



# Conectando-se a Serviços no CAP Node.js

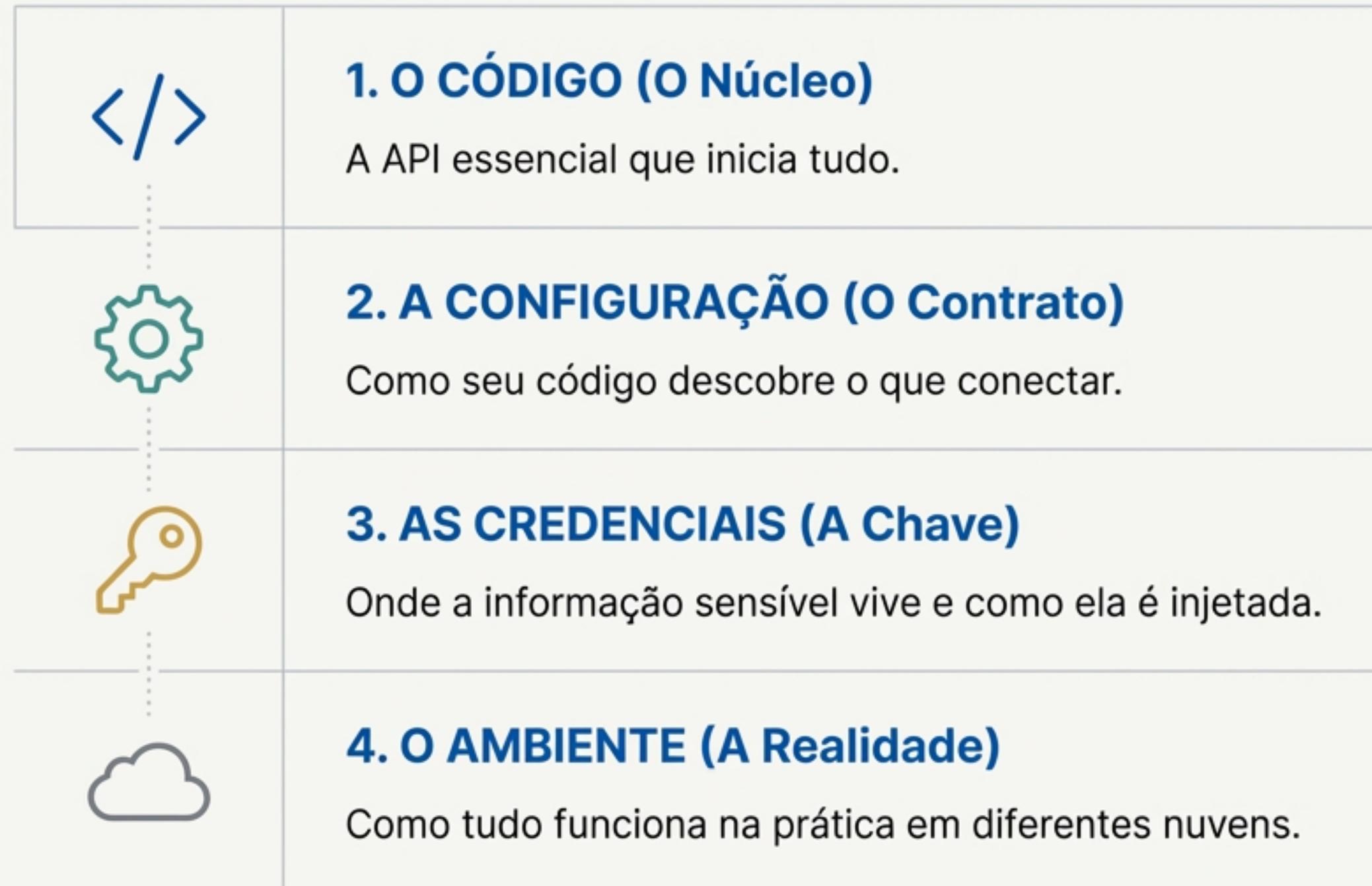
Um Guia Visual: Do Código à Nuvem



Para desenvolvedores Node.js com o SAP Cloud Application Programming Model.

