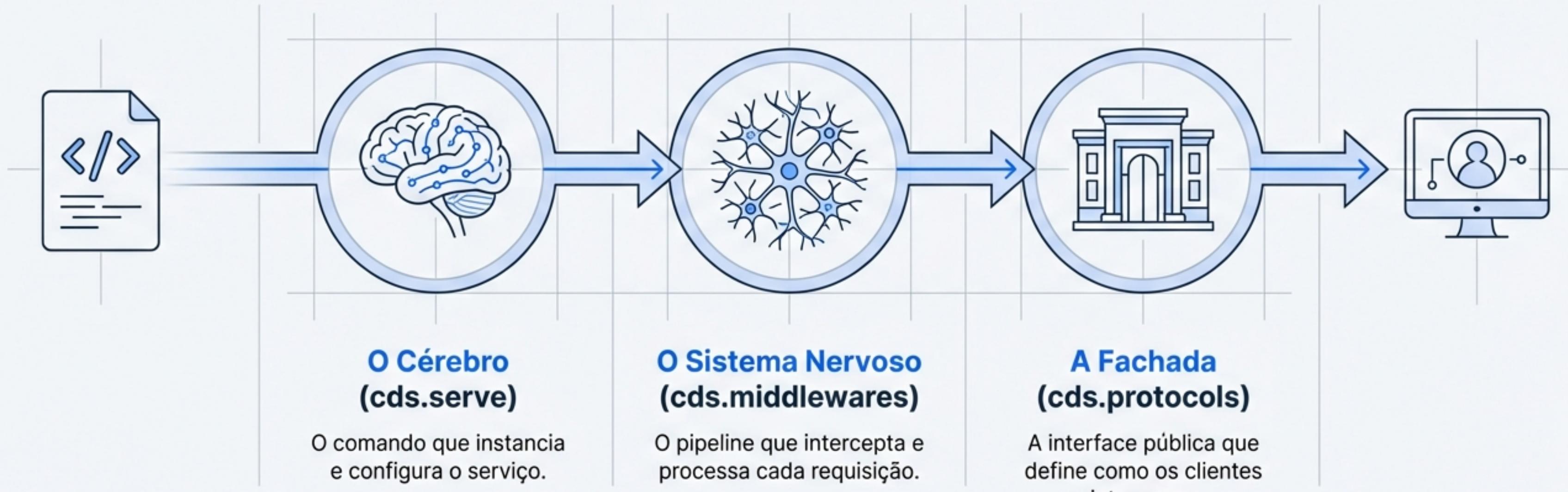


A Anatomia de um Serviço CAP

Um guia visual para orquestrar, controlar e expor seus serviços Node.js.

Do Código ao Cliente: Como um Serviço Ganha Vida?

Você definiu seus modelos e entidades em CDS. O próximo passo é crucial: como tornar essa lógica acessível, segura e utilizável? Esta apresentação dissecá o processo em três sistemas vitais.



