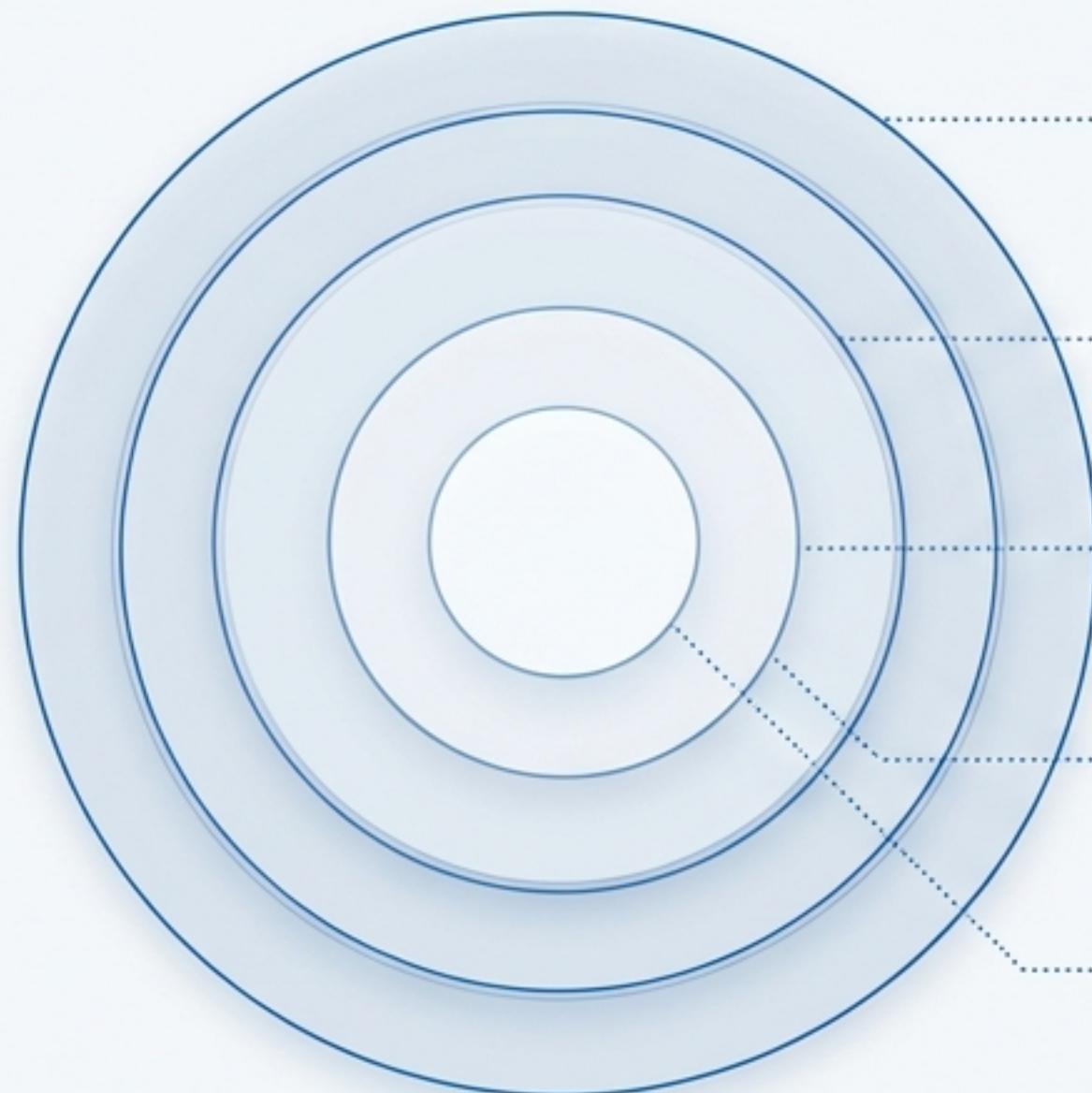


Dominando o Bootstrap de Servidores CAP Node.js

De `cds serve` ao Controle Total do Ciclo de Vida

A Jornada de Hoje: Desvendando o Processo de Bootstrap em 5 Camadas



- **1. O Ponto de Partida**
O comando `cds serve` que todos conhecemos.
- **2. Por Trás da Cortina**
O `server.js` padrão e sua coreografia.
- **3. Assumindo o Controle**
Criando seu próprio `server.js`.
- **4. Os Pontos de Encaixe**
Dominando os eventos do ciclo de vida.
- **5. Ajustes Finos**
Modificando o comportamento via configuração.

