php-sentinel

API Contract Monitor for PHP

Full Project Specification Document

Version	1.0.0 — Initial Release
Date	February 2026
Build Target	7–8 Weeks (Senior Solo Developer)
PHP Requirement	PHP 8.3+ | Zero required dependencies
Packagist	php-sentinel/sentinel
License	MIT

This document is the complete technical specification for php-sentinel — a passive API contract monitor that detects when third-party APIs silently change their response structure. It covers architecture, feature design, implementation plan, API surface, testing strategy, and release roadmap.

1. Executive Summary

PHP Sentinel is a pure PHP 8.3 library that sits between your application and any PSR-18 HTTP client, silently profiling the JSON structure of every external API response. After building a probabilistic baseline schema across multiple real responses, it detects when a third-party API changes — a renamed field, a type flip from string to integer, a once-required key that now sometimes disappears — and surfaces that change as a structured, actionable event before your production code breaks.

The problem it solves is not theoretical. Every developer who has consumed a Shopify, Stripe, HubSpot, QuickBooks, or WooCommerce API has experienced silent API drift: a third party silently renames a field, changes a data type, or restructures a response in a minor version update without announcing it. The developer finds out via a production error, a client complaint, or a failed cron job at 3 AM.

> *Core value proposition: php-sentinel is the difference between "our Shopify integration silently broke at 2 AM" and "we got a Slack alert 6 hours ago that Shopify changed order.total to order.current_total_price."*

2. Problem Statement

2.1 The Silent Drift Problem

Third-party API providers — Shopify, Stripe, HubSpot, QuickBooks, WooCommerce REST, payment gateways, logistics APIs — change their response structures regularly. Unlike breaking changes to an API's contract version, these are often:

- Field renames between minor API versions (order.total → order.current_total_price)
- Type coercions — numeric values returned as strings in one version, integers in another
- Previously guaranteed fields becoming nullable or conditional
- Nested object structures being flattened or deepened
- Enum values expanding or changing case (ACTIVE → active)
- New required fields added to request payloads the API now enforces

None of these changes typically trigger a version bump. They are discovered by developers in production, under client pressure, with no warning.

2.2 Why Existing Tools Fail

Tool / Approach	What It Does	Why It Fails Here
Pact / Consumer-Driven Contracts	Both sides write and verify contracts	Shopify will never run Pact against your contract
OpenAPI Contract Testing	Validates against a spec file you maintain	Requires you to maintain spec as third party changes

Tool / Approach	What It Does	Why It Fails Here
php-vcr / HTTP Record & Replay	Records HTTP responses for test replay	Detects nothing; replays stale responses
APM / Datadog / New Relic	Infrastructure-level monitoring	HTTP 200 ≠ correct data; no field-level awareness
Manual Schema Tests	Assertions written by developer	Covers what you thought to test; misses the unknown

3. System Architecture

3.1 Design Principles

- Zero dependencies — pure PHP 8.3. No Guzzle, no Symfony, no Laravel required
- PSR-18 middleware — works with any compliant HTTP client (Guzzle, Symfony HttpClient, etc.)
- PSR-14 events — dispatch drift events into any framework event system
- PSR-3 logging — plug into any PSR-3 logger for schema change logs
- Non-invasive — the calling code is completely unaware of monitoring
- Probabilistic, not brittle — builds confidence before alerting to prevent false positives

3.2 Component Map

Component	Responsibility
SchemaWatcher (Middleware)	PSR-18 middleware. Intercepts responses, routes payload to inference engine, passes response through unchanged.
InferenceEngine	Walks a decoded JSON structure and produces a formal JSON Schema Draft 7 object. Handles nesting, arrays, nullable detection, and enum candidate flagging.
SampleAccumulator	Collects N response samples before hardening a schema. Computes nullability and optionality probabilities across the sample window.
SchemaStore (Interface)	Persists and retrieves schemas by endpoint key. Ships with File, PDO, and Redis drivers.
EndpointNormalizer	Strips variable segments from URLs to produce stable schema keys. GET /orders/12345 → GET /orders/{id}.
DriftDetector	Performs structural diff between persisted schema and new inferred schema. Produces typed change objects with severity classification.
DriftReporter	Assembles a SchemaDrift value object from DriftDetector output and dispatches PSR-14 SchemaDriftDetected event.
CLI Profiler	Standalone binary. Commands: profile, diff, list, inspect. Zero

Component	Responsibility
	framework dependency.