`cds.serve`: O Comando que Inicia Tudo

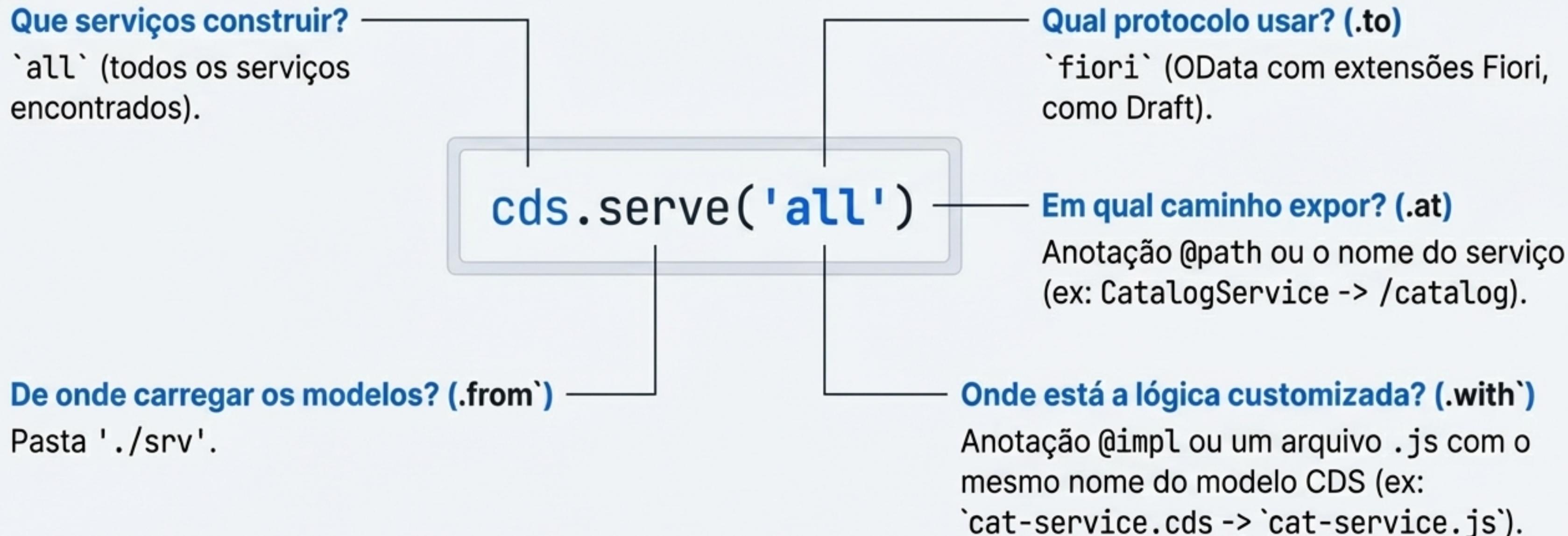
O `cds.serve` é o ponto de partida para expor qualquer serviço. Ele localiza as definições, constrói os provedores de serviço e os prepara para serem servidos. É o centro de comando da sua aplicação.

```
// A forma mais comum de iniciar, em um custom server.js:  
const app = require('express')()  
cds.serve('all').in(app)  
app.listen()
```

Os serviços construídos são cacheados em `cds.services`, permitindo que sejam acessados por `cds.connect` ou estendidos por chamadas subsequentes ao `cds.serve`.

A Simplicidade do `cds.serve('all')` e Seus Padrões Inteligentes

Um único comando é suficiente para servir todos os serviços de um projeto padrão. O CAP utiliza um conjunto de padrões poderosos para minimizar a necessidade de configuração explícita.



Desconstruindo `cds.serve`: A API Fluente

Para controle granular, `cds.serve` oferece uma API fluente. Vamos dissecar os métodos essenciais para definir a fonte dos modelos e a lógica de implementação.

.from(model): Especificando a Fonte (O 'Quê')

Define de onde carregar as definições de serviço.

- Nome de um arquivo (`'my-services.cds'`)
- Nome de uma pasta (`'./srv'`)
- Um objeto CSN já parseado

```
const csn = await cds.load('my-services.cds')
cds.serve('all').from(csn)
```

.with(impl): Anexando a Lógica ("Como")

Conecta a definição do serviço a uma função ou módulo que contém os event handlers. Só pode ser usado ao servir um único serviço.

```
// Usando um arquivo de implementação
cds.serve('./srv/cat-service.cds').with('./srv/cat-s
```

```
// Usando uma função inline
cds.serve('CatalogService').with(srv => {
  srv.on('READ', 'Books', (req) => req.reply([...]))
})
```

Desconstruindo `cds.serve`: Integração e Exposição

A API fluente também controla como e onde seu serviço é exposto, definindo o servidor, o protocolo de comunicação e o endpoint público.

.in(app): Onde o Serviço Vive

Adiciona os provedores de serviço como roteadores a uma aplicação Express existente.

```
const app = require('express')()
cds.serve('all').in(app)
```

.to(protocol): Como o Serviço se Comunica

Define o protocolo de exposição.