Camada 1: Tudo Começa com `cds serve`

O processo de um servidor CAP Node.js é geralmente iniciado pelo comando `cds serve`.

Os comandos `cds run` e `cds watch` são variantes de conveniência para o desenvolvimento, oferecendo funcionalidades como reinicialização automática.

Para Deploy

Quando o pacote de desenvolvimento `@sap/cds-dk` não está disponível (em ambientes de produção), use o binário `cds-serve` do pacote principal `@sap/cds`.

```
{  
  "name": "my-cap-project",  
  "scripts": {  
    "start": "cds-serve"  
  },  
  "dependencies": {  
    "@sap/cds": "^7"  
  }  
}
```

Camada 2: O que `cds serve` Realmente Faz?

- Ele executa um `server.js` padrão, um módulo poderoso e embutido no framework que orquestra toda a inicialização.
- Este script constrói uma aplicação `express.js` e carrega todos os serviços CAP definidos no seu modelo.
- O módulo pode ser acessado programaticamente via `cds.server`, que é um atalho para `require('@sap/cds/server')`.



A Coreografia do `server.js` Padrão



1. Preparar App Express

Cria a instância da aplicação `express.js` que servirá de base para tudo.



2. Carregar Modelos

Carrega e compila todos os modelos de dados e serviços (`cds.load('*')`).



3. Conectar Serviços Essenciais

Conecta-se a serviços como banco de dados (`cds.connect.to('db')`) e mensageria.



4. Servir Serviços da Aplicação

Monta os endpoints HTTP para todos os serviços definidos (`cds.serve('all')`).



