Primeiros Passos — Egypteam Laravel App Logger (pt-BR)

Última atualização: 2025-12-26

Este guia mostra uma integração **fim a fim** do `egypteam/laravel-app-logger` em uma aplicação Laravel:

- instalar o pacote
- configurar o canal `app_log`
- definir variáveis de ambiente (CloudWatch)
- adicionar padrões úteis de middleware/contexto
- validar os logs no AWS CloudWatch Logs (e Logs Insights)
- rodar localmente com fallback
- adicionar testes e dicas de troubleshooting

> Premissas: PHP 8.2+, Laravel 10/11/12, Monolog 3. O provider padrão é **CloudWatch**.

---

1) Instalar o pacote

Opção A — Packagist (recomendado)

```
composer require egypteam/laravel-app-logger
```

Opção B — Instalar de um repositório Git

Se você publicar o pacote em um repositório privado, adicione no `composer.json` da aplicação:

```
{
  "repositories": [
    {
      "type": "vcs",
      "url": "git@github.com:EgypTeam/laravel-app-logger.git"
    }
  ]
}
```

Depois instale:

```
composer require egypteam/laravel-app-logger:dev-main
```

Opção C — Path local (desenvolvimento)

Se você estiver desenvolvendo o pacote localmente junto com a aplicação:

```
{
  "repositories": [
    {
      "type": "path",
      "url": "../laravel-app-logger",
      "options": { "symlink": true }
    }
  ]
}
```

Depois:

```
composer require egypteam/laravel-app-logger:dev-main
```

---

2) Publicar a configuração (opcional, mas recomendado)

```
php artisan vendor:publish \
  --provider="Egypteam\\LaravelAppLogger\\LaravelAppLoggerServiceProvider" \
  --tag=config
```

Isso cria:

- `config/app-logger.php`

Se você preferir manter tudo via `.env`, pode pular essa etapa.

---

3) Adicionar o canal `app_log` em `config/logging.php`

Adicione este canal na sua aplicação:

```php
'app_log' => [
    'driver' => 'custom',
    'via'    => \Egypteam\LaravelAppLogger\Logging\AppLoggerService::class,
    'level'  => env('LOG_LEVEL', 'info'),
],
```

Você pode manter o canal padrão do Laravel sem mudanças. O objetivo aqui é ter um **canal dedicado e estruturado** para logs de aplicação.

---

4) Configurar variáveis de ambiente (`.env`)

Config mínima (CloudWatch)

```env
APP_LOG_PROVIDER=cloudwatch
AWS_REGION=us-east-1
CW_LOG_GROUP=my-app
CW_LOG_STREAM=web
```

Config recomendada para produção

```env
APP_LOG_PROVIDER=cloudwatch

AWS_REGION=us-east-1
CW_LOG_GROUP=my-app
CW_LOG_STREAM=web
CW_RETENTION_DAYS=14
CW_BATCH_SIZE=50
CW_FLUSH_SECONDS=2
CW_FALLBACK_PATH=/var/log/laravel/cloudwatch-fallback.log

APP_LOG_SAMPLING_DEBUG=0.10
```

Credenciais AWS

Use um dos mecanismos padrão do AWS SDK:

- **IAM Role / Instance Profile** (recomendado na AWS)
- Variáveis de ambiente:
  - `AWS_ACCESS_KEY_ID`

    - `AWS_SECRET_ACCESS_KEY`
    - `AWS_SESSION_TOKEN` (opcional)
- Arquivo `~/.aws/credentials`
- ECS Task Role / EKS IRSA

---

5) Permissões IAM (CloudWatch Logs)

A identidade usada pela aplicação precisa de permissão para:

- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:PutLogEvents`
- `logs:DescribeLogStreams`
- (opcional) `logs:PutRetentionPolicy`

Exemplo de policy (idealmente escopar para seu log group):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:DescribeLogStreams",
        "logs:PutLogEvents",
        "logs:PutRetentionPolicy"
      ],
      "Resource": "*"
    }
  ]
}
```

---

6) Primeiro teste (smoke test)

A) Teste rápido no Tinker

```
php artisan tinker
```

```
use Egypteam\LaravelAppLogger\Facades\AppLog;

AppLog::info('hello from tinker', ['feature' => 'getting-started']);
```

B) Teste via rota/controller

Crie uma rota:

```
use Egypteam\LaravelAppLogger\Facades\AppLog;

Route::get('/log-smoke', function () {
    AppLog::withContext(['tenant' => 'demo'])
        ->pushContext('smoke')
        ->info('smoke test ok', ['ts' => now()->toIso8601String()]);

    return ['ok' => true];
});
```

Chame:

```
curl -s http://localhost:8000/log-smoke
```

---

7) Validar no AWS CloudWatch

1. Console AWS → **CloudWatch** → **Logs** → **Log groups**
2. Abra seu grupo `CW_LOG_GROUP`
3. Abra seu stream `CW_LOG_STREAM`

Você deve ver linhas JSON.

Opcional: query no CloudWatch Logs Insights

Em **Logs Insights**, selecione o grupo e rode:

```
fields @timestamp, @message
| sort @timestamp desc
| limit 20
```

Para extrair campos do JSON:

```
fields @timestamp, @message
| parse @message /"level":"(?<level>[^"]+)"/
| parse @message /"message":"(?<msg>[^"]+)"/
| filter level = "ERROR"
| sort @timestamp desc
| limit 50
```

---

8) Padrões recomendados (fim a fim)

8.1 Middleware de Request ID (recomendado)

O formatter lê `X-Request-Id`. Se você não usa isso ainda, adicione.

`app/Http/Middleware/RequestId.php`

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

final class RequestId
{
    public function handle(Request $request, Closure $next): Response
    {
        $id = $request->headers->get('X-Request-Id') ?: (string)
\Illuminate\Support\Str::uuid();
        $request->headers->set('X-Request-Id', $id);

        /** @var Response $response */
        $response = $next($request);
        $response->headers->set('X-Request-Id', $id);

        return $response;
    }
}
```

Registre no `app/Http/Kernel.php`.