- 'rest'
- 'odata'
- 'fiori' (padrão)

.at(path): O Endereço Público

Especifica programaticamente o ponto de montagem do serviço. Só funciona ao servir um único serviço.

```
cds.serve('CatalogService').at('/cat')
// Substitui a anotação @path ou o nome do serviço
```

Parte 2: O Sistema Nervoso

cds.middlewares

`cds.middlewares`: Controlando o Fluxo de Requisições

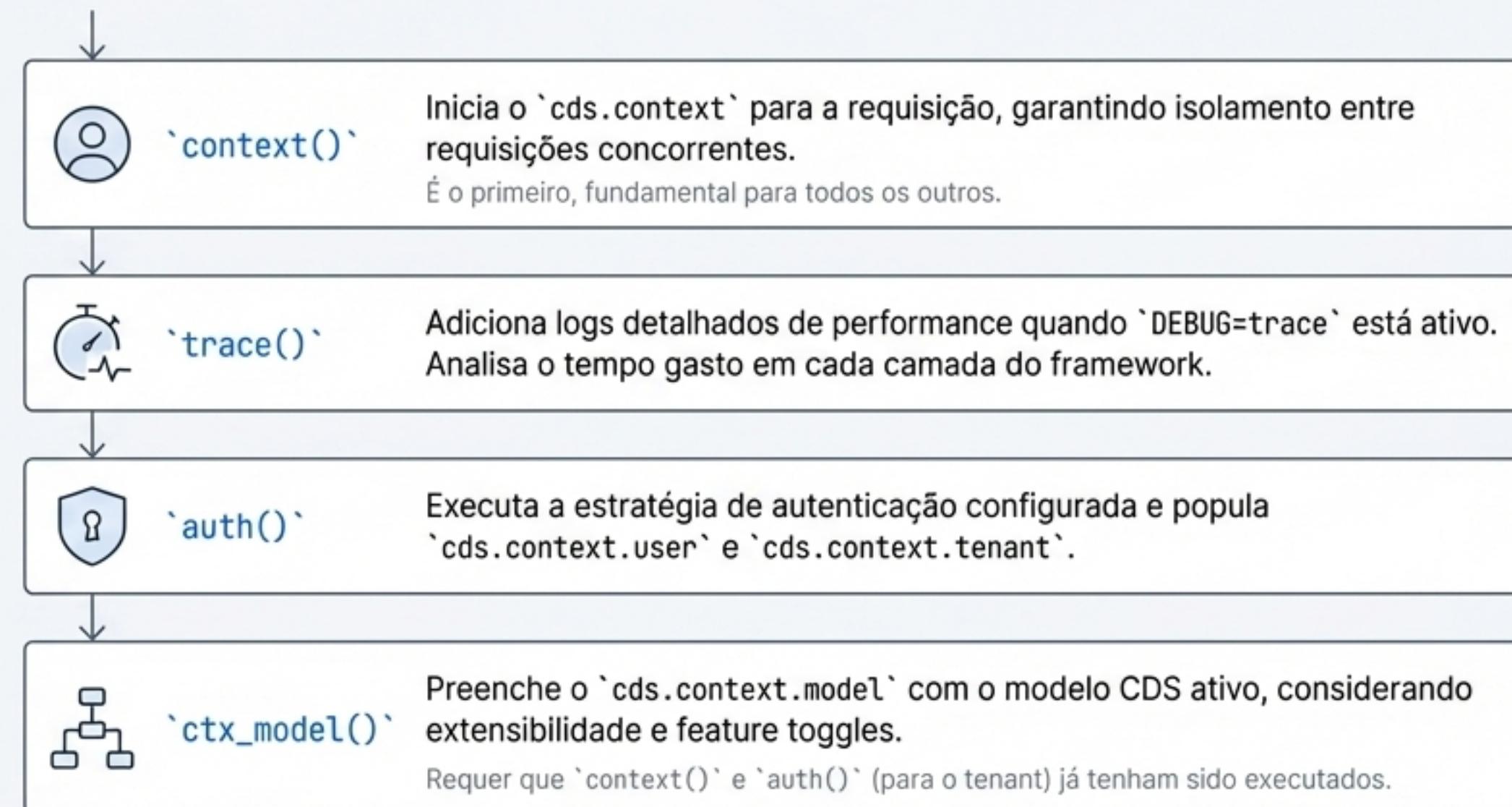
Um serviço em execução é apenas o começo. Para cada requisição que chega, uma série de etapas é executada. O `cds.middlewares` é o sistema nervoso da sua aplicação: um pipeline que processa o contexto, a autenticação e a lógica customizada antes que a requisição chegue ao seu serviço.



```
// Como o framework registra os middlewares
app.use(cds.middlewares.before, protocol_adapter)
```

A Anatomia do Pipeline Padrão de Middlewares

Por padrão, o CAP registra um conjunto de middlewares essenciais em uma ordem específica. Compreender este fluxo é fundamental para a customização e para a depuração de problemas.



