.

# Reforçando a Performance: Modelagem Otimizada

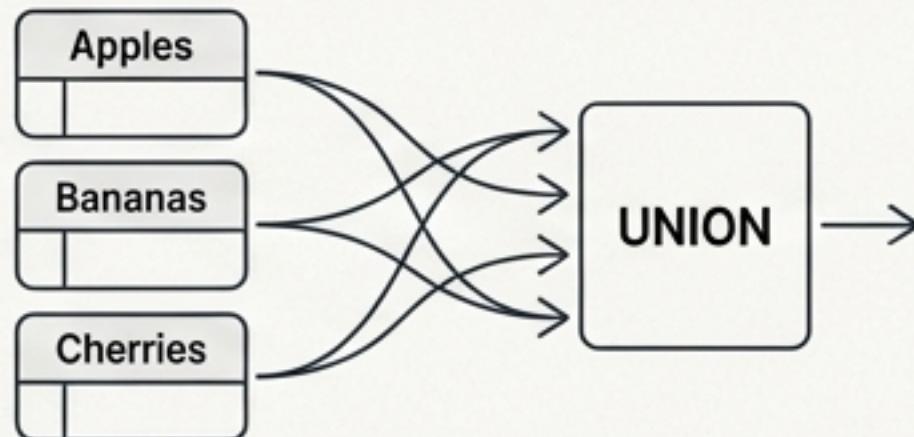
Um bom modelo de dados não é apenas correto, mas também performático. O CAP ajuda, mas **boas práticas de modelagem** são essenciais.



**AVISO:** `UNION`'s em views vêm com uma **penalidade de performance e modelagem complexa**.

**Cenário: Modelando Polimorfismo (Ex: Diferentes tipos de frutas)**

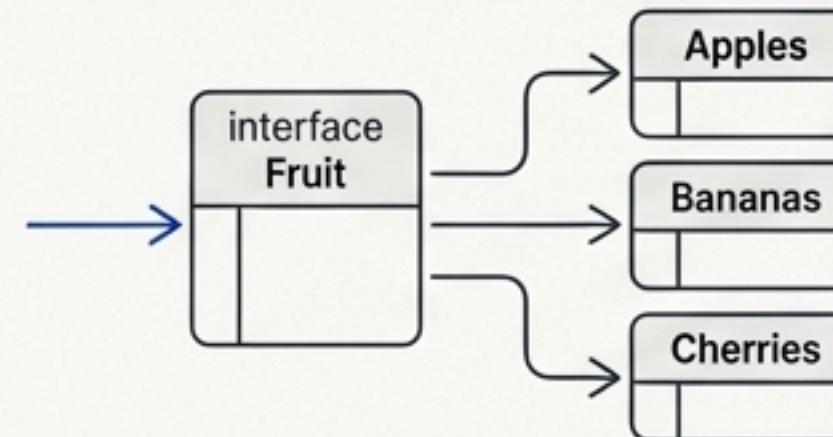
**X** Abordagem RUIM (Evite)



```
1 // Exemplo de código com UNION
2 view FruitsByVendor as
3   select from Apples UNION
4   select from Bananas UNION
5   select from Cherries
6   { ID, description, vendor }
7 where vendor.description = 'TopFruitCompany';
```

cds

**✓** Abordagem BOA (Prefira)



```
1 // Exemplo de código com associação
2 entity Fruit : cuid, managed {
3   type: String enum { apple; banana; cherry };
4   ...
5 }
6 entity Apples : Fruit { ... }
7 // A consulta é feita diretamente em 'Fruit'
```

cds

Dificulta a otimização pelo banco de dados, especialmente com filtros e ordenação.

Modelo mais limpo, código mais simples e consultas muito mais performáticas, pois o banco de dados pode usar índices de forma eficiente.

# Reforçando a Segurança: Proteção por Design

O CAP é “**Seguro por Padrão**”. **Funcionalidades** de desenvolvimento (como a página de índice e usuários mock) são desativadas por padrão no perfil de produção.

## Pilares da Segurança em CAP



### 1. Comunicação Segura

Na SAP BTP, toda a comunicação (entrada e saída) é criptografada via HTTPS/TLS por padrão.

“Aplicações CAP não precisam lidar com TLS, criptografia de comunicação ou certificados.”



### 2. Autenticação Robusta

O CAP integra-se nativamente com o serviço de identidade da plataforma (XSUAA). “Com a autenticação configurada, todos os endpoints do CAP são autenticados por padrão.”

