

Uma História de Dois Ambientes: Dominando o `cds.log` do Desenvolvimento Local à Nuvem de Produção

Um guia prático para transformar logs de simples saídas de console em telemetria poderosa em suas aplicações CAP Node.js.



O que é `cds.log`? Mais que um Simples `console.log`.

`cds.log` é uma fachada de log minimalista e sensível ao contexto. Ela oferece uma API familiar, semelhante ao `console`, mas com superpoderes para configuração, formatação e integração.



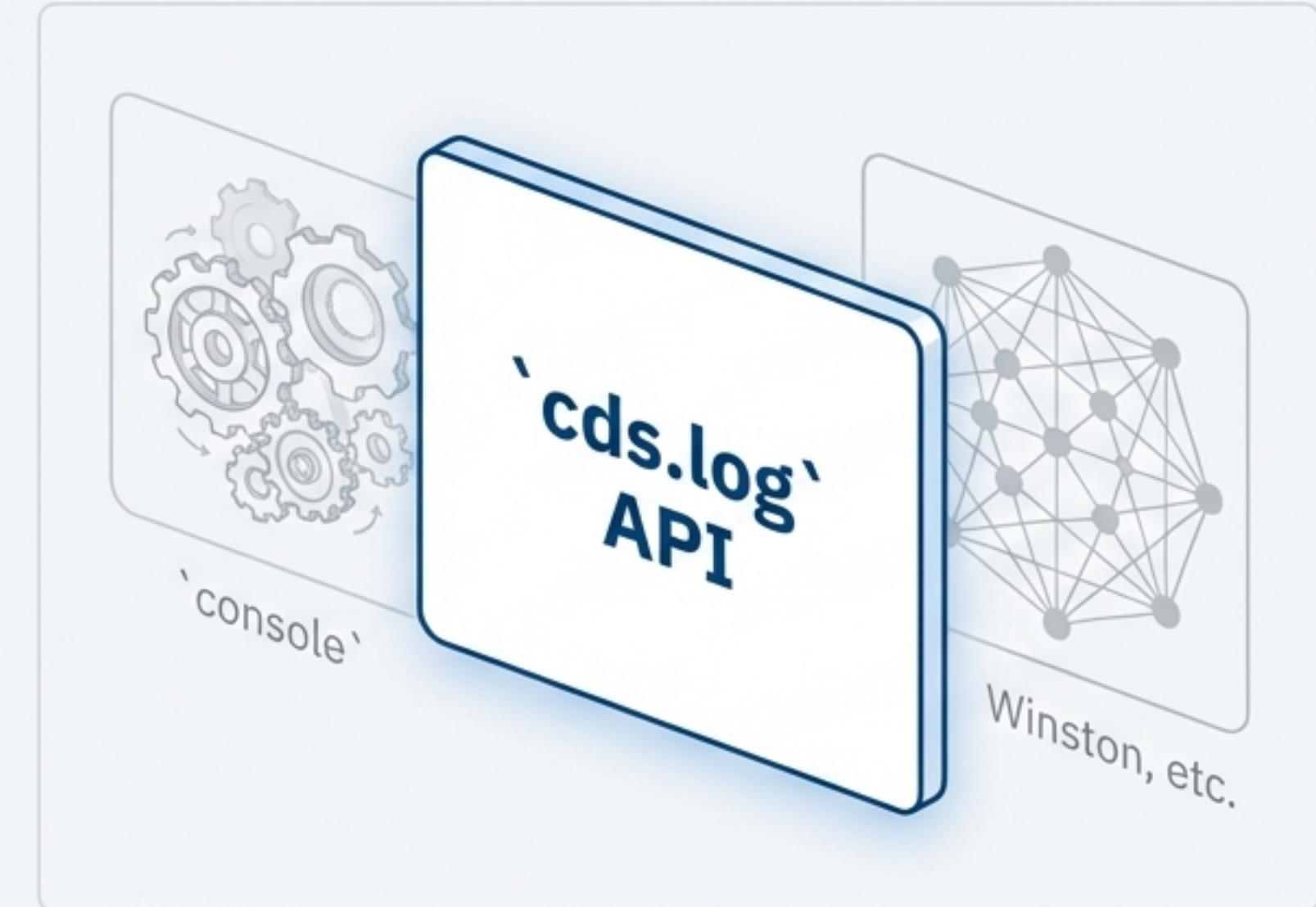
Simples de Usar: Obtenha um logger com um ID único: `const LOG = cds.log('sql')`.



