GUL'S NOTEBOOK · ENGINEERING · OPEN ACCESS

Setting up laravel-agent-guard: budgets, loops, kill switch

Muhammad Gulzaib
https://orcid.org/0009-0000-8709-6989
SynergyFlow Labs · Islamabad, Pakistan

Published	2026-11-26
URL	https://gulzaib.vercel.app/blog/setting-up-laravel-agent-guard-budgets-loops-kill-switch
Licence	Creative Commons Attribution 4.0 International (CC BY 4.0), https://creativecommons.org/licenses/by/4.0/

ABSTRACT

Wire laravel-agent-guard into a laravel/ai agent: tenant budgets, loop detection, the kill switch and the gotchas I found running v1.2.0 step by step.

Keywords: laravel, php, ai, agents, open-source, multi-tenancy

> **TL;DR**
>
> Add `GuardAgentSteps` to your agent's `middleware()` method, pass the tenant ID in explicitly, set `run` and `tenant` budgets, and call `enforce()` yourself, because the env setting alone doesn't enforce anything. Before you trust the dollar limits, check that your model is in the pricing catalog. Then watch the events in observe mode for a week. Numeric tool arguments and a short output estimate are the two things most likely to surprise you.

This is the developer follow-up to why I built laravel-agent-guard. That post covered the problem. This one covers setup, how each check behaves and the parts that tripped me up.

Everything here is checked against the v1.2.0 code [1]. I didn't want to describe the package from memory, so I ran the middleware step by step against the real `PendingStep`, `StepResult` and `StepResponse` classes from laravel/ai. The step outputs quoted below come from those runs.

> **Use v1.2.0 or later**
>
> Earlier 1.x releases only check the kill switch for the current run, so `agent-guard:kill --tenant` and `--global` don't stop an agent that is already running. v1.2.0 checks the global, tenant and run switches on every step.

Install it, then fix the pricing path

You need PHP 8.3, Laravel 12 or 13, `laravel/ai` ^1.0 and `laravel/mcp` ^1.0. Use Redis if you can. The database driver works, but the package doesn't publish its migration, so you have to copy `create_agent_guard_budgets_table.php` from the package yourself.

```
composer require imgul/laravel-agent-guard
php artisan vendor:publish --tag=agent-guard-config
```

After publishing, open `config/agent-guard.php` and change `pricing.catalog`. The default path is built from `__DIR__`. Once the file moves into your app's `config/` folder, that path points at a `src/` folder your app doesn't have, and the first guarded step fails with "Pricing catalog not found". Copy the JSON into your app and point to the copy, since you'll edit prices in it anyway:

`config/agent-guard.php`

```
'pricing' => [
    // cp vendor/imgul/laravel-agent-guard/src/Support/Pricing/prices.json
    //    resources/agent-guard/prices.json
    'catalog' => resource_path('agent-guard/prices.json'),
],
```

Attach the guard to an agent

laravel/ai calls `middleware()` on every agent that implements `HasMiddleware` and runs the result around each step [2]. It calls `middleware()` once per prompt, so every prompt gets a fresh guard with a new run ID.

`app/Ai/Agents/FeeAssistant.php`

```
#[Model('claude-sonnet-4-5')]
class FeeAssistant implements Agent, HasMiddleware, HasTools
{
    use Promptable;

    public function __construct(public School $school) {}

    // instructions() and tools() as usual

    public function middleware(): array
    {
        $guard = GuardAgentSteps::make()
            ->forTenant(fn () => $this->school->id)
            ->budget([
                'run' => [0.25, 0.50],       // soft, hard (USD)
```

```
                'tenant' => [15.00, 20.00],  // per school, per day
            ])
            ->downshiftAt(0.80, toModel: 'claude-haiku-4-5')
            ->breakLoops(maxRepeats: 3, frequency: 8);

        return [
            config('agent-guard.mode') === 'enforce'
                ? $guard->enforce()
                : $guard->observe(),
        ];
    }
}
```

Pass the tenant in through the constructor. The README example uses `fn () => Auth::id()`, which returns `null` inside a queued job. When the resolver returns `null` or throws, the guard treats the run as having no tenant and quietly skips the tenant budget. Integer IDs and `Stringable` values are fine.

Observe mode is the default, even with the env flag set

`AGENT_GUARD_MODE=enforce` sets the runtime default, but `GuardAgentSteps` keeps its own mode, and that mode is observe until you call `enforce()`. Observe mode still reserves budget and runs every detector. It fires events and changes nothing, apart from honouring an open kill switch. That's why the example above reads the config value itself.