# A Jornada de Uma Conexão

Vamos desvendar a conexão de serviços camada por camada, do conceito mais simples ao mais complexo.



# O Ponto de Partida: `cds.connect.to()`

**Tudo começa aqui. Para consumir qualquer serviço, seja local ou remoto, você usa `cds.connect.to()`.**

```
// Dentro da sua lógica de serviço...
const ReviewsService = await cds.connect.to('ReviewsService');

// Agora você pode usar o objeto 'Service' como um proxy.
const reviews = await ReviewsService.read('Reviews');
```

## O que retorna?

Uma `Promise` que resolve para uma instância de `Service`, que atua como um proxy para a API do serviço.

## Otimização

O CAP armazena em cache as instâncias de serviço em `cds.services`. Chamadas subsequentes com o mesmo nome retornam a mesma instância, incluindo serviços locais definidos com `cds.serve`.

# A Mágica por Trás da Conexão: cds.requires

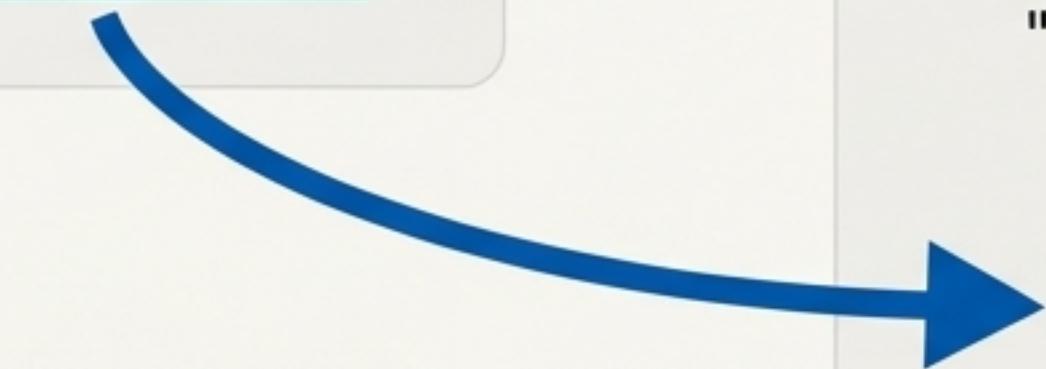
Como `cds.connect.to('ReviewsService')`  
sabe aonde se conectar?

O Código

```
await cds.connect.to('ReviewsService');
```

A Configuração (package.json)

```
{  
  "cds": {  
    "requires": {  
      "db": {  
        "...": "..."  
      },  
      "ReviewsService": {  
        "kind": "odata",  
        "model": "@capire/reviews"  
      }  
    }  
}
```



O nome passado para `connect.to()` é a chave usada para procurar a configuração do serviço em `cds.requires`.

# Anatomia de um Serviço Requerido

## "kind"

O tipo de serviço. Define um conjunto de configurações padrão.

Exemplos comuns: `odata`, `sqlite`, `hana`. As configurações herdam da cadeia de `kind`.

## "service"

(Opcional) Usado quando o nome do serviço dentro do arquivo de modelo (`model`) é diferente da chave da configuração (`BusinessPartnerService`).

```
"BusinessPartnerService": {  
    "kind": "odata",  
    "model": "srv/external/API_BUSINESS_PARTNER.csn",  
    "impl": "./lib/custom-handler.js",  
    "service": "API_BUSINESS_PARTNER.service"  
}
```

## "model"

O caminho para a definição da API do serviço (CSN/CDS). Essencial para serviços remotos, permitindo reflexão e recursos genéricos. O runtime carrega este modelo no `cds.model` efetivo.

## "impl"

(Opcional) O caminho para um módulo Node.js com uma implementação de serviço personalizada. Prefixe com `./` para caminhos relativos à raiz do projeto.