Eficiente: Loggers com o mesmo ID são instâncias compartilhadas e cacheadas: `cds.log('foo') === cds.log('foo') //> true`.



Inteligente: Adapta seu comportamento e formato de saída com base no ambiente (desenvolvimento vs. produção).



```
// Obtenha um logger para o seu módulo
const LOG = cds.log('my-service');

// Use-o como o console
LOG.info('Serviço iniciado com sucesso');
LOG.debug('Conectando ao banco de dados...');
```

A Realidade do Desenvolvedor: Necessidades Diferentes para Mundos Diferentes

No Desenvolvimento



- **Clareza Imediata:** Logs legíveis por humanos para depuração rápida.
- **Feedback Rápido:** Saída instantânea no terminal.
- **Stack Traces Clicáveis:** Navegação fácil para a origem do erro.
- **Foco no Código:** Mínima poluição visual, máximo sinal.

Em Produção



- **Observabilidade:** Logs estruturados e analisáveis por máquinas.
- **Correlação:** Rastrear uma única requisição através de múltiplos serviços.
- **Análise e Alertas:** Filtrar por severidade, componente ou usuário.
- **Segurança:** Mascara dados sensíveis.

Ato I - Desenvolvimento: Foco na Clareza e Velocidade

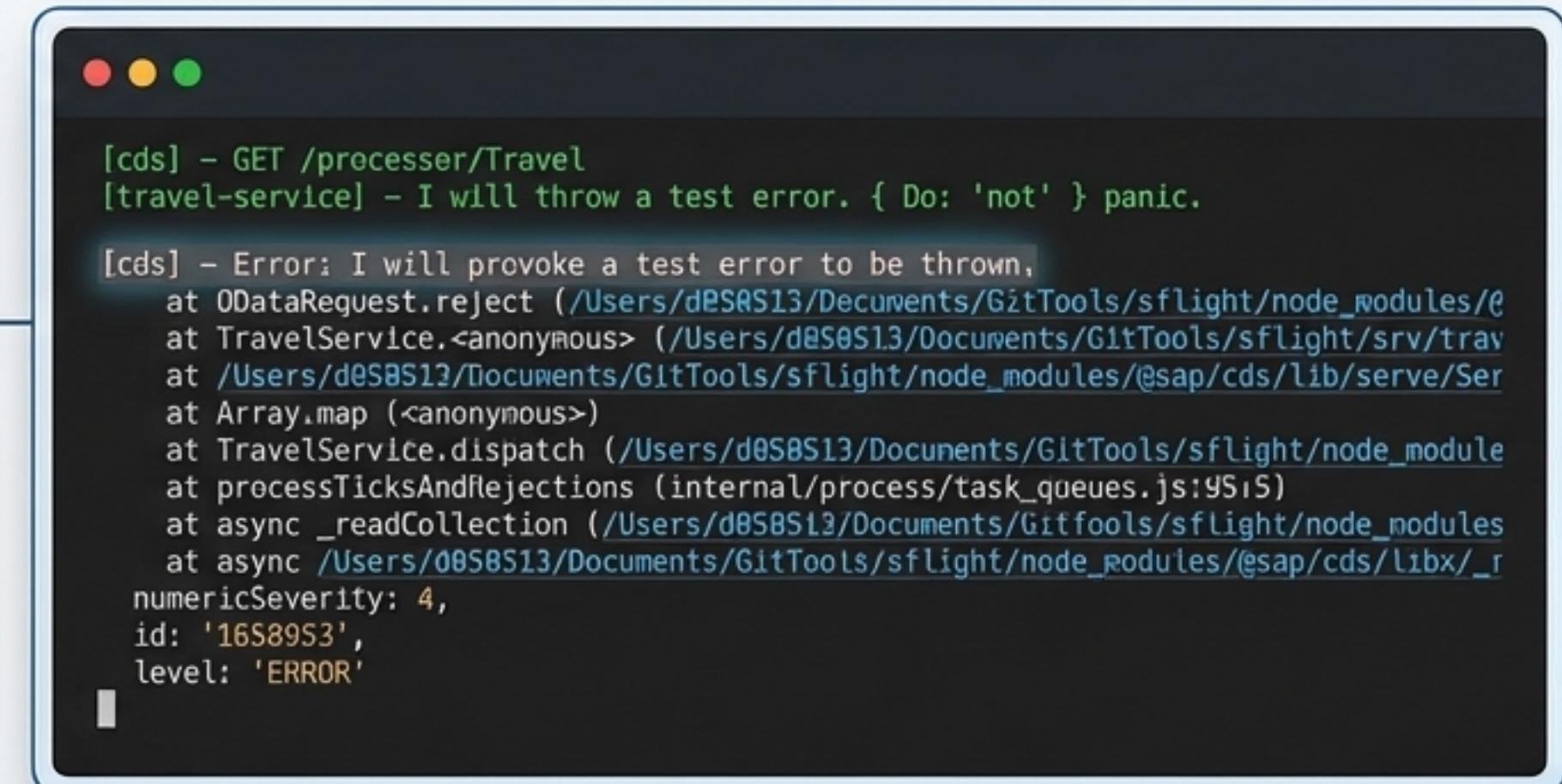
No ambiente de desenvolvimento, o `cds.log` utiliza o formatador `plain` por padrão. O objetivo é fornecer uma saída limpa, concisa e legível por humanos diretamente no seu console.

O formatador `plain` simplesmente prefixa sua mensagem de log com `'[<id>] - '.

Código & Efeito

```
const LOG = cds.log('travel-service');

LOG.info('I will throw a test error. Do: "Not'