! Aplicações CAP precisam garantir que um método de autenticação apropriado esteja configurado.



### 3. Autorização Granular

O modelo de autorização do CAP é derivado diretamente do modelo CDS usando anotações `@requires` e `@restrict`.

! Garanta que papéis técnicos como `cds.Subscriber` ou `mtcallback` NUNCA sejam incluídos em papéis de negócio.



### 4. Proteção contra Input Malicioso

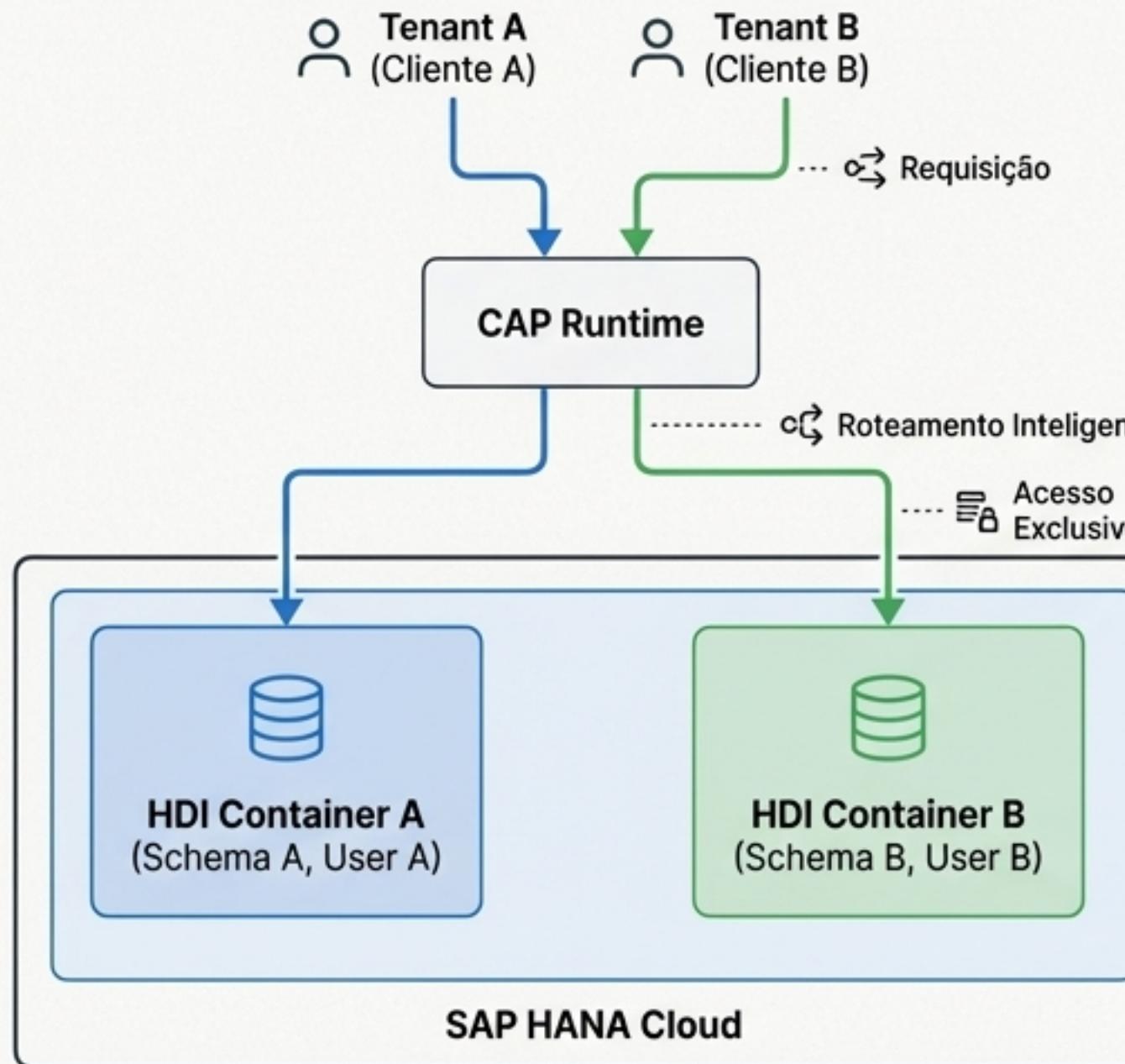
O motor de queries do CAP é imune a SQL Injection para valores de parâmetros.

