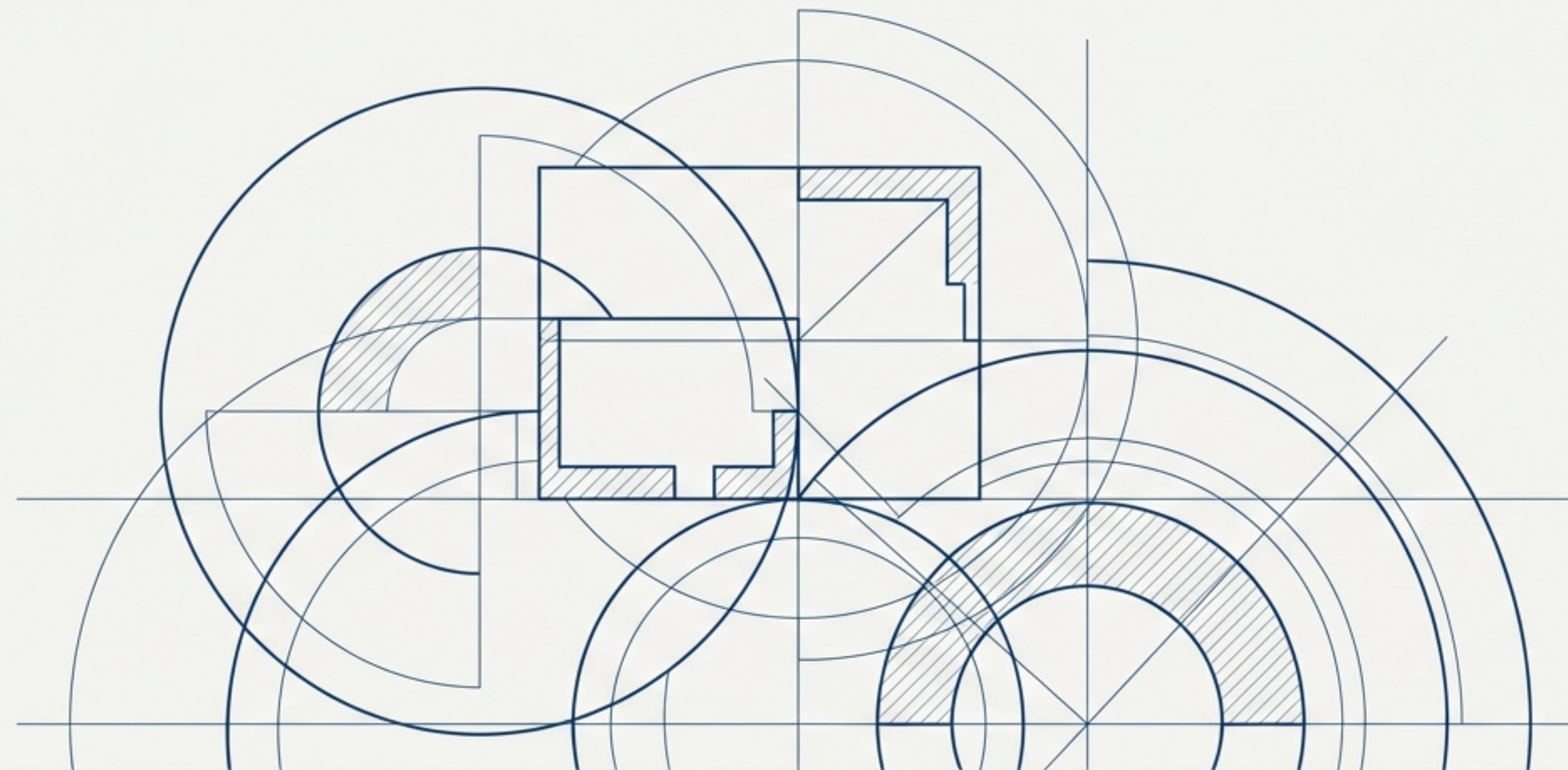


Desvendando o `cds.env`

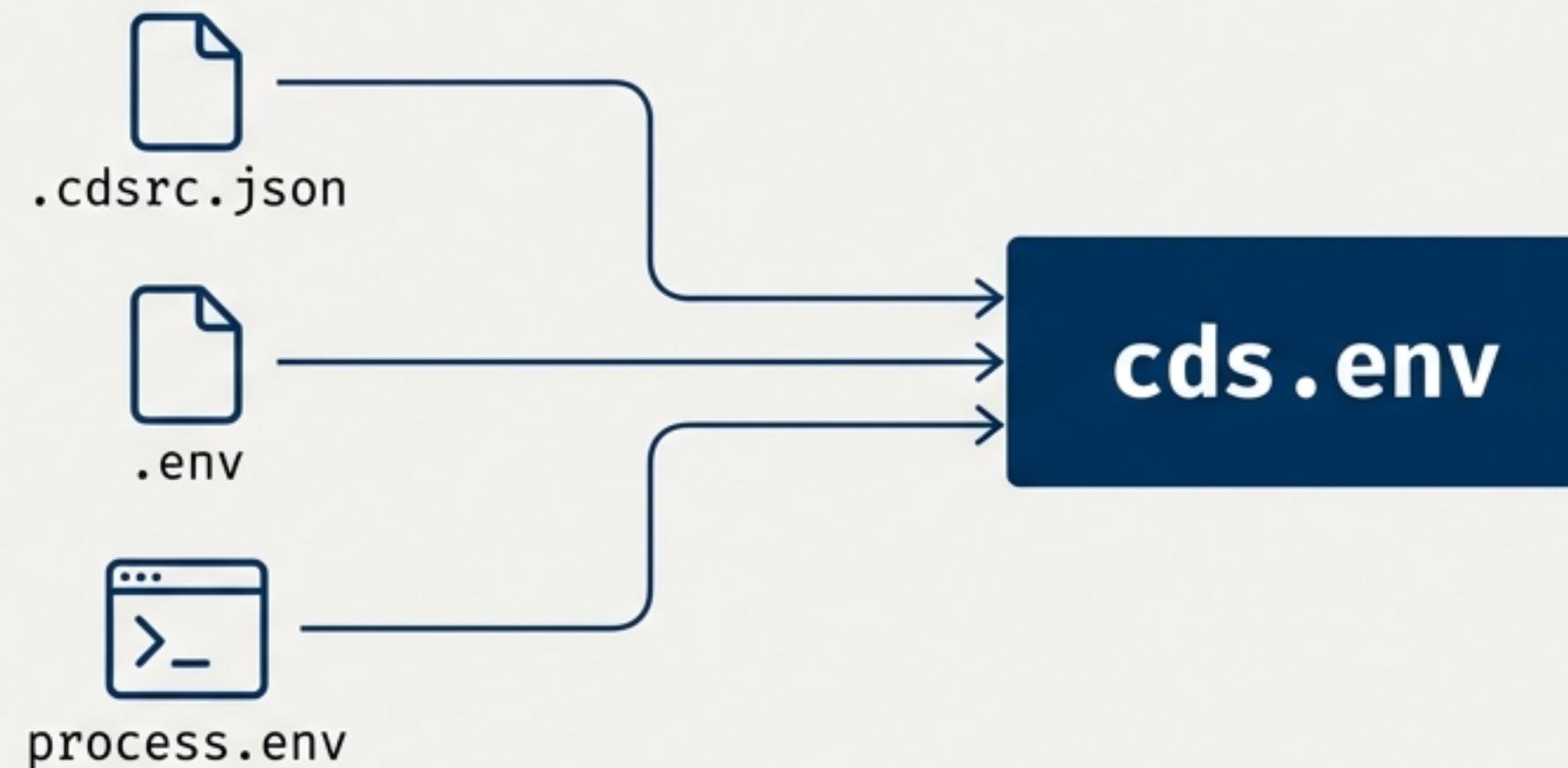
Um Guia Arquitetural para a Configuração de Aplicações SAP CAP



O Destino Final: O Objeto `cds.env`

No coração de toda aplicação CAP Node.js está o objeto `cds.env`. Ele representa a configuração efetiva e unificada, consolidada a partir de múltiplas fontes. Compreender como ele é construído é a chave para gerenciar sua aplicação em qualquer ambiente.

`cds.env` é o seu "one-stop shop" para todas as configurações em tempo de execução.



Inspecionando a Configuração Ativa com `cds env`

A primeira ferramenta em seu arsenal. Use o comando `cds env` na raiz do seu projeto para ver a configuração final que sua aplicação está usando.