          'panic.');
LOG.error('Error: I will provoke a test error to
          be thrown.');
```



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are three colored window control buttons (red, yellow, green). Below them, the log output is displayed:

```
[cds] - GET /precesser/Travel
[travel-service] - I will throw a test error. { Do: 'not' } panic.

[cds] - Error: I will provoke a test error to be thrown,
      at 0DataRequest.reject (/Users/d0S0S13/Documents/GitTools/sflight/node_modules/@sap/cds/lib/serve/Ser
      at TravelService.<anonymous> (/Users/d0S0S13/Documents/GitTools/sflight/srv/trav
      at /Users/d0S0S13/Documents/GitTools/sflight/node_modules/@sap/cds/lib/serve/Ser
      at Array.map (<anonymous>)
      at TravelService.dispatch (/Users/d0S0S13/Documents/GitTools/sflight/node_module
      at processTicksAndRejections (internal/process/task_queues.js:95:15)
      at async _readCollection (/Users/d0S0S13/Documents/GitTools/sflight/node_modules/@sap/cds/libx/_r
      numericSeverity: 4,
      id: '1658953',
      level: 'ERROR'
```

O Superpoder do Desenvolvedor: Ativando Logs Detalhados com `DEBUG`

Esqueça adicionar e remover `console.log`. Use a variável de ambiente `DEBUG` para ativar dinamicamente a saída de depuração para módulos específicos, diretamente da linha de comando.

```
...  
[cds] - using profiles: [ 'development' ]  
[cds] - serving TravelService at /travel  
[cds] - server listening on { url: 'http://localhost:4804' }  
[cds] - launched in: 615.710ms
```