# A Peça Faltante: Credenciais e Segredos



**ERRADO**

```
// NUNCA FAÇA ISSO!
"db": {
  "kind": "hana",
  "credentials": {
    "user": "myuser",
    "password": "mysecretpassword"
  }
}
```

**CERTO**

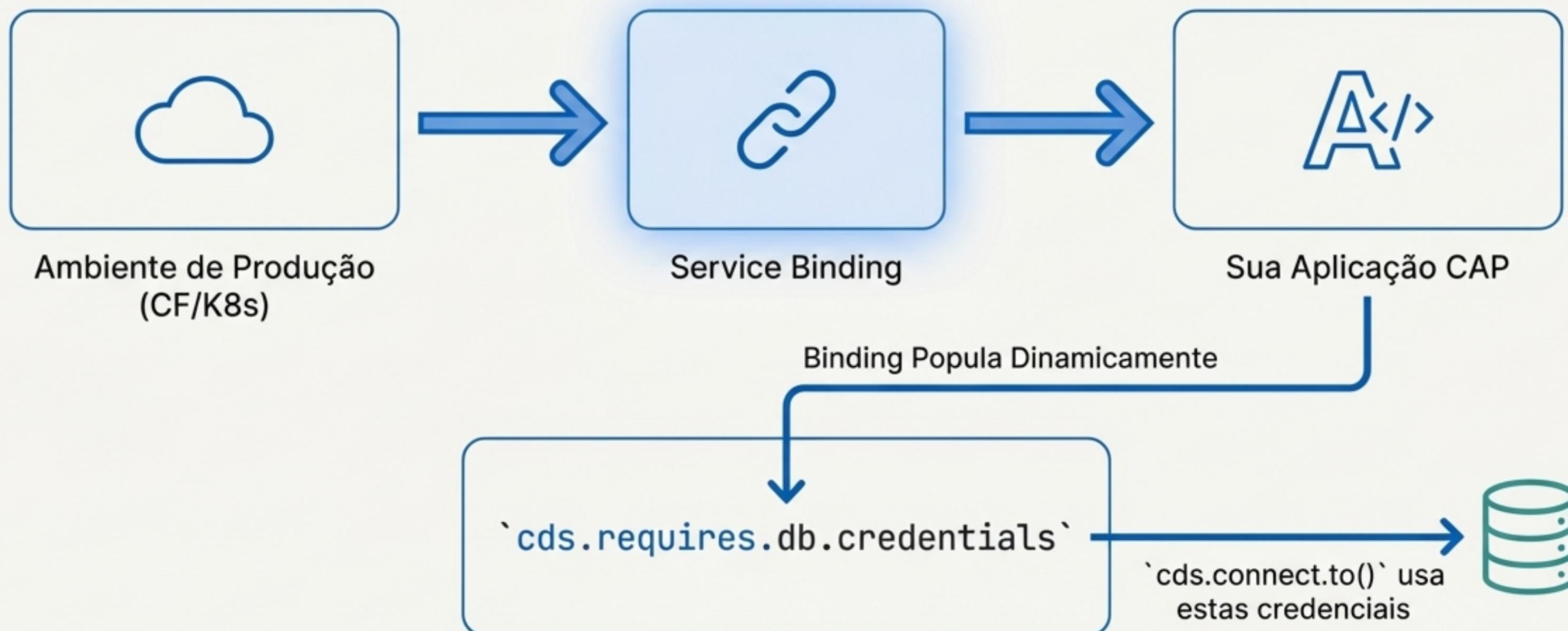
```
// CORRETO
"db": {
  "kind": "hana",
  "credentials": {
    /* Injetado pelo ambiente em runtime */
  }
}
```

Regra de Ouro: **Nunca comite segredos, senhas ou chaves de API no seu repositório.**

A propriedade `credentials` é o local designado para informações sensíveis.  
O runtime do CAP preenche (e sobrescreve) esta seção a partir do ambiente do processo na inicialização.