Comando	Descrição e Saída de Exemplo
cds env ls (ou `cds env`)	#> lista todas as configurações em formato de propriedades
cds env ls requires.db	#> lista apenas as configurações para 'requires.db'
cds env get	#> imprime todas as configurações em formato JSON
cds env get requires.db	#> imprime a configuração de 'requires.db' em JSON

Exemplo: `cds env get requires.db`

```
{  
  "impl": "@cap-js/sqlite",  
  "credentials": {  
    "url": ":memory:"  
  },  
  "pool": {  
    "max": 1  
  },  
  "kind": "sqlite"  
}
```

Acesso Programático à Configuração

Além do CLI, você pode acessar o objeto `cds.env` diretamente para scripting e depuração.

Usando `cds repl`

```
$ cds -r  
> cds.env.requires.db
```

```
{  
  impl: '@cap-js/sqlite',  
  credentials: { url: ':memory:' },  
  kind: 'sqlite'  
}
```

Usando `cds eval`

```
> cds -e .env.requires.db
```

```
{  
  impl: '@cap-js/sqlite',  
  credentials: { url: ':memory:' },  
  pool: { max: 1 },  
  kind: 'sqlite'  
}
```

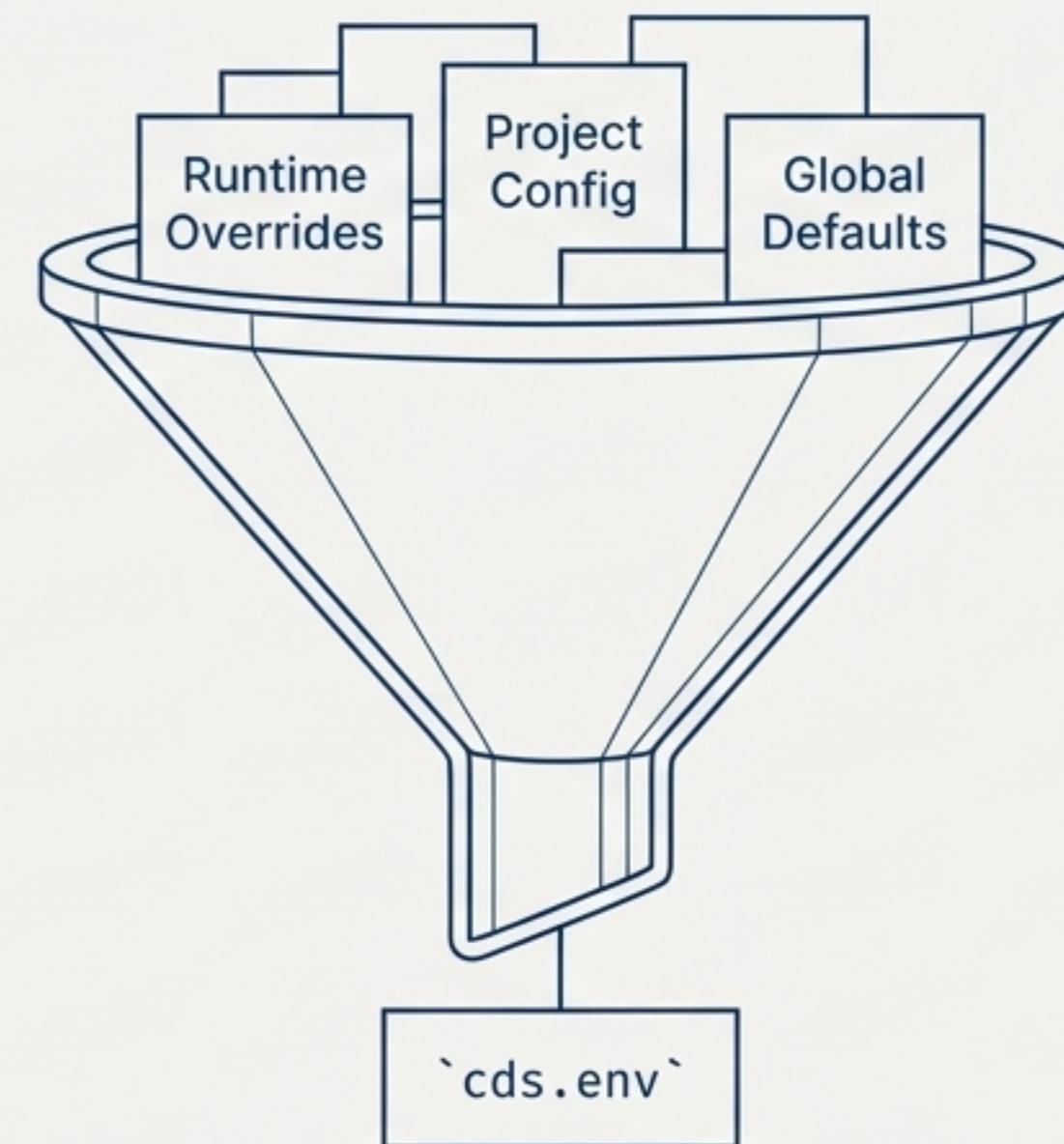
Dentro do Código Node.js

```
const cds = require('@sap/cds')  
console.log(cds.env.requires.sql)  
console.log(cds.env.requires.sql)
```

Imprime a mesma saída que `cds env get requires.sql`.