3.3 Request Flow

The middleware intercepts every HTTP response transparently. The calling code never changes.

```
HTTP Request → SchemaWatcher (PSR-18 Middleware) → Original HTTP Client → Third-
Party API                          ↓ (response passes through unchanged)
Response returned to caller                              ↓
EndpointNormalizer → schema key derivation                                ↓
InferenceEngine → JSON Schema from payload                                ↓
SampleAccumulator → is baseline stable?                          ↓ yes          ↓ no
(still sampling)               DriftDetector     Store sample, return
↓               SchemaDrift value object                                  ↓
PSR-14 EventDispatcher → SchemaDriftDetected                                  ↓
Developer listener: Slack / Log / DB / Alert
```

4. Feature Specifications

4.1 PSR-18 Middleware Integration

The SchemaWatcher class implements the PSR-18 middleware pattern. It wraps an inner PSR-18 client and intercepts all responses. The caller's code changes by exactly one line: the HTTP client construction.

```
use Sentinel\Middleware\SchemaWatcher; use Sentinel\Sentinel;  $sentinel =
Sentinel::create()     ->withStore(new
FileSchemaStore(__DIR__.'/sentinel/schemas/'))     ->withSampleThreshold(20)
// collect 20 samples before hardening     ->withDriftSeverity(Severity::BREAKING)
// only alert on breaking changes     ->build();  $stack = HandlerStack::create();
$stack->push(new SchemaWatcher($sentinel));  $client = new GuzzleClient(['handler'
=> $stack]);  // From this point: every response is silently profiled. Zero
changes to call sites. $response = $client-
>get('https://myshopify.com/admin/api/2024-01/products.json');
```

Supported HTTP clients via PSR-18 adapter:

- Guzzle 7+ (via GuzzleHttp\Client)
- Symfony HttpClient (via symfony/http-client PSR-18 adapter)
- Any PSR-18 compliant client

4.2 JSON Schema Inference Engine

The InferenceEngine is the core intellectual component. Given a decoded JSON payload, it walks the structure recursively and produces a formal JSON Schema Draft 7 object that describes every field, type, nesting depth, array structure, and nullability.

Type Inference Rules

PHP Value	Schema Type	Notes
"42"	string	Numeric strings flagged as coercion candidates
42	integer	Type coercion alert if same field was previously string
42.5	number	Distinct from integer; tracks precision changes
true / false	boolean	
null	null	Combined with observed type as nullable union
[ ]	array	Item schema inferred from all array elements
{ }	object	Recurse into properties map

Advanced Inference Capabilities

- Enum detection — string fields returning ≤8 distinct values across ≥30 samples are flagged as enum candidates with the observed value set recorded
- Nullable vs optional — null on a field ≠ the field being absent. Sentinel tracks both separately: a field present as null is nullable; a field absent entirely is optional
- Array homogeneity — arrays with inconsistent item types across samples generate a oneOf item schema and a heterogeneous array warning
- Deep nesting — unlimited recursion depth with cycle detection for any pathological edge cases
- ISO 8601 and UUID detection — string fields consistently matching date or UUID patterns are annotated with format hints for developer awareness

4.3 Probabilistic Sample Accumulation

A single API response is not enough to know the true shape of that API. A fresh Shopify order may have a different structure from one with partial fulfillment, multiple line items, or a discount code applied. Sentinel does not harden a schema until it has observed N responses from the same endpoint key.

Sample Lifecycle States

State	Trigger	Behaviour
SAMPLING	Response count < threshold	Responses accumulated. No drift detection. No alerts.
HARDENING	Count = threshold	Final schema computed from all samples. Nullability and optionality probabilities frozen.

State	Trigger	Behaviour
MONITORING	Schema hardened	Every subsequent response is compared against hardened schema. Drift alerts active.
DRIFTED	Breaking drift detected	Previous schema archived. New sampling window begins automatically for re-hardening.