Leave it in observe mode for a week and log what it would have done:

`app/Providers/AppServiceProvider.php`

```
Event::listen(function (LoopDetected $e) {
    Log::warning('agent-guard loop', [
        'detector' => $e->detector,
        'tool' => $e->tool,
        'evidence' => $e->evidence,
    ]);
});
```

Event	Useful fields
`LoopDetected`	`detector`, `tool`, `fingerprint`, `evidence`
`BudgetSoftCapReached`	`scope`, `key`, `spent`, `cap`, `estimated`
`BudgetHardCapReached`	`scope`, `key`, `spent`, `cap`
`ModelDownshifted`	`fromModel`, `toModel`, `tier`, `reason`
`CircuitTripped`	`key` (the run ID), `cooldownSeconds`

`KillSwitchActivated`	`scope`, `key`, `opened`

A budget is reserved before the step and corrected after it

Before each step, the guard estimates the cost and reserves it. Input tokens are the message length divided by four. Output is a flat `estimate.output_tokens`, 256 by default. With the Redis driver, the reservation is one Lua script: read the spend, refuse if spend plus cost would pass the hard cap, otherwise `INCRBYFLOAT` and `EXPIRE`. Redis runs a script as one atomic operation [3], so ten workers can't all squeeze through the same last gap. After the step, the guard swaps the estimate for the real usage the provider reported.

Scope	Redis key	Resets
`run`	`agent-guard:budget:run:usd:{runId}`	24 hours after the last write
`tenant`	`…:tenant:usd:{tenantId}:{Y-m-d}`	midnight
`agent`	`…:agent:usd:agent`	midnight

Two details from that table matter. The `agent` key has no agent name in it, so it's one daily counter shared by every guarded agent in your app. "Midnight" uses your `app.timezone`. With the default UTC, a school in Pakistan gets its daily budget back at 5 a.m. local time. The `tool` scope in the middleware writes to the same key as `agent`, so use the MCP trait (below) for real per-tool limits.

The hard cap can be passed by one step

The hard-cap check compares against the estimate, and the estimate leaves out the system prompt and tool definitions. When a step costs more than estimated, it still runs. The next step is refused. In my run with a `0.50 hard cap and no downshift, spend reached` 0.525 after step 4, and step 5 threw `BudgetExceededException`. Raise `estimate.output_tokens` closer to your real output size, and catch the exception where you call the agent:

`app/Http/Controllers/FeeChatController.php`

```
try {
    $reply = (new FeeAssistant($school))->prompt($question);
} catch (BudgetExceededException $e) {
    return back()->with('status', 'Today's AI limit is used up.');
}
```

Check that your model has a price

The catalog that ships with v1.2.0 prices six models: `gpt-5`, `gpt-5-mini`, `claude-opus-4-1`, `claude-sonnet-4-5`, `claude-haiku-4-5` and `gemini-3-pro`. The lookup is an exact match. Any

other model is costed at $0, so USD caps never fill up and no event fires.

I ran six steps on `claude-opus-4-5` with USD caps and the run ended at $0.0000 spent. The same run with a token budget counted 41,700 tokens. If you can't keep the catalog current, budget in tokens: `'run' => [20000, 40000, 'tokens']`.

To price a missing model, add it to your copied `prices.json` in dollars per million tokens, or override it at boot. Opus 4.5 is `5 input and` 25 output per million tokens [4]:

`app/Providers/AppServiceProvider.php`

```
$this->app->make(PricingCatalog::class)->override(
    'claude-opus-4-5',
    ['input' => 5.0, 'output' => 25.0],
);
```

What the loop detector counts as a repeat

After each step, the guard checks the tool calls in the response. Detectors run in this order and the first match wins: exact repeat, normalised repeat, frequency, then context growth. Semantic similarity only runs once the frequency detector has already fired.

Normalising replaces pagination keys (`page`, `offset`, `limit`, `cursor`, `per_page`, timestamps) and every integer or float with `*`, and collapses whitespace. I fed these calls straight into `LoopDetector`:

Calls	Result
`FindInvoice` with `id` 101, 102, 103	normalised repeat on call 3
`FindInvoice` with `id` "INV-101", "INV-102", "INV-103"	nothing
`SearchStudents` "fee defaulters", pages 1, 2, 3	normalised repeat on call 3
`SearchStudents` with 9 different names	frequency on call 8

The first row is the one to watch. A tool that fetches records by numeric ID looks like a loop on its third call. Exempt it with `->exemptTools(['FindInvoice'])`, using the tool's class name or whatever its `name()` method returns. The step limit and the budgets still cover exempt tools.

Where the run actually stops

The detector runs after the step, so the tool calls from the step that tripped it still execute. The breaker stops the next step. With paged searches, steps 0 to 2 ran and steps 3 to 5 came back `STOPPED`. laravel/ai ends the loop there, and you get whatever earlier steps produced. A streamed response ends with the `cancelled` finish reason. The benchmark that ships with the package (`php benchmarks/loop-budget-bench.php`) shows the same shape over 5 steps:

11,000 tokens and `0.26 without the guard, 5,700 tokens and` 0.05 with it. The breaker trips at step 3, which saves 5,300 tokens.

Downshift reads the first budget, and has a bug

The downshift threshold is a fraction of the soft cap of the first scope you set, checked in the order `run`, `agent`, `tenant`. With `'run' => [0.25, 0.50]` and `downshiftAt(0.80, …)`, it fires once the run has spent $0.20, including the current step's estimate. A tier can also drop tools with `withoutTools` and lower `withMaxTokens`. Chained tiers must go in rising order of threshold.