De Onde Vêm as Configurações? O Funil de Precedência

O objeto `cds.env` não surge do nada. Ele émeticamente construído a partir de 12 fontes, mescladas em uma ordem de precedência estrita. Fontes de ordem superior sempre sobrescrevem as de ordem inferior.



Fontes de maior ordem (topo) têm precedência sobre as de menor ordem (base).

O Blueprint Completo: As 12 Fontes de Configuração

A mesclagem segue esta ordem precisa, do número 1 (a mais baixa precedência) ao 12 (a mais alta).

Ordem	Fonte	Observações
1	`Defaults internos do @sap/cds`	
2	`~/.cdsrc.yaml (.json,.js)`	Padrões específicos do usuário
3	`./.cdsrc.yaml (.json,.js)`	Configurações estáticas do projeto
4	`./package.json`	Seção `{"cds":{ ... }}`
5	`./.cdsrc-private.json`	Configuração de projeto específica do usuário
6	`./default-env.json`	*Obsoleto, use `cds bind`*
7	`./.env`	Variáveis de ambiente específicas do projeto
8	`./.<profile>.env`	Variáveis de ambiente por perfil (ex: .hybrid.env`)
9	`process.env.CDS_CONFIG`	Configurações de runtime via JSON ou arquivo
10	`process.env`	Variáveis de ambiente do shell ou cloud
11	`process.env.VCAP_SERVICES`	Bindings de serviço (Cloud Foundry)
12	`~/.cds-services.json`	Bindings de serviço para perfil `development`

`./` representa o diretório raiz do projeto. `~/` representa o diretório home do usuário.

Zona 1: A Fundação (Defaults Globais)

A configuração começa com uma base de padrões globais, aplicados a todos os seus projetos.

Defaults Internos (Fonte 1)

****Fonte**:** `@sap/cds`

****Propósito**:** Fornece configurações padrão para propriedades essenciais como `build`, `features`, `folders`, `i18n`, `odata`, e `requires`.

****Benefício**:** Garante que você possa acessar `cds.env.requires.sql` com segurança, sem verificar valores nulos.

Defaults do Usuário (Fonte 2)

****Fonte**:** `~/.cdsrc.json`

****Propósito**:** Permite definir suas próprias configurações padrão que serão aplicadas a *todos* os seus projetos CAP em sua máquina. Ideal para preferências pessoais.

****Exemplo**:** Definir um banco de dados padrão para todos os projetos locais.

Zona 2: O Blueprint do Projeto (Configuração Estática)

Essas configurações são parte essencial da topologia do seu projeto. Elas são versionadas no Git e usadas em todos os ambientes, incluindo produção.

TIP



Não adicione opções específicas de ambiente aqui. Use as opções de ambiente de processo dinâmico para isso.

Opção 1: Em `./package.json` (Fonte 4)

Configurações dentro de uma seção ` "cds\"`.

```
{  
  "cds": {  
    "requires": {  
      "db": {  
        "kind": "sqlite"  
      }  
    }  
  }  
}
```

Opção 2: Em `./cdsrc.json` (Fonte 3)

Uma alternativa mais limpa, especialmente para configurações complexas.

```
{  
  "requires": {  
    "db": "sql"  
  }  
}
```

Zona 3: Overrides Locais (Privados e de Ambiente)

Configurações para o seu ambiente de desenvolvimento local ou para dados sensíveis. **Estes arquivos não devem ser enviados para o controle de versão.**

Configurações Privadas (Fonte 5)

Arquivo: `./.cdsrc-private.json`

Uso: Um equivalente privado do `cdsrc.json` para seus testes locais.

Ambientes por Perfil (Fonte 8)

Arquivo: `./.<profile>.env` (ex: `./.hybrid.env`)

Uso: Fornece variáveis de ambiente que são ativadas apenas quando um perfil específico está em uso.