Antes

```
...  
> DEBUG=sql,app cds watch
```

Comando

```
...  
[cds] - using profiles: [ 'development' ]  
[sql] - BESIN  
[sql] - SELECT count(*) AS "count" FROM "TravelService_Travel"  
AS "T8" WHERE "T8"."TravelUUID" = 73  
[sql] - SELECT "T8"."TravelUUID", "T8"."BeginDate", "T8"."EndDate"  
FROM "TravelService_Travel" AS "T8" WHERE "T8"."TravelUUID" = 73  
[sql] - COMNET;  
[eds] - serving TravelService at /travel  
[cds] - server listening on { url: 'http://localhost:4804' }  
[cds] - launched in: 728.113ms  
[app] - TravelService: Processing request...
```

Depois

Como Funciona

- Ativar Módulos Específicos: `DEBUG=sql,app cds watch`
- Ativar Tudo: `DEBUG=all cds watch`

Dica Pro

Você pode registrar um logger com múltiplos IDs para corresponder a diferentes configurações de `DEBUG`. O primeiro ID é usado como o rótulo.

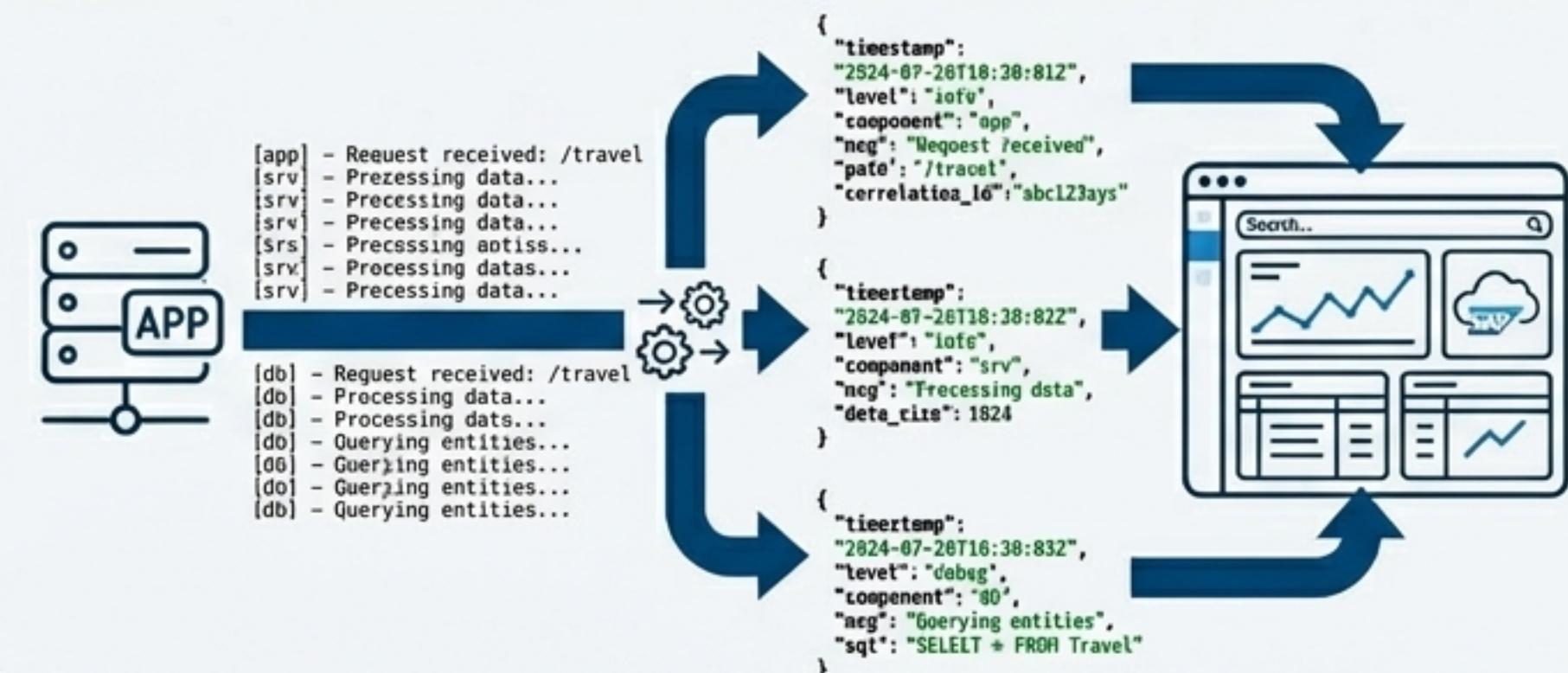
```
// Este logger será ativado por DEBUG=db ou DEBUG=sql  
const LOG = cds.log('db|sql');
```