> **In v1.2.0 a downshift lasts one step**
>
> In my run, step 1 went to Haiku, and steps 2 to 5 went back to Opus while `BudgetSoftCapReached` kept firing. laravel/ai builds every step with the agent's original model, and the guard skips a tier that "wouldn't change" the model it already switched to. Later steps are also estimated at Haiku prices, which makes the hard-cap overshoot bigger. Until a fix ships, treat the hard cap as your real limit.

Kill switches and Artisan commands

```
php artisan agent-guard:status acme
php artisan agent-guard:budgets acme
php artisan agent-guard:kill --tenant=acme
php artisan agent-guard:kill --run=d72ce9720d8856bc
php artisan agent-guard:kill --global
```

A kill switch has no expiry. It stays open until you release it. `AgentGuard::unkill($tenantId)` releases a tenant. There's no facade method for the global switch yet, so release that one in Tinker with `app(GuardRuntime::class)->killSwitch->unkill('global', KillSwitch::GLOBAL_KEY)`. Run IDs show up as `key` on `CircuitTripped` and at the end of each run budget key, so log them if you want `--run` to be useful.

The same guard for MCP tools

`GuardedMcpTool` owns `handle()` and calls your `run()`. It has the same observe default, so turn enforcement on in the constructor:

`app/Mcp/Tools/QueryFees.php`

```
class QueryFees extends Tool
{
    use GuardedMcpTool;

    public function __construct()
```

```
    {
        $this->guardEnforce();
    }

    protected function guardBudget(): array
    {
        return ['tool_daily_calls' => 200, 'max_repeats' => 3];
    }

    public function run(Request $request): Response
    {
        // your tool logic
    }
}
```

Limits are tracked per tool and per principal: the authenticated user ID by default, or whatever `forPrincipal()` returns. The MCP session ID is never used, so stateless MCP servers work. In v1.2.0 the MCP trait only checks its own tool-scope switch, so `--global` doesn't stop MCP tools yet.

Test it without Redis

`AgentGuard::fake()` swaps in an in-memory runtime and records every decision:

`tests/Feature/FeeAssistantTest.php`

```
AgentGuard::fake();

// run the guarded agent or MCP tool

AgentGuard::assertLoopDetected(tool: 'SearchStudents');
AgentGuard::assertBudgetSoftCapReached(scope: 'run');
AgentGuard::assertNothingDenied();
```

The fake prices only `gpt-5` and `gpt-5-mini`, so use those model names when a test depends on USD budgets.

If you find a loop it missed or a price that went stale, open an issue or a pull request on GitHub. It's MIT-licensed, and I kept it open so fixes like these reach everyone at once.

Key takeaways

- Call `enforce()` on the middleware yourself. `AGENT_GUARD_MODE=enforce` alone leaves `GuardAgentSteps` in observe mode.
- Pass the tenant ID through the agent's constructor, because `Auth::id()` is `null` in queued jobs and the tenant budget is skipped without a warning.
- Point `pricing.catalog` at a copy inside your app after publishing, and make sure your

> model is in it. Otherwise use token budgets.
> - Exempt tools that take numeric IDs, since every number normalises to the same value.
> - Treat the hard cap as the limit that holds. It can be passed by one step, and in v1.2.0 a downshift only lasts one step.

Questions readers ask

Does AGENT_GUARD_MODE=enforce turn on enforcement for my agents?

Not by itself. The step middleware has its own mode and stays in observe mode until you call enforce() on it. Read the config value in your agent's middleware() method and call enforce() or observe() from there.

Why does my USD budget never trigger?

The model is probably missing from the pricing catalog. An unpriced model is costed at $0, so USD caps never fill up. Add the model's prices to the catalog, or use a token budget, which works for every model.

Why did laravel-agent-guard flag a normal tool as a loop?

Loop detection treats every integer and float argument as the same value. Three lookups with three different numeric IDs look like three identical calls. Add that tool to exemptTools().

Can laravel-agent-guard enforce budgets across several queue workers?

Yes. With the Redis driver, each step reserves its estimated cost inside one Lua script, so two workers can't both pass the same hard-cap check at the same time.

References

1. Muhammad Gulzaib. laravel-agent-guard source code (GitHub), 2026.
2. Laravel. Laravel AI SDK (Laravel 13.x documentation), 2026.
3. Redis. Scripting with Lua, 2026.
4. Anthropic. Claude API pricing, 2026.

CITE AS

Gulzaib, M. (2026, November 26). Setting up laravel-agent-guard: budgets, loops, kill switch. Gul's Notebook. https://gulzaib.vercel.app/blog/setting-up-laravel-agent-guard-budgets-loops-kill-switch

You may share and adapt this text for any purpose, including commercially, as long as you credit the author and link to the original. Licence: https://creativecommons.org/licenses/by/4.0/