Release Notes

Versioning Scheme

Laravel and its other first-party packages follow Semantic Versioning. Major framework releases are released every six months (February and August), while minor and patch releases may be released as often as every week. Minor and patch releases should **never** contain breaking changes.

When referencing the Laravel framework or its components from your application or package, you should always use a version constraint such as `^6.0`, since major releases of Laravel do include breaking changes. However, we strive to always ensure you may update to a new major release in one day or less.

Support Policy

For LTS releases, such as Laravel 6, bug fixes are provided for 2 years and security fixes are provided for 3 years. These releases provide the longest window of support and maintenance. For general releases, bug fixes are provided for 6 months and security fixes are provided for 1 year. For all additional libraries, including Lumen, only the latest release receives bug fixes. In addition, please review the database versions supported by Laravel.

Version	Release	Bug Fixes Until	Security Fixes Until
5.5 (LTS)	August 30th, 2017	August 30th, 2019	August 30th, 2020
5.6	February 7th, 2018	August 7th, 2018	February 7th, 2019
5.7	September 4th, 2018	March 4th, 2019	September 4th, 2019
5.8	February 26th, 2019	August 26th, 2019	February 26th, 2020
6 (LTS)	September 3rd, 2019	September 3rd, 2021	September 3rd, 2022

Laravel 6

Laravel 6 (LTS) continues the improvements made in Laravel 5.8 by introducing semantic versioning, compatibility with Laravel Vapor, improved authorization responses, job middleware, lazy collections, sub-query improvements, the extraction of frontend scaffolding to the `laravel/ui` Composer package, and a variety of other bug fixes and usability improvements.

Semantic Versioning

The Laravel framework (`laravel/framework`) package now follows the semantic versioning standard. This makes the framework consistent with the other first-party Laravel packages which already followed this versioning standard. The Laravel release cycle will remain unchanged.

Laravel Vapor Compatibility

Laravel Vapor was built by Taylor Otwell.

Laravel 6 provides compatibility with Laravel Vapor, an auto-scaling serverless deployment platform for Laravel. Vapor abstracts the complexity of managing Laravel applications on AWS Lambda, as well as interfacing those applications with SQS queues, databases, Redis clusters, networks, CloudFront CDN, and more.

Improved Exceptions Via Ignition

Laravel 6 ships with Ignition, a new open source exception detail page created by Freek Van der Herten and Marcel Pociot. Ignition offers many benefits over previous releases, such as improved Blade error file and line number handling, runnable solutions for common problems, code editing, exception sharing, and an improved UX.

Improved Authorization Responses

Improved authorization responses were implemented by Gary Green.

In previous releases of Laravel, it was difficult to retrieve and expose custom

authorization messages to end users. This made it difficult to explain to end-users exactly why a particular request was denied. In Laravel 6, this is now much easier using authorization response messages and the new `Gate::inspect` method. For example, given the following policy method:

```
/**
 * Determine if the user can view the given flight.
 *
 * @param  \App\User  $user
 * @param  \App\Flight  $flight
 * @return mixed
 */
public function view(User $user, Flight $flight)
{
    return $this->deny('Explanation of denial.');
}
```

The authorization policy's response and message may be easily retrieved using the `Gate::inspect` method:

```
$response = Gate::inspect('view', $flight);

if ($response->allowed()) {
    // User is authorized to view the flight...
}

if ($response->denied()) {
    echo $response->message();
}
```

In addition, these custom messages will automatically be returned to your frontend when using helper methods such as `$this->authorize` or `Gate::authorize` from your routes or controllers.

Job Middleware

Job middleware were implemented by Taylor Otwell.

Job middleware allow you to wrap custom logic around the execution of queued jobs, reducing boilerplate in the jobs themselves. For example, in previous releases of Laravel, you may have wrapped the logic of a job's `handle` method within a rate-limited callback:

```
/**
 * Execute the job.
 *
 * @return void
 */
public function handle()
{
    Redis::throttle('key')->block(0)->allow(1)->every(5)->then(function () {
        info('Lock obtained...');

        // Handle job...
    }, function () {
        // Could not obtain lock...

        return $this->release(5);
    });
}
```

In Laravel 6, this logic may be extracted into a job middleware, allowing you to keep your job's `handle` method free of any rate limiting responsibilities:

```
<?php

namespace App\Jobs\Middleware;

use Illuminate\Support\Facades\Redis;

class RateLimited
{
    /**
     * Process the queued job.
     *
     * @param  mixed  $job
     * @param  callable  $next
     * @return mixed
     */
    public function handle($job, $next)
    {
        Redis::throttle('key')
                ->block(0)->allow(1)->every(5)
                ->then(function () use ($job, $next) {
```

```
                    // Lock obtained...

                    $next($job);
                }, function () use ($job) {
                    // Could not obtain lock...

                    $job->release(5);
                });
    }
}
```

After creating middleware, they may be attached to a job by returning them from the job's `middleware` method:

```
use App\Jobs\Middleware\RateLimited;

/**
 * Get the middleware the job should pass through.
 *
 * @return array
 */
public function middleware()
{
    return [new RateLimited];
}
```

Lazy Collections

Lazy collections were implemented by Joseph Silber.