Ato II - Produção: Foco na Análise e Observabilidade

Em produção, os logs não são para leitura humana em tempo real. Eles são uma corrente de dados estruturados enviados para serviços de análise como o SAP Cloud Logging ou Application Logging Service.

`cds.log` detecta automaticamente um ambiente de produção e muda para o formatador `JSON`. Cada linha de log se torna um objeto JSON rico em metadados.

- ✓ **Filtragem Poderosa:** Encontre logs por nível, `correlation_id`, componente, etc.
- ✓ **Análise de Dados:** Crie dashboards e visualizações.
- ✓ **Alertas Automatizados:** Dispare ações com base em padrões de log.



- i **Dica:** Para desativar o formato JSON em produção (não recomendado), você pode configurar `cds.log.format: plain`.

Anatomia de um Log de Produção

O formatador JSON enriquece sua mensagem de log com contexto da requisição, informações do **tenant** e muito mais.

Time	component_name	correlation_id	level	logger	msg
Jul 22, 2021 at 14:27:06.822	aflight	55f14cd6-4206-4d00-a75b-0753b2c98d3e	ERROR	cds	I will provoke a test error to be thrown.

```
{  
  "timestamp": "2021-07-22T14:27:06.820Z",  
  "level": "ERROR",  
  "logger": "cds",  
  "msg": "I will provoke a test error to be thrown.",  
  "correlation_id": "55f14cd6-4206-4d00-a75b-0753b2c98d3e",  
  "tenant_id": "t1",  
  "remote_user": "alice",  
  ...  
}
```

Quando o evento ocorreu.

Nível de severidade (ERROR, WARN, INFO).

O ID do logger (ex: `cds`, `sql`).

A mensagem de log principal.

O ID único que conecta todos os logs de uma única requisição.

Identifica o tenant (em cenários multitenant).

O ID do usuário (se configurado).

space_id

55f14cd6-4206-4d00-a75b-0753b2c98d3e

* eyeilog_hostname

"

* tenant_id

"t1"

NotebookLM

O ‘Momento Mágico’: Rastreando Requisições de Ponta a Ponta

A “correlação de requisições” é a capacidade de visualizar todos os eventos de log associados a uma única operação. `cds.log` lida com isso automaticamente.

1. CAP inspeciona **headers** de entrada como `x-vcap-request-id`, `x-correlation-id`, etc.
2. Um ID único é garantido e armazenado em `cds.context.id`.
3. O formatador JSON inclui este ID no campo `correlation_id` de cada log.

