Toggly PHP

Core · Laravel · targeting · variants · lifecycle

Install and first flag

```
composer require toggly/feature-management-php
composer require guzzlehttp/guzzle guzzlehttp/psr7
```

Set TOGGLY_APP_KEY in the host environment. Create `new-dashboard` in Production. Save the entry point beside vendor/ and run `php first-flag.php`.

Core entry point

```
<?php
require __DIR__ . '/vendor/autoload.php';
use GuzzleHttp\Client;
use GuzzleHttp\Psr7\HttpFactory;
use Toggly\FeatureManagement\Config\TogglySettings;
use Toggly\FeatureManagement\Core\{
  FeatureProvider, FeatureManager,
  FeatureStateService, UsageStatsProvider
};
use Toggly\FeatureManagement\Http\TogglyHttpClient;

$key = getenv('TOGGLY_APP_KEY');
if (!$key) {
  throw new RuntimeException('Set TOGGLY_APP_KEY');
}
$settings = new TogglySettings([
  'app_key' => $key,
  'environment' => 'Production',
  'base_url' => getenv('TOGGLY_BASE_URL')
    ?: 'https://definitions.toggly.io/',
  'use_signed_definitions' => true,
  'enable_live_updates' => false,
]);
$http = new TogglyHttpClient(new Client(),
  new HttpFactory(), $settings->getBaseUrl());
$state = new FeatureStateService();
$provider = new FeatureProvider(
  $settings, $http, $state);
$usage = new UsageStatsProvider($settings, $http);
$fm = new FeatureManager(
  $provider, $usage, $provider);
try {
  $provider->refreshFeatures();
  echo $fm->isEnabled('new-dashboard',
    ['identity' => 'demo-reader'])
      ? 'New' : 'Original';
} finally {
  $usage->flush();
  $provider->shutdown();
}
```

Read the decision

Provider loads definitions; manager evaluates them. AlwaysOn prints New; AlwaysOff or an unknown key prints Original. Rerun after a dashboard change to refresh. Keep the disabled branch useful.

Request context

```
$ctx = [
  'identity' => 'alice',
  'groups' => ['beta-testers'],
  'claims' => ['role' => 'editor'],
  'request' => ['country' => 'US'],
];
$enabled = $fm->isEnabled('beta-access', $ctx);
```

Derive identity and claims from your authenticated request. The telemetry context provider does not inject targeting. Legacy userId/user_id remain identity fallbacks.

Filters and entities

Core evaluates Targeting, Percentage, TimeWindow, AlwaysOn/Off, UserClaims and HTTP segment filters. Use claims/request for the latter. PHP has no ContextProperty or entity registration API; arbitrary Order/plan fields do not add one.

Snapshots before construction

```
use Toggly\FeatureManagement\Storage as Store;
$snapshots =
  new Store\SnapshotProviders\FileSnapshotProvider(
  sys_get_temp_dir() . '/toggly-production',
  new Store\SnapshotSettings());
$provider = new FeatureProvider(
  $settings, $http, $state, $snapshots);
```

Core has no default snapshot provider. Rebuild the manager with this provider. Isolate storage by app/environment; PDO storage uses MySQL-compatible SQL.

Laravel bootstrap

```
composer require toggly/laravel guzzlehttp/guzzle
composer require guzzlehttp/psr7
php artisan vendor:publish --tag=toggly-config
```

.ENV

```
TOGGLY_APP_KEY=your-app-key
TOGGLY_ENVIRONMENT=Production
TOGGLY_BASE_URL=https://definitions.toggly.io/
TOGGLY_USE_SIGNED_DEFINITIONS=true
```

Auto-discovery registers the SDK. In AppServiceProvider::register, bind the PSR interfaces:

```
$this->app->singleton(
  \Psr\Http\Client\ClientInterface::class,
  fn () => new \GuzzleHttp\Client());
$this->app->singleton(
  \Psr\Http\Message\RequestFactoryInterface::class,
  fn () => new \GuzzleHttp\Psr7\HttpFactory());
```

In boot, refresh before the first check; register same-process terminating flush/shutdown. Complete class: Laravel host bootstrap.