Many developers already enjoy Laravel's powerful Collection methods. To supplement the already powerful `Collection` class, Laravel 6 introduces a `LazyCollection`, which leverages PHP's generators to allow you to work with very large datasets while keeping memory usage low.

For example, imagine your application needs to process a multi-gigabyte log file while taking advantage of Laravel's collection methods to parse the logs. Instead of reading the entire file into memory at once, lazy collections may be used to keep only a small part of the file in memory at a given time:

```
use App\LogEntry;
use Illuminate\Support\LazyCollection;

LazyCollection::make(function () {
    $handle = fopen('log.txt', 'r');

    while (($line = fgets($handle)) !== false) {
        yield $line;
    }
})
->chunk(4)
->map(function ($lines) {
    return LogEntry::fromLines($lines);
})
->each(function (LogEntry $logEntry) {
    // Process the log entry...
});
```

Or, imagine you need to iterate through 10,000 Eloquent models. When using traditional Laravel collections, all 10,000 Eloquent models must be loaded into memory at the same time:

```
$users = App\User::all()->filter(function ($user) {
    return $user->id > 500;
});
```

However, beginning in Laravel 6, the query builder's `cursor` method has been updated to return a `LazyCollection` instance. This allows you to still only run a single query against the database but also only keep one Eloquent model loaded in memory at a time. In this example, the `filter` callback is not executed until we actually iterate over each user individually, allowing for a drastic reduction in memory usage:

```
$users = App\User::cursor()->filter(function ($user) {
    return $user->id > 500;
});

foreach ($users as $user) {
    echo $user->id;
}
```

Eloquent Subquery Enhancements

Eloquent subquery enhancements were implemented by Jonathan Reinink.

Laravel 6 introduces several new enhancements and improvements to database subquery support. For example, let's imagine that we have a table of flight `destinations` and a table of `flights` to destinations. The `flights` table contains an `arrived_at` column which indicates when the flight arrived at the destination.

Using the new subquery select functionality in Laravel 6, we can select all of the `destinations` and the name of the flight that most recently arrived at that destination using a single query:

```
return Destination::addSelect(['last_flight' => Flight::select('name')
    ->whereColumn('destination_id', 'destinations.id')
    ->orderBy('arrived_at', 'desc')
    ->limit(1)
])->get();
```

In addition, we can use new subquery features added to the query builder's `orderBy` function to sort all destinations based on when the last flight arrived at that destination. Again, this may be done while executing a single query against the database:

```
return Destination::orderByDesc(
    Flight::select('arrived_at')
        ->whereColumn('destination_id', 'destinations.id')
        ->orderBy('arrived_at', 'desc')
        ->limit(1)
)->get();
```

Laravel UI

The frontend scaffolding typically provided with previous releases of Laravel has been extracted into a `laravel/ui` Composer package. This allows the first-party UI scaffolding to be developed and versioned separately from the primary framework. As a result of this change, no Bootstrap or Vue code is present in default framework scaffolding, and the `make:auth` command has been extracted from the framework as well.

In order to restore the traditional Vue / Bootstrap scaffolding present in previous releases of Laravel, you may install the `laravel/ui` package and use the `ui` Artisan command to install the frontend scaffolding:

```
composer require laravel/ui --dev

php artisan ui vue --auth
```

Become a Laravel Partner

Laravel Partners are elite shops providing top-notch Laravel development and consulting. Each of our partners can help you craft a beautiful, well-architected project.

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization
- Artisan Console
- Database
- Eloquent ORM
- Testing

Resources

- Laracasts
- Laravel News
- Laracon
- Laracon EU
- Laracon AU
- Jobs
- Certification
- Forums

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet
- DevSquad
- Ideil
- Cyber-Duck
- ABOUT YOU
- Become A Partner

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova
- Echo
- Valet
- Mix
- Spark
- Cashier
- Homestead
- Dusk
- Passport
- Scout
- Socialite

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable and creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in most web projects.