The screenshot shows the Elastic Stack interface for 'Requests and Logs'. A search bar at the top contains filters: 'space_name: sflight' and 'correlation_id: SSIFT14c6-4206-405b-4706-8723e2c85d3e'. Below the search bar, a 'Correlation IDs' section lists one item: 'correlation_id : SSIFT14c6-4206-405b-4706-8723e2c85d3e' with a count of 4. The main area is divided into 'Requests' and 'Application Logs' sections. The 'Requests' section shows a single log entry:

Time	component_name	correlation_id	level	request	response_status	response_time_ms
> Jul 22, 2021 @ 14:27:05.913	[REDACTED]-sflight	SSIFT14c6-4206-405b-4706-8723e2c85d3e	INFO	/processor/Trevel	500	48.471

The 'Application Logs' section shows three log entries, all sharing the same correlation ID:

Time	component_name	correlation_id	level	logger	msg
> Jul 22, 2021 @ 14:27:06.820	[REDACTED]-sflight	SSIFT14c6-4206-405b-4706-8723e2c85d3e	ERROR	cds	I will provoke a test error to be thrown.
> Jul 22, 2021 @ 14:27:06.814	[REDACTED]-sflight	SSIFT14c6-4206-405b-4706-8723e2c85d3e	WARN	travel-service	I will throw a test error. (Do not panic.)
> Jul 22, 2021 @ 14:27:06.778	[REDACTED]-sflight	SSIFT14c6-4206-405b-4706-8723e2c85d3e	INFO	cds	GET /processor/Trevel

A callout box highlights the correlation ID 'SSIFT14c6-4206-405b-4706-8723e2c85d3e' in the logs and states: "O Resultado: A história completa de uma requisição, desde a camada de protocolo até o banco de dados e de volta."



Protegendo Dados Sensíveis com Mascaramento de Headers

Logs nunca devem expor segredos, tokens ou informações de identificação pessoal (PII). `cds.log` mascara automaticamente valores de headers sensíveis em logs de produção.

Comportamento Padrão

Valores de headers que correspondem a expressões regulares em `cds.log.mask_headers` são substituídos por `****`.

Aviso: Se sua aplicação usa headers personalizados para dados sensíveis, certifique-se de adicioná-los a esta configuração em seu `package.json`.

Dica: Os matchers rodam no nome original do header (ex: `Foo-Bar`), não na versão normalizada (`foo_bar`).

Configuração Padrão

```
"mask_headers": [  
    "/authorization/i",  
    "/cookie/i",  
    "/cert/i",  
    "/ssl/i"  
]
```

Before:

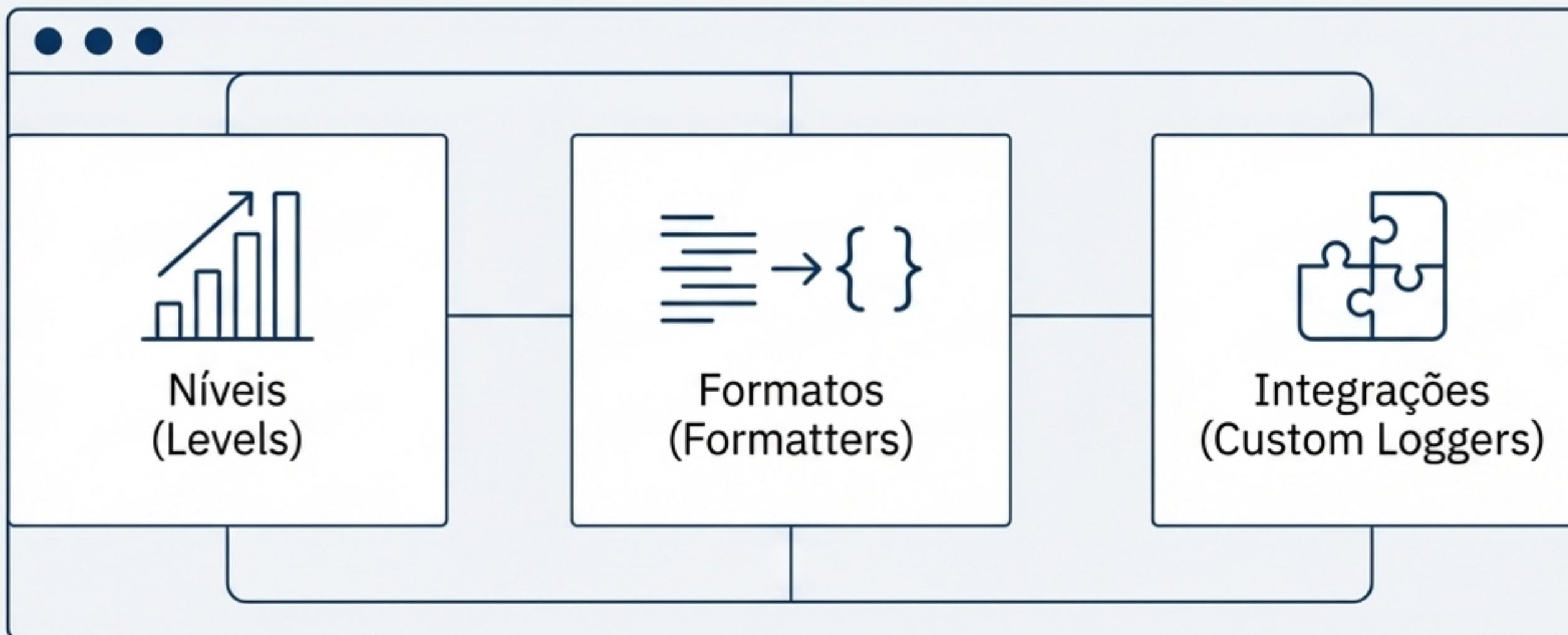
"authorization": "Bearer eyJhbGciOi...verylongtoken..."