A interdependência é crucial. `ctx_model` depende de `context` e `auth`. A ordem importa.

Customização e Injeção com `cds.middlewares.add()`

O pipeline padrão pode ser estendido para injetar sua própria lógica. Use `cds.middlewares.add()` para adicionar novos middlewares em posições específicas da cadeia. A configuração deve ser feita programaticamente, programaticamente, por exemplo, em um `server.js`.

Sintaxe da Função

```
cds.middlewares.add(middleware, position?)
```

Exemplos de Posição

- {at: 0}: Adiciona ao início do pipeline.
- {before: 'auth'}: Adiciona imediatamente antes do middleware de autenticação.
- {after: 'auth'}: Adiciona imediatamente após o middleware de autenticação.
- **Sem segundo argumento**: Adiciona ao final do pipeline.

Código de Exemplo

```
// Adiciona um middleware customizado antes da autenticação
const myMiddleware = (req, res, next) => {
  const myMiddleware = (req, res, next) => {
    console.log('Custom logic running!');
    next();
};

cds.middlewares.add(myMiddleware, { before: 'auth' });
```



Evite sobrescrever completamente a lista `cds.middlewares.before`. Use `add()` para garantir que futuras atualizações do framework sejam incorporadas automaticamente.

Casos de Uso Práticos para Middlewares Customizados

Middlewares são perfeitos para tarefas transversais, como enriquecer o contexto do usuário ou habilitar features dinamicamente.

Exemplo 1: Customizando o `cds.context.user`

Adicionar um prefixo ao ID do usuário após a autenticação padrão.

Posicionamento: após `auth()` e antes de `ctx_model()`.

```
cds.middlewares.add(function ctx_user(req, res, next) {  
    const ctx = cds.context;  
    if (ctx.user) {  
        ctx.user.id = '<my-idp>' + ctx.user.id;  
    }  
    next();  
}, { after: 'auth' });
```

Exemplo 2: Habilitando Feature Flags

Definir um vetor de features ativas para a requisição atual.