O Application Router fornece proteção contra CSRF (Cross-Site Request Forgery) por padrão.

# Segurança em Escala: Isolamento de Dados em Multi-Tenancy

Em uma aplicação SaaS, os dados de diferentes clientes (tenants) devem ser estritamente isolados, mesmo compartilhando a mesma infraestrutura.

## A Solução CAP



### Isolamento de Dados Persistentes

O CAP provisiona automaticamente um HDI Container dedicado para cada tenant no SAP HANA Cloud. Cada container possui seu próprio esquema de banco de dados e um usuário técnico exclusivo. O runtime do CAP garante que cada requisição execute na conexão de banco de dados correta.

### Isolamento de Dados Transientes (em memória)

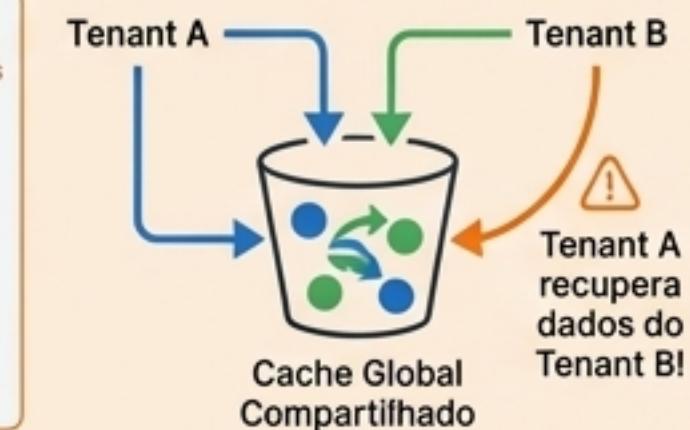
O runtime do CAP (Java e Node.js) foi projetado para ser tenant-aware. Caches internos são segregados por tenant. O contexto da requisição (incluindo o tenant) é propagado através da variável cds.context.

### Armadilha Comum (O que NÃO fazer)

```
// srv/cat-service.js
// RUIM: esta variável vaza dados entre tenants e requisições concorrentes
let books

module.exports = srv => {
  srv.on('READ', 'Books', async function(req, next) {
    if (books) return books // Retorna cache global, incorreto!
    return books = await next()
  })
}
```

**AVISO:** Certifique-se de que o código customizado não quebre o isolamento de dados do tenant ou vaze dados entre requisições concorrentes.



# Fase 4: O Ecossistema - Consumindo Serviços Remotos

Aplicações modernas raramente vivem isoladas. Elas precisam consumir dados de outros sistemas, como um S/4HANA ou outro microsserviço.

## O Processo em 3 Etapas

### 1. Importar a Definição da API



Use o comando `cds import` para converter uma especificação de serviço externo (EDMX) do SAP Business Accelerator Hub, por exemplo) em um arquivo CDS. Isso cria uma representação local do serviço remoto.

### 2. Mocking Local para Desenvolvimento

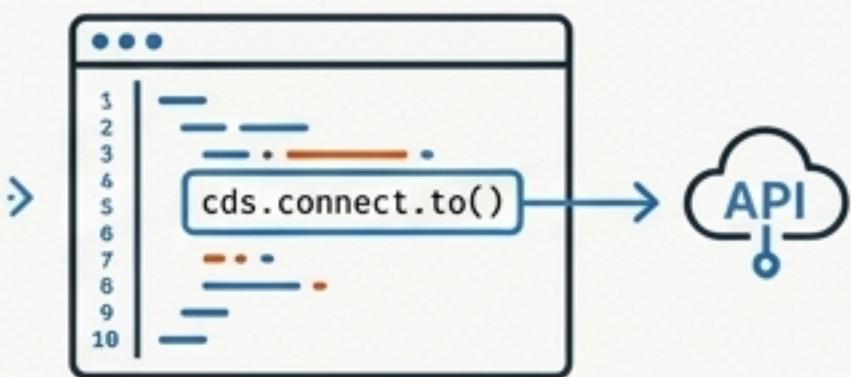


Adicione um arquivo .csv com dados de amostra. Ao executar `cds watch`, o CAP automaticamente cria um servidor mock para o serviço externo.

O desenvolvimento pode prosseguir de forma totalmente offline, sem dependência do sistema remoto.

