Getting Started — Egypteam Laravel App Logger

Last updated: 2025-12-26

This guide walks you through an **end-to-end integration** of `egypteam/laravel-app-logger` into a Laravel application:

- install the package
- configure the `app_log` channel
- set environment variables (CloudWatch)
- add useful middleware/context patterns
- verify logs in AWS CloudWatch Logs (and Logs Insights)
- run locally with fallback
- add tests and troubleshooting tips

> Assumptions: PHP 8.2+, Laravel 10/11/12, Monolog 3. The default provider is **CloudWatch**.

---

1) Install the package

Option A — Packagist (recommended)

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

Option B — Install from a Git repository

If you are publishing the package from a private repo, add a VCS repository entry to your app `composer.json`:

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

Then install:

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

Option C — Local path (development)

If you’re developing the package locally alongside your app:

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

Then:

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

---

2) Publish the configuration (optional but recommended)

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

This will create:

- `config/app-logger.php`

If you prefer to keep config centralized in `.env`, you can skip publishing.

---

3) Add the `app_log` channel in `config/logging.php`

Add this channel to your app:

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

Keep your default channel unchanged. You can also route Laravel's default logs to CloudWatch separately if you want, but the goal here is a **dedicated, structured application channel**.

---

4) Configure environment variables (`.env`)

Minimum configuration (CloudWatch)

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

Recommended production configuration

```
# Provider
APP_LOG_PROVIDER=cloudwatch

# 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

# Sampling (debug)
APP_LOG_SAMPLING_DEBUG=0.10
```

AWS credentials

Use one of the standard AWS SDK mechanisms:

- **IAM Role / Instance Profile** (recommended on AWS)
- Env variables in your runtime:
  - `AWS_ACCESS_KEY_ID`
  - `AWS_SECRET_ACCESS_KEY`
  - `AWS_SESSION_TOKEN` (optional)
- Shared credentials file (`~/.aws/credentials`)
- ECS task role / EKS IRSA

---

5) IAM permissions (CloudWatch Logs)

Your runtime principal needs permission to:

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

Example policy (scope to your log group if possible):

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

---

6) First log smoke test

A) Quick interactive test (Tinker)

```
php artisan tinker
```

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

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

B) Quick route/controller test

Create a route:

```
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];
});
```

Hit it:

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

---

7) Verify logs in AWS CloudWatch

1. AWS Console → **CloudWatch** → **Logs** → **Log groups**
2. Open your group `CW_LOG_GROUP`
3. Open your stream `CW_LOG_STREAM`

You should see JSON lines.

Optional: CloudWatch Logs Insights query

In **Logs Insights**, select your group and run:

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

To parse JSON fields:

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

8) End-to-end patterns (recommended)

8.1 Add a Request ID middleware (recommended)

The formatter reads `X-Request-Id`. If you don’t already have one, create it.

`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;
    }
}
```

Register it in `app/Http/Kernel.php` (e.g., in `$middleware` or a group).

8.2 Add tenant/user context globally (example)

If you have a tenant resolver, you can set MDC for the request:

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

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

> Tip: the package's `AppLogger` is immutable via cloning, so you can create scoped logger instances per request/operation.

---

9) Full examples

9.1 Controller example (with MDC + NDC)

```
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');

        // ... your logic
        $orderId = 123;

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

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

9.2 Job / Queue example

```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 Exception reporting example (global)

In `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 Audit-style event logging (domain events)

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

---

10) Local development and fallback

If CloudWatch isn’t reachable locally, you can still keep logs by enabling fallback:

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

The handler will write failed batches to that file after retries.

---

11) Testing your integration

Unit test with a mocked AWS client

Bind a mock `CloudWatchLogsClient` to the container. The package’s `HandlerFactory` will prefer the bound instance.

Example (PHPUnit + Laravel test case):

```
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

“No logs appear in CloudWatch”

- Check `AWS_REGION`, `CW_LOG_GROUP`, `CW_LOG_STREAM`
- Check IAM permissions
- Check your app is actually using `Log::channel('app_log')` or `AppLog::*`
- Confirm runtime credentials are present (ECS task role, IRSA, env vars, etc.)

“Logs appear in fallback file but not CloudWatch”

- Usually IAM or connectivity
- Check CloudWatch throttling (enable batching: `CW_BATCH_SIZE`, `CW_FLUSH_SECONDS`)

“I don’t see request id / user id”

- Ensure middleware sets `X-Request-Id`

- Ensure auth is configured for your requests

---

13) Next steps

- Add provider `datadog` or `loki` in `HandlerFactory`
- Add per-level sampling in config via env vars if desired
- Add a dedicated middleware to attach tenant/user context automatically

---

Quick reference

Log methods

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

Context

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