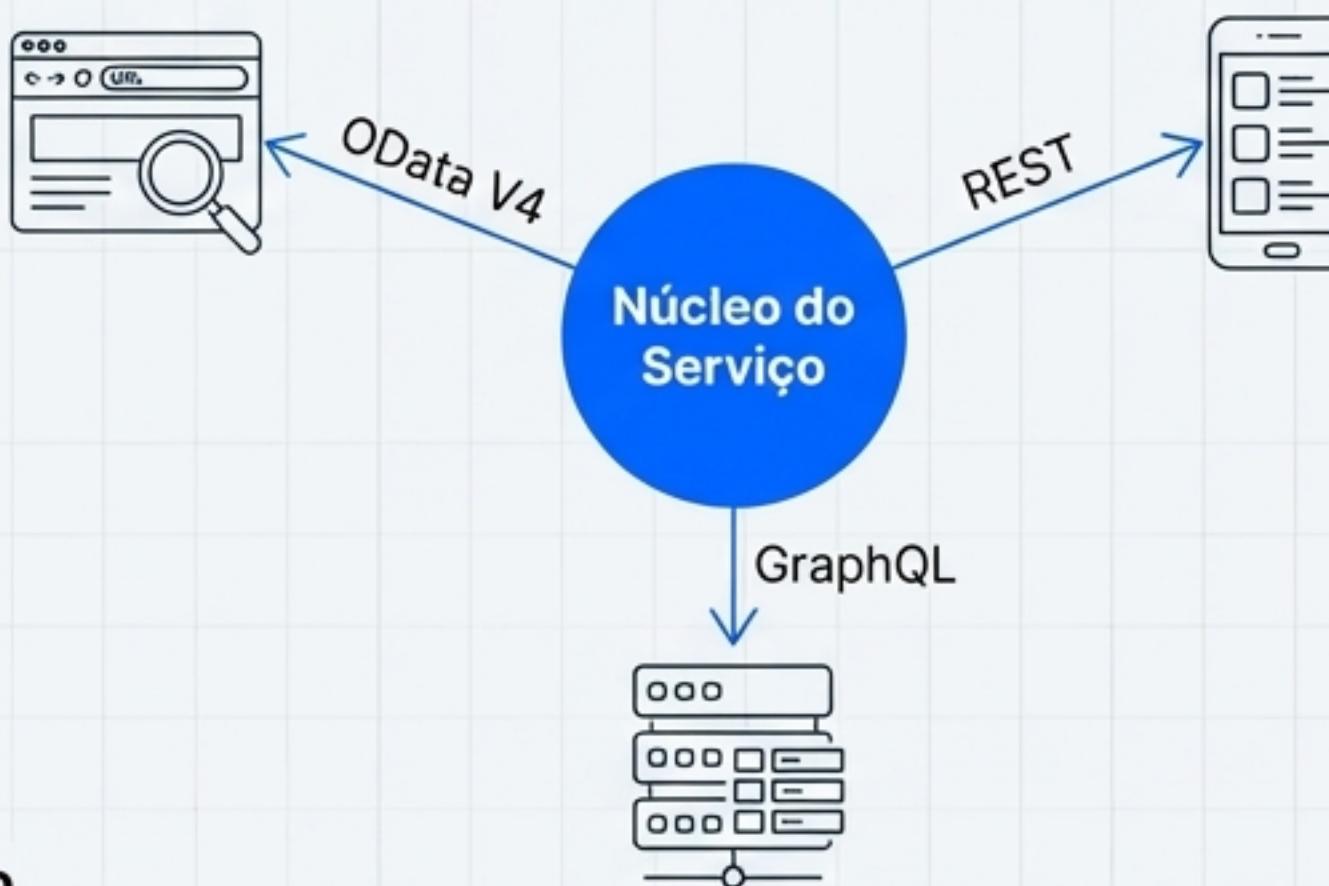
Posicionamento: antes de `ctx_model()`.

```
cds.middlewares.add(function req_features(req, res, next)  
    //  
    req.features = ['<feature-1>', '<feature-2>'];  
    next();  
, { before: 'ctx_model' });
```

Parte 3: A Fachada

`cds.protocols`: Definindo a Interface com o Mundo

Com o serviço em execução e o fluxo de requisições sob controle, a última etapa é definir como ele será exposto para os clientes. O CAP permite que o mesmo serviço seja servido através de múltiplos protocolos simultaneamente, como OData, REST e GraphQL.



Padrões de Caminho

OData V4:	/odata/v4
REST:	/rest
GraphQL:	/graphql` (requer o pacote @cap-js/graphql)

O Poder do Multi-Protocolo com Anotações

As anotações `@protocol` e `@path` no seu arquivo CDS são a forma mais direta de configurar como e onde um serviço é exposto.

@protocol: Configura em quais protocolos um serviço é servido.

```
// Padrão (OData V4) ou atalho          // Expondo em múltiplos protocolos      // Tratando como um serviço interno  
```cds                                         ```cds                                         ```cds  
@odata @protocol: ['odata', 'rest'] @protocol: 'none'
service CatalogService { ... } service CatalogService { ... } service InternalService { ... }
...
//> /odata/v4/catalog ...
 //> /odata/v4/catalog e /rest/catalog
```

---

## **@path: Configura o caminho específico de um serviço.**

```
// Caminho relativo ao protocolo // Caminho absoluto (impede múltiplos protocolos)
```cds                                         ```cds  
@path: 'browse'                           @path: '/browse'  
service CatalogService { ... }            service CatalogService { ... }  
...  
//> /odata/v4/browse                      ...  
                                              //> /browse
```

Tópicos Avançados: Comportamento de Update e Protocolos Customizados

O CAP oferece configurações para refinar o comportamento HTTP e flexibilidade para estender o framework com seus próprios adaptadores de protocolo.

PATCH vs. PUT: Refinando a Semântica de Modificação

- **PATCH**: Modificação parcial de um recurso existente. Por padrão, não cria se não existir (`patch_as_upsert: false`).
- **PUT**: Garante que um recurso existe (cria ou atualiza). Por padrão, cria se não existir (`put_as_upsert: true`).