Configuration:

```
Sentinel::create()     ->withSampleThreshold(25)          // responses before
hardening (default: 20)     ->withAdditiveThreshold(0.95)      // 95% of samples
must include a field for it to be 'required'     ->withReharden(true)
// automatically start new sample window after drift     -
>withMaxStoredSamples(50)         // keep up to 50 raw samples for manual review
->build();
```

4.4 Endpoint URL Normalization

Without normalization, GET /orders/12345 and GET /orders/99999 would create two separate schema keys despite being the same endpoint. The EndpointNormalizer prevents this, keeping the schema store from growing indefinitely.

Normalization Rules

- UUID segments — any path segment matching a UUID pattern is replaced with {uuid}
- Numeric IDs — pure numeric path segments replaced with {id}
- Hash segments — hex strings ≥8 characters replaced with {hash}
- Query parameters — stripped from the key by default; configurable to include whitelisted params
- Custom patterns — developer-supplied regex replacements for domain-specific ID formats

```
// Default normalization examples: GET /orders/12345                          →
GET /orders/{id} GET /products/a1b2c3d4-e5f6-7890-abcd-ef0123456789  →  GET
/products/{uuid} GET /admin/api/2024-01/products.json?limit=250       →  GET
/admin/api/2024-01/products.json  // Custom normalizer: $normalizer = new
EndpointNormalizer(); $normalizer->addPattern('/\/shop\/[a-z0-9-
]+\.myshopify\.com/', '/shop/{shop}');
```

4.5 Drift Detection and Severity Classification

The DriftDetector performs a recursive structural diff between the hardened schema and the newly inferred schema. Every change is typed and classified by severity.

Change Types and Severity

Change Type	Severity	Description

Change Type	Severity	Description
FieldRemoved	BREAKING	A field that was present in the hardened schema no longer appears in responses
TypeChanged	BREAKING	A field's type has changed (string → integer, object → array, etc.)
RequiredNowOptional	BREAKING	A field that was always present is now sometimes absent
NowNullable	BREAKING	A previously non-null field now returns null values
EnumValueRemoved	BREAKING	A previously observed enum value no longer appears
FieldAdded	ADDITIVE	A new field not previously in the schema appears in responses
EnumValueAdded	ADDITIVE	A new enum value not previously observed now appears
OptionalNowRequired	ADVISORY	A field that was sometimes absent is now always present (positive but notable)
FormatChanged	ADVISORY	String format annotation changed (e.g., date-time → date)

SchemaDrift Value Object

```
SchemaDrift { endpoint: 'GET /admin/api/2024-01/orders/{id}.json 200', detectedAt: DateTimeImmutable('2026-02-20 03:41:22 UTC'), severity: Severity::BREAKING, changes: [ FieldRemoved { path: 'order.total_price', previousType: 'string' }, FieldAdded { path: 'order.current_total_price', type: 'string' }, TypeChanged { path: 'order.financial_status', from: 'string', to: 'string|null' }, RequiredNowOptional { path: 'order.shipping_address', probability: 0.72 }, EnumValueAdded { path: 'order.fulfillment_status', value: 'partially_fulfilled' }, ], previousSchemaVersion: 'sha256:a1b2c3d4...', newSchemaVersion: 'sha256:e5f6a7b8...', }
```

4.6 Schema Storage — Pluggable Drivers

All schema data is persisted through a SchemaStoreInterface. Three drivers ship with the package. Developers can implement the interface to use any storage backend.

```
interface SchemaStoreInterface { public function has(string $key): bool; public function get(string $key): ?StoredSchema; public function put(string $key, StoredSchema $schema): void; public function getSamples(string $key): SampleCollection; public function addSample(string $key, array $payload): void; public function archive(string $key, StoredSchema $schema): void; public function all(): array; // returns all stored schema keys }
```

File Driver (Default)

Stores schemas as JSON files on disk. Schema files are human-readable and committable to version control — a schema change appears as a git diff in your PR. Ideal for small-to-medium integrations.

PDO Driver

Stores schemas in a relational database (MySQL, PostgreSQL, SQLite). Provides schema version history and sample retention. Includes a migration generator command.