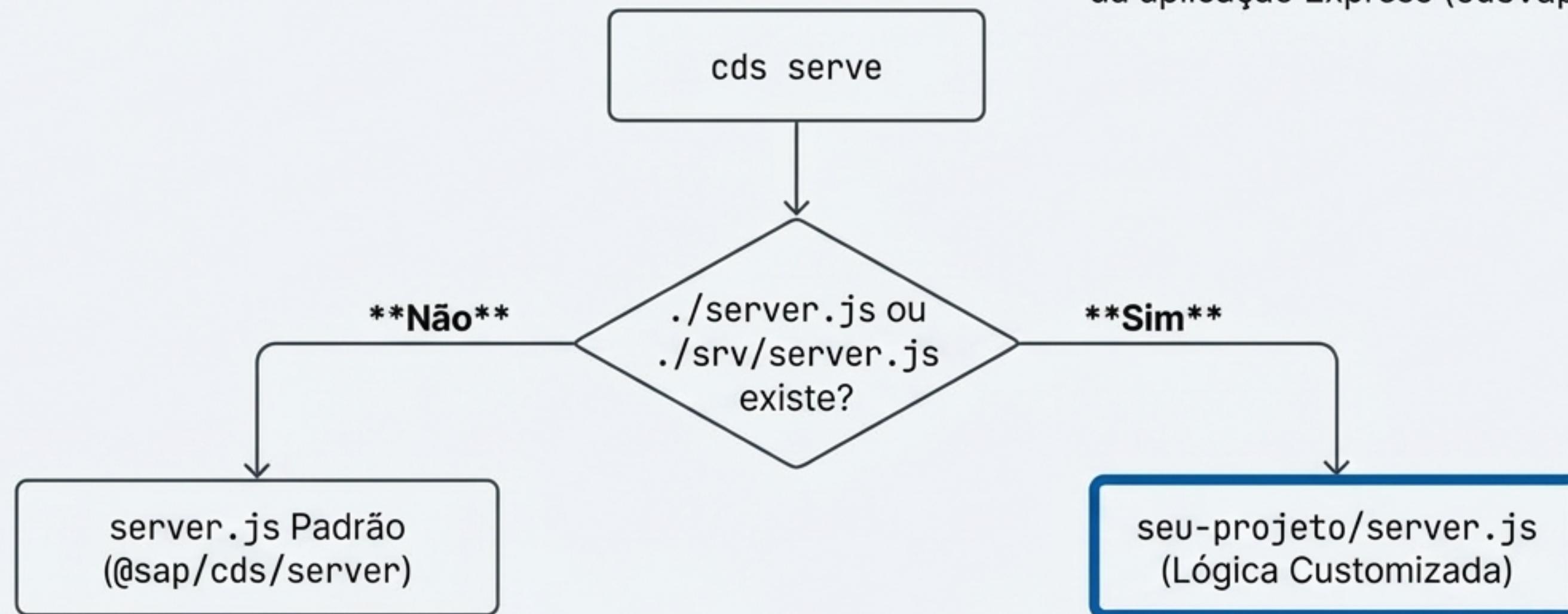
5. Iniciar Servidor HTTP

Inicia o servidor e começa a ouvir requisições na porta configurada (`app.listen`).

Camada 3: Assumindo o Controle com um `server.js` Customizado

Quando usar um `server.js` customizado?

- Para adicionar middlewares personalizados (autenticação, logging, CORS avançado).
- Para modificar opções de inicialização antes da execução padrão.
- Para obter controle total sobre a instância da aplicação Express (cds.app).



Dois Caminhos para a Customização no seu `server.js`

Plug-in nos Eventos (Abordagem Cirúrgica)

Use `cds.on(...)` para injetar lógica em momentos específicos do ciclo de vida. Esta é a abordagem preferida para a maioria dos casos por ser limpa e desacoplada.

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

// Adiciona middleware antes de qualquer rota do CAP
cds.on('bootstrap', app => {
  app.use('/meu-middleware', (req, res, next) => {
    // Sua lógica customizada aqui
    next();
  });
});
```

Sobrescrever `cds.server()` (Controle Total)

Exporte sua própria função para acessar e modificar as opções de linha de comando antes de delegar para o servidor padrão. Use com cautela.

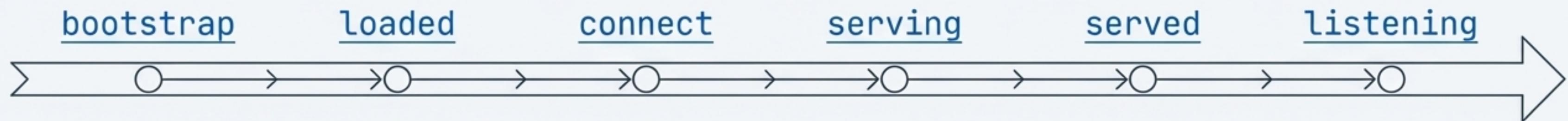
```
const cds = require('@sap/cds');
const express = require('express');

module.exports = (o) => {
  // Ex: Crie e configure seu próprio app Express
  o.app = express();
  o.app.use(require('helmet')()); // Adiciona middleware de segurança

  // Delegue para a implementação padrão com suas opções
  return cds.server(o);
};
```

Camada 4: O Ciclo de Vida do Servidor em Eventos

O CAP emite eventos em cada etapa chave do bootstrap. Você pode registrar handlers usando `cds.on()` para injetar sua lógica.



***Execução Síncrona:** Handlers de eventos executam de forma síncrona, na ordem em que são registrados. As únicas exceções são os eventos `served` e `shutdown`, que suportam handlers assíncronos (`async`).

Eventos em Foco: `bootstrap` e `served` na Prática

`bootstrap`

Quando

Imediatamente após a criação do app Express, antes de qualquer middleware do CAP ser adicionado.

Uso Ideal

Adicionar middlewares globais, como `express.static` para servir recursos estáticos (CSS, imagens, etc.) ou parsers de body customizados.

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

// Serve a pasta 'public' na raiz do servidor
cds.on('bootstrap', app => {
  app.use('/public', express.static(__dirname +
  '/srv/public'));
});
```

`served`

Quando

Após todos os serviços terem sido montados e estarem prontos para uso. Suporta handlers assíncronos.