**O CAP abstrai a complexidade da comunicação remota, permitindo que os desenvolvedores usem um modelo de programação unificado.**

### 3. Consultar o Serviço Remoto



No código, conecte-se ao serviço usando `cds.connect.to()` e use a mesma API de query (CQN) que você usaria para o banco de dados local.

```
// Conecta ao serviço definido no package.json
const bupa = await cds.connect.to('API_BUSINESS_PARTNER');

// Usa CQN para consultar o serviço remoto
const { A_BusinessPartner } = bupa.entities;
const result = await bupa.run(SELECT(A_BusinessPartner).limit(100));
```

# Fase 5: O Selo de Qualidade - Testando com `cds.test`

Garantir a qualidade é a etapa final antes da entrega. A biblioteca `@cap-js/cds-test` fornece utilitários para escrever testes eficazes para aplicações CAP Node.js.

## Configuração Simples

Em um arquivo de teste (`.test.js`), inicie o servidor CAP com uma única linha:

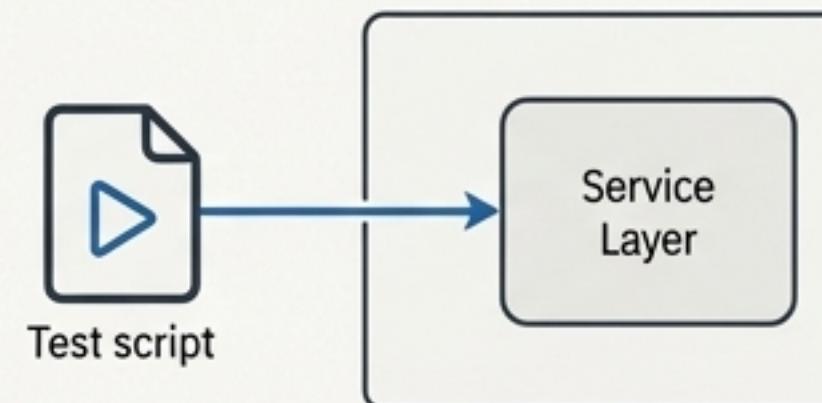
```
const cds = require('@sap/cds')

describe('Meu conjunto de testes', ()=>{
  const test = cds.test(__dirname+'/.')
  // ... seus testes aqui
})
```

**\*O que isso faz?\*** Inicia o servidor em um hook `beforeAll()` e o desliga de forma controlada em um `afterAll()`, gerenciando todo o ciclo de vida do teste.

## Dois Modos de Teste

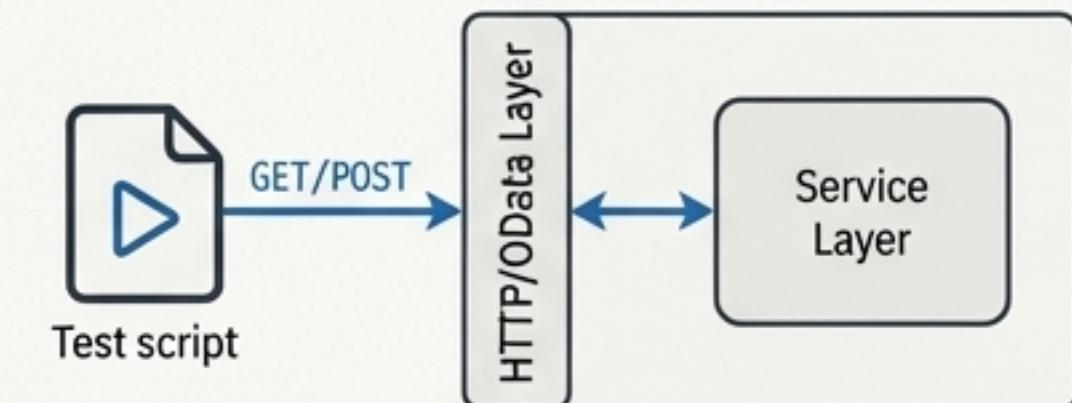
### 1. Testando APIs Programáticas (Service API)



Acesse os serviços diretamente via `cds.connect.to` e execute operações, ideal para testar a lógica de negócio nos handlers.

```
it('Testa a API programática', async () => {
  const AdminService = await cds.connect.to('AdminService');
  const { Authors } = AdminService.entities;
  const authors = await AdminService.run(SELECT.from(Authors));
  // ... asserções com expect...
});
```