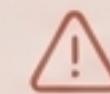
Variáveis de Ambiente (Fonte 7)

Arquivo: `./.env`

Uso: Definir variáveis de ambiente para o seu projeto localmente.

```
# Formato padrão  
cds_requires_db_kind = sql
```

```
# Formato com notação de ponto (válido apenas em  
arquivos .env)  
cds.requires.db.kind = sql
```



A notação de ponto (`.`) só pode ser usada em arquivos `.env`.

Zona 4: A Autoridade Final (Overrides de Runtime)

Estas são as configurações de maior precedência, definidas no momento da execução. Elas sobrescrevem todas as outras fontes.

Método Principal: `process.env.CDS_CONFIG` (Fonte 9)

Uma variável de ambiente extremamente flexível.

- **Modo 1:** String JSON

```
CDS_CONFIG='{"requires":{"db":{"kind":"sqlite\\\"}}}' cds serve
```

- **Modo 2:** Arquivo JSON

```
CDS_CONFIG=./my-cdsrc.json cds serve
```

- **Modo 3:** Diretório

```
CDS_CONFIG=/etc/secrets/cds cds serve
```

O CAP mapeia a estrutura do diretório e o conteúdo dos arquivos para o objeto `cds.env`. Perfeito para segredos montados como volumes (ex: Kubernetes).

Outros Métodos de Runtime

`process.env` (Fonte 10): Variáveis de ambiente individuais (ex: 'CDS_REQUIRES_DB_KIND=sql cds run').`

`'VCAP_SERVICES` / `~/.cds-services.json` (Fontes 11, 12): Para bindings de serviços na nuvem ou localmente.`

A Mesclagem em Ação: Um Exemplo de Precedência

Vamos ver como as configurações de diferentes fontes são combinadas para formar o resultado final.

``.cdsrc.json` (Ordem 3)`

```
{  
  "requires": {  
    "db": {  
      "model": "./db"  
    }  
  }  
}
```

``package.json` (Ordem 4)`

```
{  
  "cds": {  
    "requires": {  
      "db": {  
        "kind": "sqlite"  
      }  
    }  
  }  
}
```

``./.env` (Ordem 7)`

```
cds.requires.db.credentials.database = my.sqlite
```

Resultado Final: Configuração Efetiva em `cds.env`

```
{  
  "requires": {  
    "db": {  
      "kind": "sqlite",          // Do package.json  
      "model": "./db",          // Do .cdsrc.json  
      "credentials": {  
        "database": "my.sqlite" // Do .env  
      }  
    }  
  }  
}
```

Padrão Essencial: Gerenciando Ambientes com Perfis

Use perfis para definir configurações específicas para diferentes ambientes (ex: `development`, `production`) dentro dos seus arquivos de configuração estática.

Exemplo em `package.json`

```
"cds": {  
  "requires": {  
    "db": {  
      "[development)": { "kind": "sqlite" },  
      "[production)": { "kind": "hana" }  
    }  
  }  
}
```

Como um Perfil é Determinado (Ordem de Prioridade):

1. Argumento de linha de comando: --production
2. Argumento de linha de comando: --profile <nome>
3. Variável de ambiente: NODE_ENV
4. Variável de ambiente: CDS_ENV

Se nenhum perfil de produção for definido, o perfil `development` é ativado automaticamente.

Exemplo, `a custom profile command:
`CDS_ENV=my-custom-profile cds run`

Estendendo o Blueprint: Configurações Específicas da Aplicação

O sistema de configuração não é apenas para o framework CAP. Você pode usá-lo para gerenciar as configurações da sua própria aplicação de forma limpa e centralizada.

Passo 1: Defina sua Configuração (em `package.json`)

Adicione uma seção de nível superior com o nome da sua aplicação ou módulo.