# Service Bindings: A Ponte Para o Mundo Real

Bindings são o mecanismo pelo qual um ambiente de runtime (como Cloud Foundry ou Kubernetes) "injeta" as credenciais necessárias na configuração da sua aplicação na inicialização.



# O Mecanismo Básico e Desenvolvimento Local

Para testes locais, você pode simular o comportamento de bindings usando arquivos ` `.env` ou variáveis de ambiente.

## Arquivo ` `.env` (NÃO COMITAR!)

```
# Arquivo .env  
  
cds.requires.ReviewsService.credentials.url = .....  
    http://localhost:4005/reviews  
  
cds.requires.db.credentials.url = db.sqlite
```

## Como o CAP “vê” em ` `cds.env`

```
// Em cds.env.requires...  
"ReviewsService": {  
    "credentials": {  
        "url": "http://localhost:4005/reviews"  
    }  
},  
"db": {  
    "credentials": {  
        "url": "db.sqlite"  
    }  
}
```

Você também pode passar credenciais diretamente via linha de comando para testes ad-hoc, por exemplo:  
`export cds_requires_db_credentials_url=db.sqlite`

# Cenário de Produção 1: Cloud Foundry

No Cloud Foundry, os bindings são fornecidos através da variável de ambiente `VCAP\_SERVICES`. O CAP faz a **correspondência automática**.

