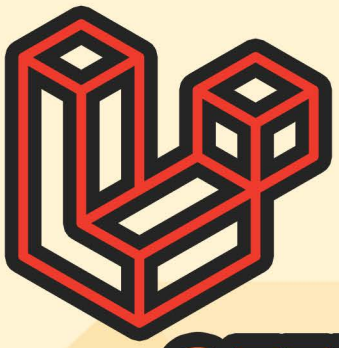

Laravel 8.x

OFFICIAL DOCUMENTATION

Laravel 8.x Official Documentation

This book is for sale at https://leanpub.com/laravel8x
This version was synced and publish on **07/09/2020**

Laravel Documentation

You can find the online version of the Laravel documentation at https://laravel.com/docs

Contribution Guidelines

If you are submitting documentation for the **current stable release**, submit it to the corresponding branch. For example, documentation for Laravel 8 would be submitted to the `8.x` branch. Documentation intended for the next release of Laravel should be submitted to the `master` branch.

Prologue

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 `^8.0`, 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
7	March 3rd, 2020	September 10th, 2020	March 3rd, 2021
8	September 8th, 2020	March 8th, 2021	September 8th, 2021

Laravel 8

Laravel 8 continues the improvements made in Laravel 7.x by introducing Laravel Jetstream, model factory classes, migration squashing, job batching, improved rate limiting, queue improvements, dynamic Blade components, Tailwind pagination views, time testing helpers, improvements to `artisan serve`, event listener improvements, and a variety of other bug fixes and usability improvements.

Laravel Jetstream

Laravel Jetstream was written by Taylor Otwell.

Laravel Jetstream is a beautifully designed application scaffolding for Laravel. Jetstream provides the perfect starting point for your next project and includes login, registration, email verification, two-factor authentication, session management, API support via Laravel Sanctum, and optional team management. Laravel Jetstream replaces and improves upon the legacy authentication UI scaffolding available for previous versions of Laravel.

Jetstream is designed using Tailwind CSS and offers your choice of Livewire or Inertia scaffolding.

Models Directory

By overwhelming community demand, the default Laravel application skeleton now contains an `app/Models` directory. We hope you enjoy this new home for your Eloquent models! All relevant generator commands have been updated to assume models exist within the `app/Models` directory if it exists. If the directory does not exist, the framework will assume your models should be placed within the `app` directory.

Model Factory Classes

Model factory classes were contributed by Taylor Otwell.

Eloquent model factories have been entirely re-written as class based factories and improved to have first-class relationship support. For example, the `UserFactory` included with Laravel is written like so:

```php
<?php

namespace Database\Factories;

use App\Models\User;
use Illuminate\Database\Eloquent\Factories\Factory;
use Illuminate\Support\Str;

class UserFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = User::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'name' => $this->faker->name,
            'email' => $this->faker->unique()->safeEmail,
            'email_verified_at' => now(),
            'password' => '$2y$10$92IXUNpkjO0rOQ5byMi.Ye4oKoEa3Ro9llC/.og/at2.uheWG/igi', // password
            'remember_token' => Str::random(10),
        ];
    }
}
```

Thanks to the new `HasFactory` trait available on generated models, the model factory may be used like so:

```php
use App\Models\User;

User::factory()->count(50)->create();
```

Since model factories are now simple PHP classes, state transformations may be written as class methods. In addition, you may add any other helper classes to your Eloquent model factory as needed.

For example, your `User` model might have a `suspended` state that modifies one of its default attribute values. You may define your state transformations using the base factory's `state` method. You may name your state method anything you like. After all, it's just a typical PHP method:

```php
/**
 * Indicate that the user is suspended.
 *
 * @return \Illuminate\Database\Eloquent\Factories\Factory
 */
public function suspended()
{
    return $this->state([
        'account_status' => 'suspended',
    ]);
}
```

After defining the state transformation method, we may use it like so:

```php
use App\Models\User;

User::factory()->count(5)->suspended()->create();
```

As mentioned, Laravel 8's model factories contain first class support for relationships. So, assuming our `User` model has a `posts` relationship method, we may simply run the following code to generate a user with three posts:

```php
$users = User::factory()
            ->hasPosts(3, [
                'published' => false,
            ])
            ->create();
```

To ease the upgrade process, the laravel/legacy-factories package has been released to provide support for the previous iteration of model factories within Laravel 8.x.