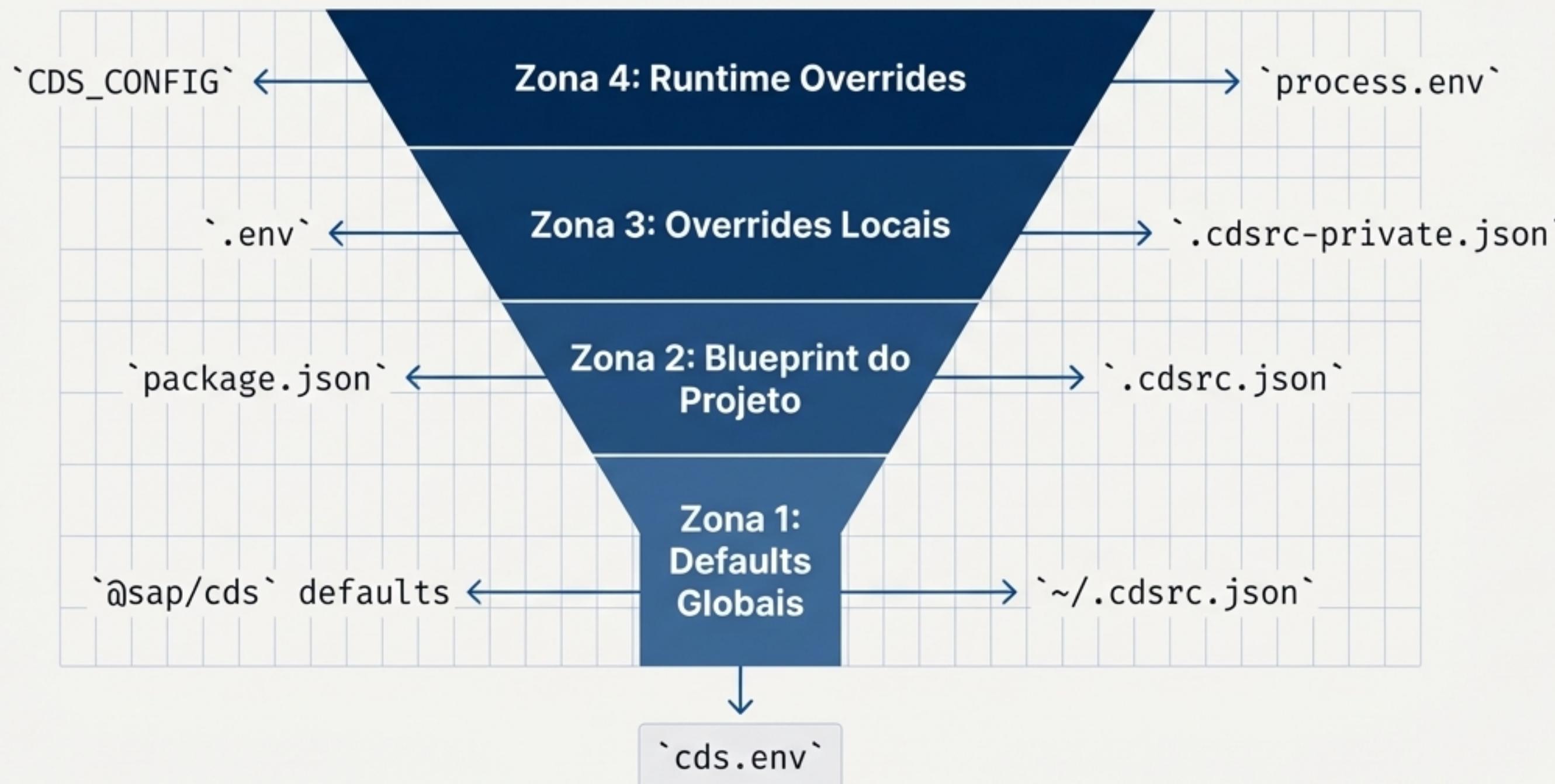
```
{  
  "cds": { ... },  
  "my-app": {  
    "myoption": "value",  
    "featureToggle": true  
  }  
}
```

Passo 2: Acesse sua Configuração no Código

Use a função `cds.env.for('<seu-nome>')` para obter um objeto de configuração isolado.

```
const { myoption, featureToggle } =  
  cds.env.for('my-app')  
  
if (featureToggle) {  
  // ... execute a lógica da feature  
}
```

O Funil de Configuração: Uma Visão Consolidada



A configuração é um processo de mesclagem previsível e em camadas.
Dominar essa ordem permite que você configure qualquer cenário com confiança.

Dominando sua Configuração

Lembre-se destes princípios para construir aplicações robustas e fáceis de manter:



Inspecione Sempre

Use `cds env` como sua primeira ferramenta para entender o estado atual.



Respeite a Hierarquia

Entenda a ordem das 12 fontes para prever o comportamento da configuração.



Use Perfis para Ambientes

Mantenha as configurações de `development` e `production` limpas e separadas.



Mantenha Segredos Seguros

Use `.env`, `.cdsrc-private`, ou variáveis de ambiente de runtime para credenciais.

Nunca os coloque no `package.json`.



Estenda o Sistema

Use `cds.env.for('my-app')` para suas próprias configurações.