Redis Driver

In-memory storage using Redis hashes and sorted sets. Built for high-frequency APIs (payment gateways, real-time inventory). Schema data is automatically promoted to a PDO or file store after hardening — Redis handles hot sampling, durable storage handles persistence.

4.7 PSR-14 Event Dispatching

Drift notifications are dispatched as PSR-14 events — no framework coupling. Plug into Laravel's event system, Symfony's EventDispatcher, or any PSR-14 compatible dispatcher.

```
// Laravel — in EventServiceProvider: Event::listen(SchemaDriftDetected::class,
function(SchemaDriftDetected $event) {     if ($event->drift->severity ===
Severity::BREAKING) {         Notification::send(
User::role('developer')->get(),             new ApiDriftAlert($event->drift)
);     }     Log::channel('sentinel')->warning('Schema drift', [
'endpoint' => $event->drift->endpoint,         'changes'  => $event->drift-
>changes,     ]); });
```

```
// Symfony — in services.yaml: App\Listener\SentinelDriftListener:     tags:
- { name: kernel.event_listener, event: Sentinel\Events\SchemaDriftDetected }
```

Additional events dispatched by Sentinel:

- SchemaHardened — fired when a baseline schema is finalized from the sample window
- SampleCollected — fired on every response during the sampling phase (for observability)
- SchemaArchived — fired when a drifted schema is archived and re-sampling begins

4.8 CLI Profiler and Diffing Tools

The standalone CLI binary provides four commands. No framework required — runs on bare PHP.

sentinel profile

Calls an endpoint N times and builds a baseline schema from real responses. Use for onboarding existing integrations before deploying the middleware.

```
vendor/bin/sentinel profile \   --url
"https://shop.myshopify.com/admin/api/2024-01/products.json" \   --method  GET \
--header  "X-Shopify-Access-Token: shpat_xxx" \   --samples 50 \   --output
./sentinel/schemas/  # Paginated profile — follows cursor automatically
vendor/bin/sentinel profile \   --url        "https://api.example.com/orders" \
--paginate    cursor \   --cursor-key  next_cursor \   --max-pages   10 \   --
output      ./sentinel/schemas/
```

sentinel diff

Compares two saved schema files and outputs a colorized, field-level diff with severity annotations. Suitable for CI pipelines — exits with code 1 on BREAKING changes.

```
vendor/bin/sentinel diff \   ./sentinel/schemas/shopify_products_baseline.json \
./sentinel/schemas/shopify_products_current.json  # Example output: ✓ ADDITIVE
order.discount_applications[].code       (new field: string)  ✗ BREAKING
order.total_price                        (removed)  ✗ BREAKING
order.financial_status                   (string → string|null)  ⚠ ADVISORY
order.note                               (optional → required)  Exit code: 1
(BREAKING changes detected)
```

sentinel inspect

Shows the full hardened schema for an endpoint key, including sample count, hardening date, version history, and all observed enum values.

sentinel list

Lists all tracked endpoint keys with their current state (sampling / hardened / drifted), sample counts, and last activity timestamp.

4.9 Framework Adapters

Two thin adapter packages provide framework-native integration without adding coupling to the core library.

php-sentinel/laravel

- Service provider auto-registers SchemaWatcher on the Laravel HTTP client via a macro
- Config published via php artisan vendor:publish — sentinel.php config file
- Uses Laravel's Cache, Queue, and Event systems as the PSR backends automatically
- Artisan commands: sentinel:status, sentinel:profile, sentinel:diff, sentinel:reset {endpoint}
- Optional Livewire dashboard behind a viewSentinel gate — lists all tracked endpoints with live sample progress and drift history

php-sentinel/symfony

- Bundle with zero-config autowiring for the HttpClient integration
- Integrates with Symfony's EventDispatcher and Messenger for async drift processing

- Configuration via sentinel.yaml
- Console commands mirroring the standalone CLI

5. Technical Specification

5.1 PHP Requirements and Dependencies

Requirement	Minimum	Notes
PHP	8.3	Uses readonly classes, enum, fibers (for CLI profiler), first-class callables
psr/http-client	^1.0	The only direct Composer dependency
psr/event-dispatcher	^1.0	For SchemaDriftDetected event dispatch
psr/log	^3.0	Optional; pluggable for debug logging
ext-json	any	PHP built-in; no external library needed

Core principle: PSR interfaces are the only dependencies. The library imposes zero HTTP client, framework, or infrastructure choices on the consuming application.