After:

"authorization": "****"

O Painel de Controle: Ajustando os Logs à Sua Medida

O comportamento do `cds.log` é altamente configurável. Você tem controle total sobre os níveis de log, formatos e até mesmo a implementação subjacente do logger.



Onde Configurar?

A maioria das configurações estáticas é feita em seu arquivo `package.json` sob a chave `\"cds": { "log": { ... } }`.

Controle de Granularidade: Configurando Níveis de Log

Determine exatamente qual nível de detalhe cada parte da sua aplicação deve registrar.

☰ Declarativo (em `package.json`)

A maneira preferida para definir níveis de log iniciais para diferentes módulos.

```
"cds": {  
  "log": {  
    "levels": {  
      "sql": "debug",  
      "cds": "info",  
      "app": "warn"  
    }  
  }  
}
```

{ } Programático (no Código)

Útil para alterar dinamicamente o nível de log de um logger compartilhado em tempo de execução.

```
// Em outro módulo, eleva o nível do logger 'app'  
cds.log('app', 'debug');
```

Referência Rápida - Níveis Disponíveis

SILENT (SILENT) ERROR (#DC3545) WARN (#0A4982) INFO (#0A4982) DEBUG (#28A745) TRACE (IBM Mono)

Além do Padrão: Criando Seus Próprios Formatos de Log

Se os formatadores `plain` e `JSON` não atenderem às suas necessidades, você pode fornecer sua própria função de formatação.

- Uma função de formatação recebe `id`, `level` e os argumentos do log, e deve retornar um array de valores que serão passados para o `console`.