Uso Ideal

Interagir com as instâncias de serviço já inicializadas, por exemplo, para adicionar dados iniciais ou configurar listeners.

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

// Acessa os serviços após estarem prontos
cds.on('served', async (services) => {
  const { CatalogService, db } = services;
  console.log(`Serviço de Catálogo está disponível:`,
  CatalogService.name);
  // const tx = db.tx(); -> Iniciar transações, etc.
});
```

Conhecendo seu Kit de Ferramentas de Eventos

`loaded`

Emitido sempre que um modelo CDS é carregado via `cds.load()`. Útil para inspecionar ou modificar o modelo (CSN) dinamicamente.

`connect`

Emitido para cada serviço construído via `cds.connect`, como o serviço de banco de dados (`db`). Ocorre antes de `serving`.

`serving`

Emitido para cada serviço de aplicação montado por `cds.serve`. Permite modificar uma instância de serviço antes que ela comece a aceitar requisições.

`listening`

Emitido quando o servidor HTTP está online e pronto. Fornece o objeto `server` e a `url` completa, incluindo a porta.

`shutdown`

Emitido no encerramento do processo. Ideal para tarefas de limpeza (cleanup), como fechar conexões. Suporta handlers assíncronos.

Camada 5: Ajustes Finos via Configuração

Muitos comportamentos do `server.js` padrão podem ser customizados declarativamente, sem escrever lógica JavaScript.

As configurações podem ser definidas na seção `cds` do `package.json` ou em arquivos dedicados como `.cdsrc.json`.

Esta abordagem é ideal para ajustes de ambiente e configurações que não mudam dinamicamente.

```
{  
  "cds": {  
    "requires": {  
      "db": {  
        "kind": "sqlite",  
        "credentials": {  
          "database": "db.sqlite"  
        }  
      }  
    },  
    "server": {  
      "port": 4004,  
      "//": "Outras configurações de servidor aqui..."  
    }  
  }  
}
```

Configuração na Prática: Body Size e CORS

Limite do Tamanho do Request Body

Problema: O limite padrão de `100kb` para o corpo da requisição é muito baixo para uploads de arquivos.

Solução Global (`.cdsrc.json`):

```
"cds": { "server": { "body_parser": { "limit":  
"5mb" } } }
```

Solução Específica (anotação no CDS):