Esses comportamentos podem ser alterados via `cds.runtime.patch_as_upsert` e `cds.runtime.put_as_upsert`.

Adaptadores de Protocolo Customizados

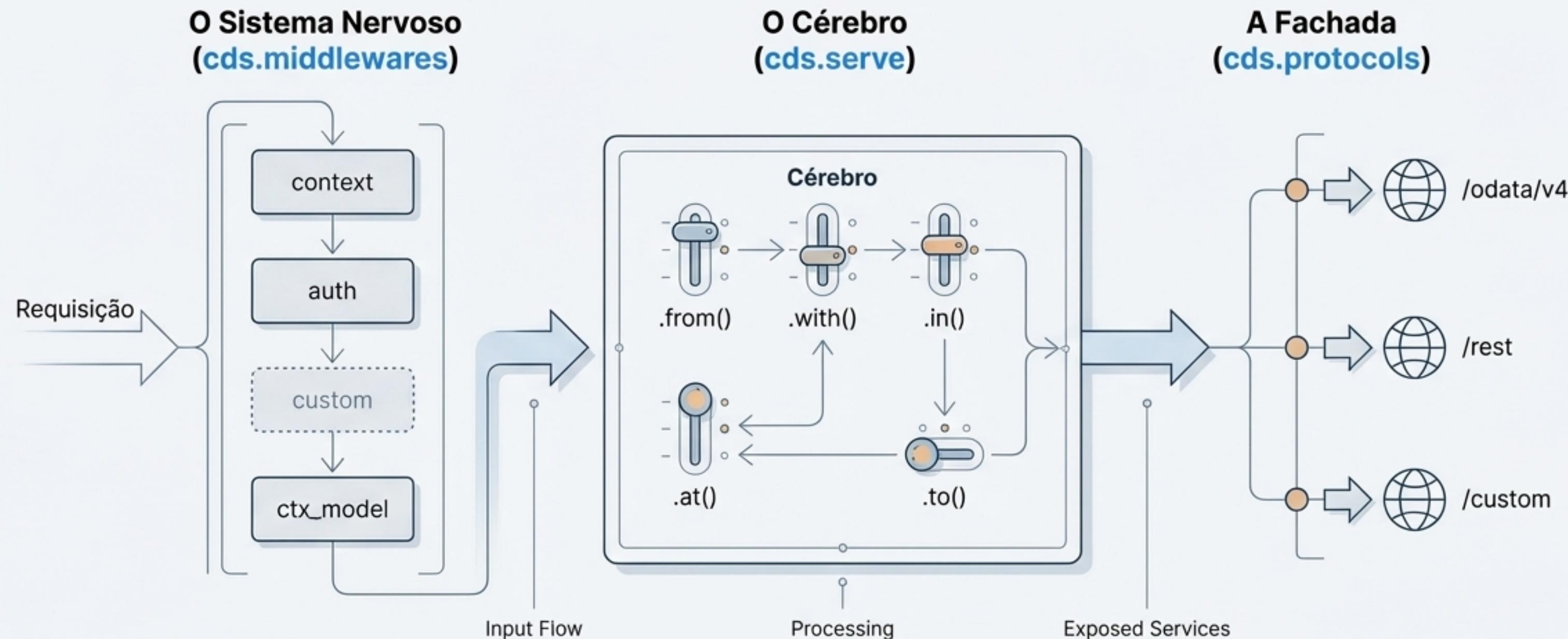
É possível registrar um adaptador para um protocolo totalmente novo programaticamente.

Código de Configuração:

```
// Em um arquivo de configuração ou server.js
cds.env.protocols = {
  'custom-protocol': {
    path: '/custom',
    impl: '<custom-impl.js>',
    // ... outras opções
  }
}
```

A Anatomia Completa: Do Código ao Cliente

Estes três sistemas — `serve`, `middlewares` e `protocols` — formam a anatomia completa de um serviço CAP. Dominá-los significa ter controle total sobre como sua lógica de negócio é instanciada, processada e exposta.



Este é o seu toolkit para construir serviços robustos, seguros e escaláveis com o SAP Cloud Application Programming Model.