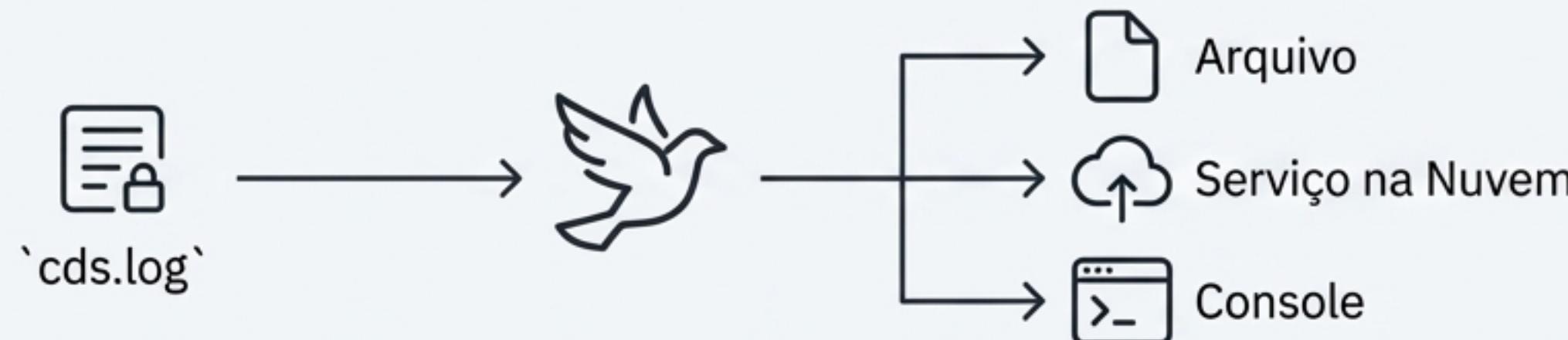
```
// Exemplo de Formato Verboso
const _levels = ['SILENT', 'ERROR', 'WARN', 'INFO', 'DEBUG', 'TRACE'];

cds.log.format = (id, level, ...args) => [
  '[' , (new Date).toISOString(),
  '|', _levels[level].padEnd(5),
  '|', cds.context?.id || '-',
  '|', id, '] -', ...args
];
```

 Dica: Você também pode definir um formato para um logger específico usando `LOG.setFormat(...)`.

Poder de Fogo Total: Integrando com Frameworks Avançados como o Winston

Por ser uma fachada, `cds.log` pode delegar a escrita de logs para bibliotecas mais poderosas, como o `winston`, para cenários complexos como transporte de logs para múltiplos destinos (arquivos, serviços de terceiros, etc.).



A Maneira Fácil

`npm add winston`

```
// em seu server.js
const cds = require('@sap/cds');
cds.log.Logger = cds.log.winstonLogger();
```

Personalização Completa

Passe opções de configuração do `winston`.

```
cds.log.Logger = cds.log.winstonLogger({
  format: winston.format.simple(),
  transports: [
    new winston.transports.File({ filename: 'errors.log', level: 'error' })
  ]
});
```

Escrevendo Logs de Alta Performance

Construir mensagens de log, especialmente para níveis de `debug`, pode ser custoso. Evite trabalho desnecessário se o log nem mesmo será exibido.

Deixe a Formatação para o Logger

✗ **NÃO FAÇA**

```
LOG.debug(`Objeto complexo: ${JSON.stringify(obj)}`);
```

✓ **FAÇA**

```
LOG.debug('Objeto complexo:', obj);
```

Por quê? A serialização do objeto só acontecerá se o nível **debug** estiver ativo.

Verifique o Nível Explicitamente

Para operações ainda mais custosas, use as propriedades booleanas `_<level>` para pular completamente a lógica de construção do log.

```
if (LOG._debug) {  
  const data = computeExpensiveDataForLogging();  
  LOG.debug('Dados computados:', data);  
}
```

Guia de Bolso do `cds.log`

Módulo e ID do Logger

Componente	IDs do Logger
Servidor/Comum	`cds`
Serviços de Aplicação	`app`
Banco de Dados	`db sql`
Serviços Remotos	`remote`
Autenticação	`auth`
Multitenancy	`mtx`
Protocolo OData	`odata`
Auditoria	`audit-log`

Chave de Configuração

`cds.log.levels`

Define o nível de log por ID.

Ex: `{ "sql": "debug\" }`

`cds.log.format`

Força um formato. Ex: `\\"plain\\"`

`cds.log.mask_headers`

Adiciona regras de mascaramento. Ex:
`[" //x-secret-header/i\"]`

`cds.log.user`

Define como `true` para logar `req.user.id`.