5.2 Directory Structure

```
php-sentinel/ ├── src/ │   ├── Sentinel.php              # Main builder / entry
point │   ├── Middleware/ │   │   └── SchemaWatcher.php      # PSR-18 middleware │
├── Inference/ │   │   ├── InferenceEngine.php    # JSON → JSON Schema │   │   ├──
TypeResolver.php       # PHP value → JSON Schema type │   │   ├──
EnumCandidateDetector.php │   │   └── FormatHintDetector.php # ISO8601, UUID
detection │   ├── Sampling/ │   │   ├── SampleAccumulator.php │   │   └──
SampleCollection.php │   ├── Normalization/ │   │   └── EndpointNormalizer.php │
├── Schema/ │   │   ├── StoredSchema.php       # Value object │   │   ├──
SchemaVersion.php      # Immutable versioned schema snapshot │   │   └──
SchemaStoreInterface.php │   ├── Store/ │   │   ├── FileSchemaStore.php │   │
├── PdoSchemaStore.php │   │   └── RedisSchemaStore.php │   ├── Drift/ │   │   ├──
DriftDetector.php │   │   ├── DriftReporter.php │   │   ├── SchemaDrift.php
# Value object │   │   ├── Severity.php           # Enum: BREAKING / ADDITIVE /
ADVISORY │   │   └── Changes/              # FieldRemoved, TypeChanged, etc. │
├── Events/ │   │   ├── SchemaDriftDetected.php │   │   ├── SchemaHardened.php │
│   └── SampleCollected.php │   └── Console/ │       ├── SentinelApplication.php │
└── Commands/              # profile, diff, list, inspect ├── tests/ │   ├── Unit/
│   └── Integration/ ├── bin/ │   └── sentinel                  # CLI entry point
├── composer.json ├── README.md └── CHANGELOG.md
```

6. Testing Strategy

6.1 Test Philosophy

Every component is tested in strict isolation. Zero live API calls in the test suite — all HTTP responses are provided as PHP arrays or raw JSON fixtures. The inference engine, drift detector, normalizer, and accumulator all operate on plain data structures and are trivially unit-testable.

6.2 Coverage Targets

Component	Target Coverage	Testing Approach
InferenceEngine	100%	Pure function — input payload in, schema out. No mocking needed.
DriftDetector	100%	Parameterized tests: before schema + after schema → expected drift changes
EndpointNormalizer	100%	Dataset of raw URLs → expected normalized keys
SampleAccumulator	95%	State machine transitions tested through sample threshold boundaries
SchemaWatcher middleware	95%	PSR-18 mock client; verify response pass-through is unchanged
Store drivers (File/PDO/Redis)	90%	Integration tests with SQLite in-memory; Redis via Mockery
CLI Commands	85%	Fixture JSON responses; assert exit codes and output format

6.3 Key Test Scenarios

- InferenceEngine correctly identifies nullable vs optional for deeply nested fields
- DriftDetector raises BREAKING on FieldRemoved but not on FieldAdded
- SampleAccumulator transitions from SAMPLING → HARDENING at exact threshold N
- EndpointNormalizer handles Shopify shop-domain URLs, UUIDs, numeric IDs, and custom patterns
- SchemaWatcher returns original response body byte-for-byte unchanged after profiling
- Full integration test: middleware → real JSON fixture → drift detected → PSR-14 event dispatched
- Real-world fixture tests using saved Shopify and Stripe API response samples

7. Implementation Timeline

Week	Focus Area	Deliverables
1	Foundation	Repository setup, composer.json, PSR interfaces resolved, Sentinel builder class skeleton, FileSchemaStore working, Pest configured