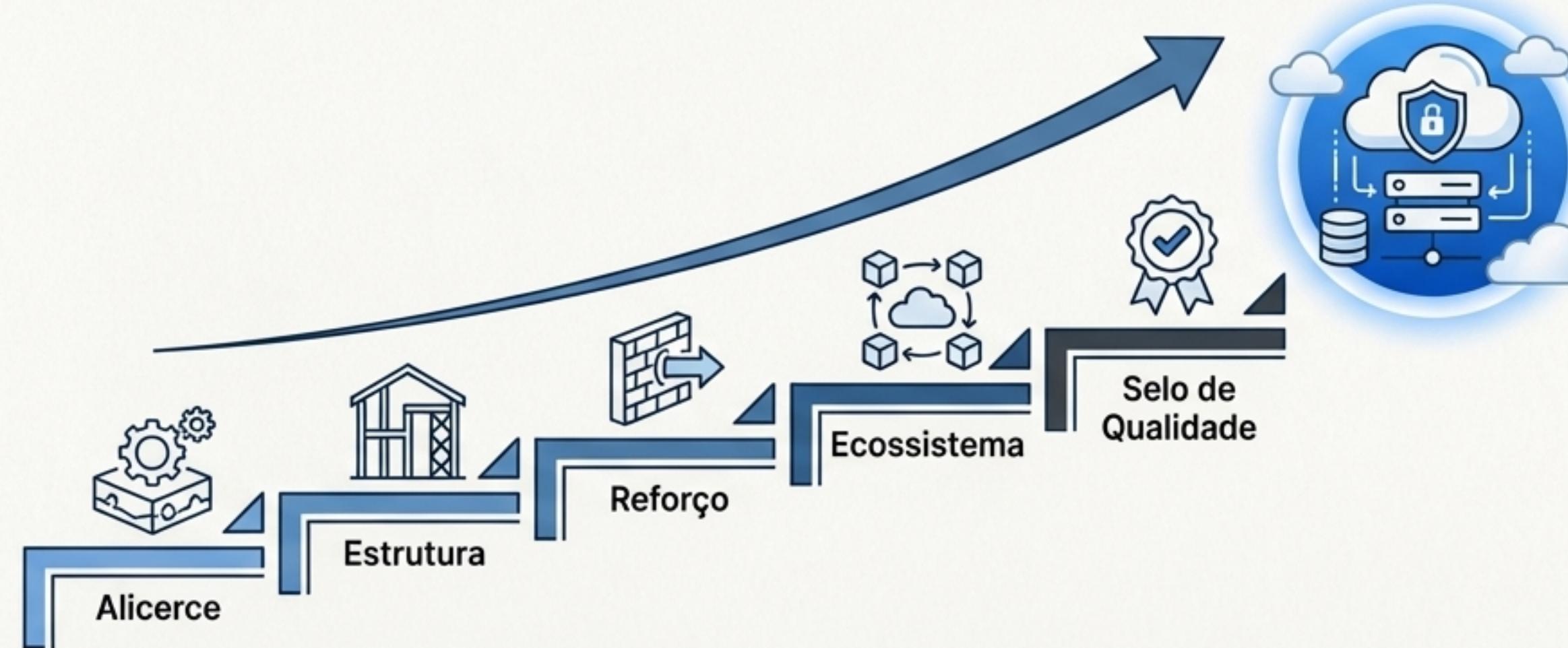
### 2. Testando APIs HTTP (OData)



Use os helpers `GET`, `POST`, `PATCH` para simular requisições HTTP reais, testando o comportamento de ponta a ponta.

```
it('Testa a API HTTP', async () => {
  const { GET } = test;
  const { data } = await GET('/browse/Books');
  expect(data.value).to.not.be.empty;
});
```

# A Jornada Completa: Do Modelo à Aplicação Enterprise-Ready



**O CAP não é apenas um conjunto de ferramentas, mas um framework coeso que guia o desenvolvedor através de todo o ciclo de vida da aplicação.**

## Foco no Negócio

Modele a intenção de negócio com CDS, e o framework cuida da complexidade da implementação.

## Flexibilidade de Persistência

Escolha o banco de dados ideal para cada ambiente (SQLite, PostgreSQL, SAP HANA) sem alterar seu modelo de domínio.

## Enterprise-Ready por Padrão

Funcionalidades críticas como segurança, multi-tenancy e internacionalização são integradas, não adicionadas a posteriori.

## Desenvolvimento Acelerado

O out-of-the-box para serviços OData, mocking de serviços e um conjunto de testes integrado reduzem o código boilerplate e aumentam a produtividade.

**Com o CAP, você constrói aplicações que não apenas funcionam, mas são seguras, escaláveis, performáticas e prontas para os desafios do mundo corporativo.**