Facade and native context

```
use Toggly\Laravel\Facades\Toggly;
$ctx = ['identity' => auth()->check()
  ? (string) auth()->id() : null];
$enabled = Toggly::isEnabled('beta-access', $ctx);
```

Facade calls need explicit targeting context. Blade and middleware derive identity, groups and IP; they do not supply full claims/request segment data.

Middleware registration

```
// Import at the top of bootstrap/app.php:
use Toggly\Laravel\Middleware as Flags;
// In the withMiddleware callback:
$m->alias([
  'feature' => Flags\FeatureGateMiddleware::class,
]);
$m->web(append: [
  Flags\FeatureGateAttributeMiddleware::class,
]);
```

Register the attribute middleware before relying on controller attributes. A disabled route gate returns 404 by default.

Attributes

```
use Toggly\Laravel\Attributes\{
  FeatureGate, FeatureUsage
};
#[FeatureGate('new-dashboard')]
#[FeatureUsage('new-dashboard')]
class DashboardController
{
  #[FeatureGate('analytics')]
  public function index() { return 'Analytics'; }
}
```

Class and method gates both apply. Middleware checks gates before usage regardless of declaration order. FeatureUsage performs an additional evaluation, not a separate explicit used-count call.

Blade

```
@feature('beta-access')
  <p>Beta is on</p>
@else
  <p>Original view</p>
@endfeature

@unlessfeature('legacy')
  <p>Legacy is off</p>
@endunlessfeature

<x-feature :name="['cart','a','b']"
           requirement="any">
  <p>At least one flag is on</p>
</x-feature>
```

Initially configured variants

```
$variantSettings = new TogglySettings([
  'app_key' => $key,
  'environment' => 'Production',
  'base_url' => getenv('TOGGLY_BASE_URL')
    ?: 'https://definitions.toggly.io/',
  'use_signed_definitions' => true,
  'enable_variants' => true,
  'identity' => 'alice',
  'enable_live_updates' => false,
]);
$variantHttp = new TogglyHttpClient(new Client(),
  new HttpFactory(),
  $variantSettings->getBaseUrl());
$vp = new FeatureProvider($variantSettings,
  $variantHttp, new FeatureStateService());
try {
  $vp->refreshFeatures();
  $variant = $vp->getVariant('pricing-test');
  $name = $variant['name'] ?? 'default';
  $value = $variant['configurationValue'] ?? null;
} finally {
  $vp->shutdown();
}
```

Uses imports/key from page 1. getVariant has no per-call context. Isolate providers and snapshots by identity. Neither the facade nor FeatureManager has getVariant.

Metrics and callbacks

```
use Toggly\FeatureManagement\Core\{
  MetricsService, MetricsRegistryService
};
$registry = new MetricsRegistryService();
$metrics = new MetricsService(
  $settings, $http, $provider, $registry);
$id = $registry->registerMeasurements(
  fn () => ['queue-depth' => 3.0]);
$metrics->sendMetrics();
$registry->unregisterMetrics($id);

$id = $state->whenDefinitionsChange(
  fn () => error_log('Definitions changed'));
$state->unregisterDefinitionsChange($id);
```

Host lifecycle

refresh_interval does not create a timer. Refresh at startup without a usable snapshot and schedule subsequent refreshFeatures calls. Drive tick for WebSocket work; it does not replace polling. Flush usage/metrics in their owning process and shut down the provider.

Persistent workers

Keep targeting, authorization and telemetry request-local. Laravel singleton telemetry captures a Request and dedupe state: a shared cache alone does not prove Octane or concurrent-worker isolation. See the guide before reusing these services.

Laravel snapshots

```
// config/toggly.php
'snapshot_provider' => 'cache',
'cache' => [
  'store' => 'redis', 'prefix' => 'toggly',
],
```

A scheduled process can update shared persistence; it cannot drain another process's in-memory counters.

WordPress

The helper is the global function `toggly_is_enabled`. Load the Composer autoloader, register cron intervals before activation, select snapshot storage and refresh before evaluation. See WordPress setup.

Resources

Core guide · Laravel guide · Advanced usage · Samples catalog

PHP 7.4+ through 8.5 (tested: 8.5). Laravel 8+; packed checks include Laravel 13. Run composer check-platform-reqs for your installation.