8.2 Contexto global (tenant/usuário) — exemplo

Se você possui resolução de tenant, pode setar MDC no início do request:

```php
use Egypteam\LaravelAppLogger\Facades\AppLog;

AppLog::withContext([
  'tenant'  => $tenantId,
  'feature' => 'billing',
])->info('request started');
```

> Dica: o `AppLogger` é “imutável” (via clone). Isso facilita criar instâncias “escopadas” por request/operação.

---

9) Exemplos completos

9.1 Exemplo de Controller (MDC + NDC)

```php
use Egypteam\LaravelAppLogger\Facades\AppLog;

final class OrderController
{
    public function store()
    {
        $log = AppLog::withContext(['tenant' => 'acme', 'module' => 'orders'])
            ->pushContext('checkout');

        $log->info('order.store.start');

        // ... sua lógica
        $orderId = 123;

        $log->info('order.store.success', ['order_id' => $orderId]);

        return response()->json(['order_id' => $orderId]);
    }
}
```

9.2 Exemplo com Job / Queue

```php
use Egypteam\LaravelAppLogger\Facades\AppLog;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;

final class ProcessPayment implements ShouldQueue
{
    use Queueable;

    public function __construct(private int $orderId) {}

    public function handle(): void
    {
        AppLog::withContext(['module' => 'payments', 'order_id' => $this->orderId])
            ->pushContext('queue')
            ->info('payment.process.start');

        // ...
        AppLog::info('payment.process.done', ['status' => 'ok']);
    }
}
```

9.3 Exemplo de report global de exceções

No `app/Exceptions/Handler.php`:

```php
use Egypteam\LaravelAppLogger\Facades\AppLog;
use Throwable;

public function report(Throwable $e): void
{
    AppLog::error('exception.reported', [
        'exception' => get_class($e),
        'message' => $e->getMessage(),
        'code' => $e->getCode(),
    ]);

    parent::report($e);
}
```

9.4 Exemplo de auditoria

```php
AppLog::withContext(['tenant' => 'acme', 'audit' => true])
    ->info('user.password.changed', ['user_id' => 42]);
```

---

10) Desenvolvimento local e fallback

Se o CloudWatch não estiver acessível localmente, habilite fallback:

```
CW_FALLBACK_PATH=/tmp/cloudwatch-fallback.log
```

O handler grava em arquivo após as tentativas de retry falharem.

---

11) Testando a integração

Unit test com AWS client mockado

Faça bind de um `CloudWatchLogsClient` mockado no container. O `HandlerFactory` do pacote prefere o client já bindado.

Exemplo (PHPUnit + TestCase Laravel):

```
use Aws\CloudWatchLogs\CloudWatchLogsClient;
use Aws\Result;
use Illuminate\Support\Facades\Log;

public function test_app_log_channel_sends_to_cloudwatch(): void
{
    $mock = $this->createMock(CloudWatchLogsClient::class);

    $mock->method('createLogGroup')->willReturn(new Result([]));
    $mock->method('createLogStream')->willReturn(new Result([]));
    $mock->method('describeLogStreams')->willReturn(new Result([
        'logStreams' => [['logStreamName' => 'web', 'uploadSequenceToken' =>
null]],
    ]));

    $mock->expects($this->once())
        ->method('putLogEvents')
        ->willReturn(new Result(['nextSequenceToken' => 'tok-1']));

    $this->app->instance(CloudWatchLogsClient::class, $mock);

    Log::channel('app_log')->info('test message');
}
```

---

12) Troubleshooting

“Não aparecem logs no CloudWatch”

- Verifique `AWS_REGION`, `CW_LOG_GROUP`, `CW_LOG_STREAM`
- Verifique permissões IAM
- Verifique se sua app está usando `Log::channel('app_log')` ou `AppLog::*`
- Confirme que as credenciais AWS estão disponíveis no runtime

“Logs caem no fallback, mas não no CloudWatch”

- Geralmente é IAM ou conectividade
- Ajuste batching: `CW_BATCH_SIZE`, `CW_FLUSH_SECONDS`
- Verifique throttling no CloudWatch

“Não vejo request id / user id”

- Garanta middleware de `X-Request-Id`
- Garanta autenticação configurada

---

13) Próximos passos

- Criar provider `datadog` ou `loki` no `HandlerFactory`
- Expor sampling por nível via `.env` (opcional)
- Criar middleware dedicado para anexar tenant/usuário automaticamente

---

Referência rápida

Métodos

- `AppLog::trace($msg, $ctx = [])`
- `AppLog::debug(...)`
- `AppLog::info(...)`
- `AppLog::warn(...)`
- `AppLog::error(...)`
- `AppLog::fatal(...)`

Contexto

- `AppLog::withContext(['k' => 'v'])` → MDC
- `AppLog::pushContext('scope')` → NDC
- `AppLog::popContext()`