Week	Focus Area	Deliverables
2	Inference Engine	Complete InferenceEngine with all type rules, nullable/optional tracking, array item schemas, enum detection, ISO8601/UUID format hints. 100% unit test coverage.
3	Sampling + Normalization	SampleAccumulator with state machine, sample threshold, hardening logic. EndpointNormalizer with UUID/numeric/hash patterns and custom regex support.
4	Drift Detection	DriftDetector with all 9 change types, severity classification, SchemaDrift value object, PSR-14 event dispatch. Full parameterized test suite.
5	Storage Drivers	PDO driver with migration generator, Redis driver with hot/cold promotion strategy. Schema versioning and archive support across all drivers.
6	CLI Profiler	Standalone binary. All four commands (profile, diff, list, inspect). Colorized terminal output. CI-friendly exit codes. Paginated profile with cursor/offset support.
7	Framework Adapters	php-sentinel/laravel service provider, config, Artisan commands, Livewire dashboard. php-sentinel/symfony bundle with autowiring and console commands.
8	Release	Final test sweep to coverage targets, Docusaurus documentation site with real Shopify/Stripe examples, GitHub Actions CI pipeline, Packagist release, README with badges

8. Release and Distribution Plan

8.1 Packagist Packages

- php-sentinel/sentinel — core library (this spec)
- php-sentinel/laravel — Laravel adapter (service provider, config, Artisan, Livewire dashboard)
- php-sentinel/symfony — Symfony adapter (bundle, services.yaml, console commands)

8.2 GitHub Repository Standards

- PHP-CS-Fixer enforced via GitHub Actions on every PR
- PHPStan Level 8 — maximum static analysis strictness
- Pest test suite — must pass on PHP 8.3 and 8.4
- CHANGELOG.md following Keep a Changelog format
- CONTRIBUTING.md with PR guidelines and issue templates
- Semantic versioning from 1.0.0

8.3 Documentation Site

- Docusaurus with custom dark theme
- Quick Start guide: install → wrap client → see first schema — under 5 minutes
- Real-world integration guides: Shopify Orders API, Stripe Events, HubSpot Contacts, WooCommerce Products
- Interactive schema diff viewer — paste two JSON responses, see the diff
- API reference auto-generated from PhpDoc blocks

8.4 Community Growth Strategy

- Write "How Shopify silently broke our PHP integration — and the package that now prevents it" as a launch post on Dev.to and Medium
- Submit to Laravel News and PHP weekly newsletters
- Create a 4-minute YouTube walkthrough: real Shopify API drift detected live
- Answer "third party API changed PHP" questions on Stack Overflow with a reference to the package

9. Competitive Analysis and Moat

Verified February 2026: zero packages on Packagist solve this specific problem. The closest category — contract testing — requires bilateral cooperation that is impossible with third-party APIs you do not control.

Capability	php-sentinel	Pact PHP	APM Tools
Passive (no API cooperation needed)	✓	✗	✓
Field-level drift detection	✓	✓	✗
Severity classification	✓	✗	✗
Works for all third-party APIs	✓	✗	✓
Framework agnostic Composer package	✓	✓	✗
CLI offline diff tool	✓	✗	✗
Probabilistic baseline (no false positives)	✓	✗	✗

10. Success Metrics

Metric	Target	Timeframe
GitHub Stars	200+	First 3 months post-launch
Packagist Installs	1,000+	First 6 months
Dependents	10+ public packages	12 months
Test Coverage	>= 90%	At v1.0.0 release
PHPStan Level	Level 8	At v1.0.0 release
Laravel News Feature	1 article	Within 2 weeks of launch

11. Appendix — Composer Dependencies

```
// composer.json (core library) {     "name": "php-sentinel/sentinel",
"description": "Passive API contract monitor: detects when third-party APIs
silently change.",     "type": "library",     "license": "MIT",     "require": {
"php": "^8.3",         "psr/http-client": "^1.0",         "psr/event-dispatcher":
"^1.0",         "psr/log": "^3.0"     },     "require-dev": {
"pestphp/pest": "^3.0",         "phpstan/phpstan": "^1.10",
"friendsofphp/php-cs-fixer": "^3.0",         "guzzlehttp/guzzle": "^7.0",
"mockery/mockery": "^1.6"     },     "autoload": {         "psr-4": {
"Sentinel\\": "src/" }     },     "bin": ["bin/sentinel"] }
```

— END OF SPECIFICATION —