```
annotate CatalogService with  
@cds.server.body_parser.limit: '1mb';
```

⚠️ Nota: A anotação específica do serviço tem precedência sobre a configuração global.

Cross-Origin Resource Sharing (CORS)

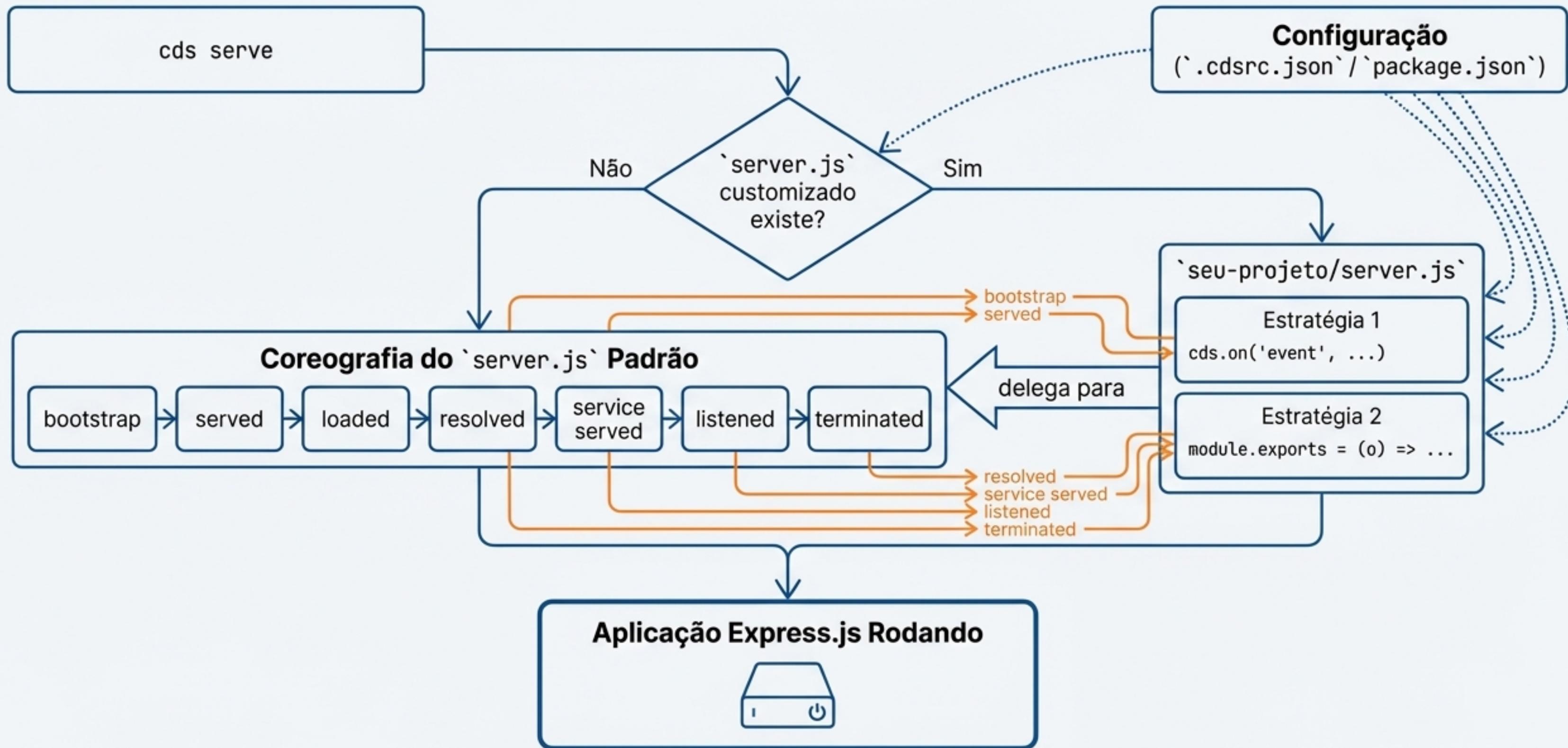
Comportamento Padrão: Habilitado por padrão em todos os ambientes, exceto quando `NODE_ENV=production`.

Customização (`.cdsrc.json`):

```
"cds": {  
  "server": {  
    "cors": {  
      "origin": "https://meu-app.com"  
    }  
  }  
}
```

⚠️ Nota: As opções disponíveis são as mesmas do popular middleware `cors` do Express.js.

O Quadro Completo: Juntando Todas as Peças



Princípios para Lembrar



Confie no Padrão.

O `server.js` embutido é robusto e lida com a maior parte do trabalho pesado para você. Não o substitua sem um bom motivo.



Estenda, Não Reinvente.

Use um `server.js` customizado para adicionar funcionalidades, mas delegue à implementação padrão sempre que possível para manter a compatibilidade futura.



Prefira Eventos.

Os eventos do ciclo de vida são a forma mais limpa e desacoplada de injetar lógica nos momentos certos, sem interferir no fluxo principal.



Configure Antes de Codificar.

Sempre verifique as opções de configuração. A solução para seu problema pode ser apenas uma linha de JSON, o que é mais simples e seguro do que código customizado.

Próximos Passos na sua Jornada

Para Customizações Reutilizáveis



Explore a técnica de cds-plugin para encapsular e compartilhar extensões do servidor entre múltiplos projetos. É a evolução natural da customização via `server.js`.



A Fonte da Verdade

Consulte a documentação oficial do SAP CAP para detalhes aprofundados, opções de configuração adicionais e as últimas atualizações do framework.



Documentação Oficial SAP CAP - Bootstrapping

Obrigado!