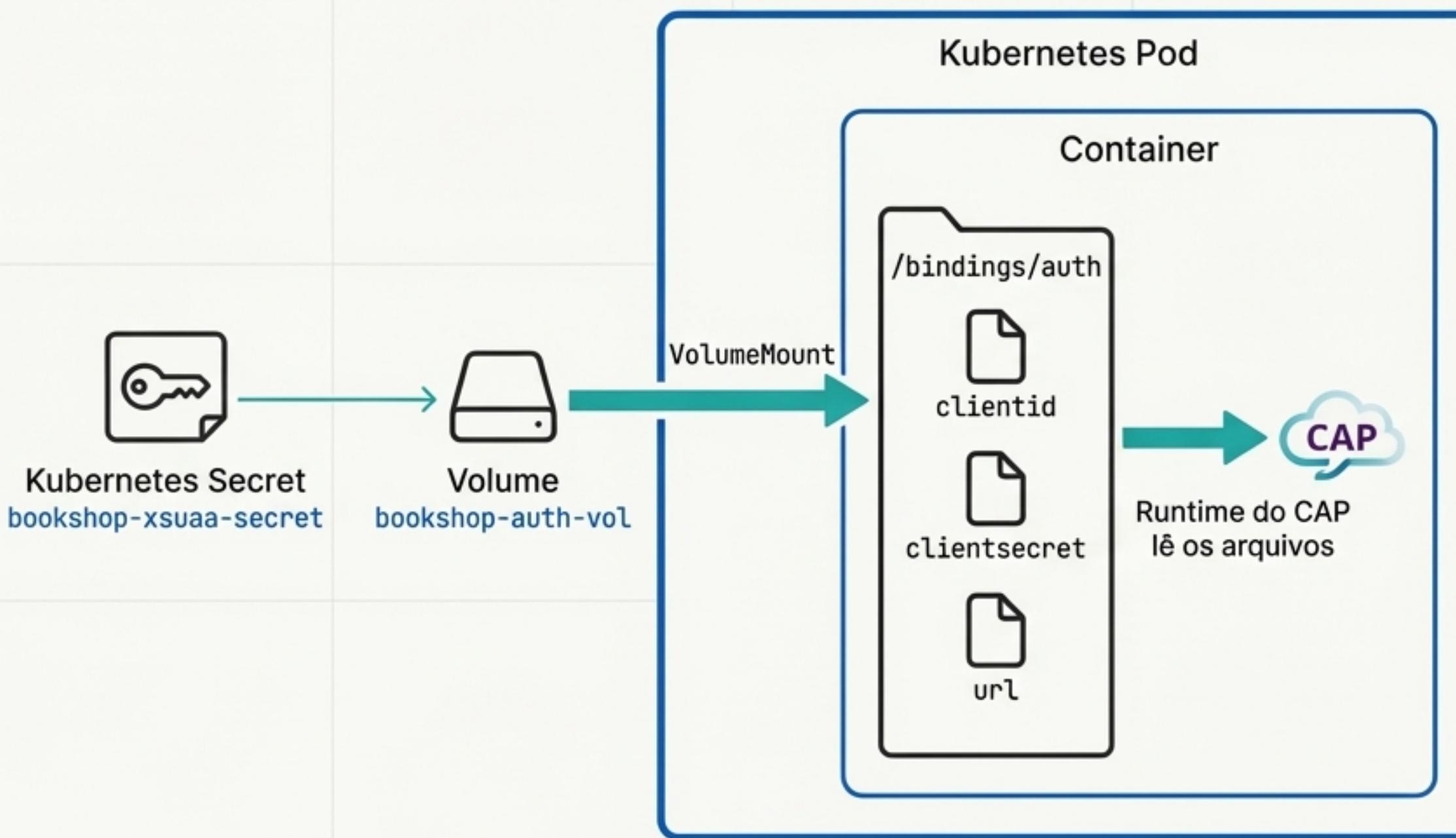
## Ordem de Precedência

- 1. Configuração `vcap` explícita na sua definição de serviço (`cds.requires.db.vcap.name`).
- 2. Nome do serviço ('db') vs. `binding\_name` do serviço no VCAP.
- 3. Nome do serviço ('db') vs. `tags` do serviço.
- 4. `kind` do serviço ('hana') vs. `tags` do serviço.
- 5. `kind` do serviço ('hana') vs. `label` do serviço.

package.json (cds.requires)	VCAP_SERVICES
<pre>{   "db": {     "kind": "hana"   } }</pre>	<pre>{   "hana": [{     "label": "hana", ... }] }</pre>
Match pelo `kind` e `label`	
<pre>{ "db": {   "vcap": {     "name": "myDb"   } }</pre>	<pre>{   "hana": [{     "name": "myDb", ... }] }</pre>
Match pelo `vcap.name`	

# Cenário de Produção 2: Kubernetes/Kyma (Método 1)

O CAP suporta a especificação `servicebinding.io`, que projeta segredos como arquivos em um diretório.



## Passo 1: Definir a Raiz dos Bindings

```
env:  
- name: SERVICE_BINDING_ROOT  
  value: /bindings
```

## Passo 2: Montar os Segredos

```
volumeMounts:  
- name: bookshop-auth-vol  
  mountPath: "/bindings/auth"  
  readOnly: true  
volumes:  
- name: bookshop-auth-vol  
  secret:  
    secretName: bookshop-xsuaa-secret
```

# Kubernetes/Kyma (Métodos Alternativos)

## Opção A: Injeção Direta de Variáveis de Ambiente

Adiciona todas as chaves de um `Secret` como variáveis de ambiente no Pod, usando um prefixo que mapeia para a configuração do CAP.

**Atenção:** Use `\_` como delimitador no prefixo. O nome do serviço não deve conter `\_`.

deployment.yaml

```
envFrom:  
- prefix: cds_requires_db_credentials_  
  secretRef:  
    name: app-db
```

## Opção B: Configuração via Sistema de Arquivos (`CDS\_CONFIG`)

Define um diretório raiz com `CDS\_CONFIG`. O CAP lê arquivos de configuração de uma estrutura de diretórios espelhada.

```
<CDS_CONFIG>/requires/<seu_serviço>/credentials/
```

A variável `user` do `Secret` seria montada como um arquivo chamado `user` dentro do diretório de credenciais.

# Estratégias de Binding Adicionais

## Testes Híbridos

**Ferramenta (Secondary Font):**

Arquivo `cdsrc-private.json`

**Uso (Secondary Font):** Para fornecer credenciais localmente ao testar contra serviços na nuvem.



AVISO: Adicione este arquivo ao `gitignore`. **Nunca comite!**

## Emulando `VCAP\_SERVICES`

**Flag (Secondary Font):**

```
`features.emulate_vcap_service  
s: true`
```

**Uso (Secondary Font):** Em ambientes que não são CF (como K8s), para bibliotecas que esperam `VCAP\_SERVICES`. É um recurso de compatibilidade.

Como funciona (Secondary Font, small): Gera a variável `VCAP\_SERVICES` a partir dos seus serviços configurados em `cds.requires` que possuem credenciais e uma propriedade `vcap.label`.

## Comandos Úteis

```
cds env get  
requires.<servicename>
```

(Annotation: Para ver a configuração padrão)

```
cds env get VCAP_SERVICES  
--process-env
```

(Annotation: Para ver a `VCAP\_SERVICES` gerada)

# O Mapa Completo da Conexão

## Layer 1: O Código



## Layer 2: A Configuração

### A Configuração

```
cds.requires(  
{  
  "Srv": {  
    "kind": "hana"  
  }  
})
```

The diagram shows configuration code:

```
cds.requires(  
{  
  "Srv": {  
    "kind": "hana"  
  }  
})
```

Annotations point to the following parts of the code:

- `cds.requires` points to a light blue rounded rectangle.
- `{` and `}` point to light green rounded rectangles.
- `"Srv": {` points to a light blue rounded rectangle.
- `"kind": "hana"` points to a light green rounded rectangle.

## Layer 3: A Injeção de Credenciais

### Cloud Foundry



Cloud Foundry

```
VCAP_SERVICES: {  
  "hana": [...],  
  "credentials": {...}  
}
```

### Layer 3: A Injeção de Credenciais

Na inicialização, o ambiente fornece um binding

### Kubernetes



Kubernetes

```
SERVICE_BINDING_ROOT  
/bindings  
  user  
  password  
envFrom  
prefixinggments
```

### Local



cds run

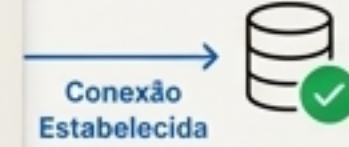


```
VCAP_SERVIEES[...]  
COS_BEGUTRES_Srv_CREDENTIALS_user[...]
```

## Layer 4: A Conexão

### A Credentials

```
cds.requires.Srv.credentials  
{  
  "user": "...",  
  "passxerd": "...",  
  "urt": "..."  
}
```



Conexão Estabelecida



# Pontos-Chave e Boas Práticas

## 1 Ponto de Entrada Unificado

Use `cds.connect.to()` para todas as conexões de serviço. É a API consistente e central.

## 2 Declare Suas Dependências

Defina explicitamente cada serviço consumido em `cds.requires`. Isso torna a arquitetura da sua aplicação clara e gerenciável.

## 3 Separe Configuração de Segredos

A propriedade `credentials` nunca deve ser versionada. Confie nos mecanismos de binding do seu ambiente.

## 4 Entenda Seu Ambiente

O mecanismo de binding é a principal diferença entre os ambientes de produção. Saiba se você está lidando com `VCAP\_SERVICES` (CF) ou montagem de volumes/variáveis de ambiente (K8s).

# Para Aprofundar e Explorar

Continue sua jornada com a documentação oficial e ferramentas de linha de comando.

## Documentação Oficial

- 🔗 [Connecting to Required Services:](#)  
[link\_para\_documento\_oficial]
- 🔗 [Service Bindings in Detail:](#)  
[link\_para\_documento\_de\_bindings]
- 🔗 [Configuration using `cds.env`:](#)  
[link\_para\_documento\_de\_env]

## Ferramentas para o Dia a Dia

Para inspecionar sua configuração final e resolver problemas de binding, use os comandos `cds env`:

```
cds env get requires
```

```
cds env get VCAP_SERVICES --process-env
```

```
cds env get requires.db
```

Perguntas? Obrigado!