Laravel's re-written factories contain many more features that we think you will love. To learn more about model factories, please consult the database testing documentation.

Migration Squashing

Migration squashing was contributed by Taylor Otwell.

As you build your application, you may accumulate more and more migrations over time. This can lead to your migration directory becoming bloated with potentially hundreds of migrations. If you would like, you may now "squash" your migrations into a single SQL file. To get started, execute the `schema:dump` command:

```
php artisan schema:dump

// Dump the current database schema and prune all existing migrations...
php artisan schema:dump --prune
```

When you execute this command, Laravel will write a "schema" file to your `database/schema` directory. Now, when you attempt to migrate your database and no other migrations have been executed, Laravel will execute the schema file's SQL first. After executing the schema file's commands, Laravel will execute any remaining migrations that were not part of the schema dump.

Job Batching

Job batching was contributed by Taylor Otwell & Mohamed Said.

Laravel's job batching feature allows you to easily execute a batch of jobs and then perform some action when the batch of jobs has completed executing.

The new `batch` method of the `Bus` facade may be used to dispatch a batch of jobs. Of course, batching is primarily useful when combined with completion callbacks. So, you may use the `then`, `catch`, and `finally` methods to define completion callbacks for the batch. Each of these callbacks will receive an `Illuminate\Bus\Batch` instance when they are invoked:

```
use App\Jobs\ProcessPodcast;
use App\Podcast;
use Illuminate\Bus\Batch;
use Illuminate\Support\Facades\Batch;
use Throwable;

$batch = Bus::batch([
    new ProcessPodcast(Podcast::find(1)),
    new ProcessPodcast(Podcast::find(2)),
    new ProcessPodcast(Podcast::find(3)),
    new ProcessPodcast(Podcast::find(4)),
    new ProcessPodcast(Podcast::find(5)),
])->then(function (Batch $batch) {
    // All jobs completed successfully...
})->catch(function (Batch $batch, Throwable $e) {
    // First batch job failure detected...
})->finally(function (Batch $batch) {
    // The batch has finished executing...
})->dispatch();

return $batch->id;
```

To learn more about job batching, please consult the queue documentation.

Improved Rate Limiting

Rate limiting improvements were contributed by Taylor Otwell.

Laravel's request rate limiter feature has been augmented with more flexibility and power, while still maintaining backwards compatibility with previous release's `throttle` middleware API.

Rate limiters are defined using the `RateLimiter` facade's `for` method. The `for` method accepts a rate limiter name and a Closure that returns the limit configuration that should apply to routes that are assigned this rate limiter:

```
use Illuminate\Cache\RateLimiting\Limit;
use Illuminate\Support\Facades\RateLimiter;

RateLimiter::for('global', function (Request $request) {
    return Limit::perMinute(1000);
});
```

Since rate limiter callbacks receive the incoming HTTP request instance, you may build the appropriate rate limit dynamically based on the incoming request or authenticated user:

```
RateLimiter::for('uploads', function (Request $request) {
    return $request->user()->vipCustomer()
                ? Limit::none()
                : Limit::perMinute(100);
});
```

Sometimes you may wish to segment rate limits by some arbitrary value. For example, you may wish to allow users to access a given route 100 times per minute per IP address. To accomplish this, you may use the `by` method when building your rate limit:

```
RateLimiter::for('uploads', function (Request $request) {
    return $request->user()->vipCustomer()
                ? Limit::none()
                : Limit::perMinute(100)->by($request->ip());
});
```

Rate limiters may be attached to routes or route groups using the `throttle` middleware. The throttle middleware accepts the name of the rate limiter you wish to assign to the route:

```
Route::middleware(['throttle:uploads'])->group(function () {
    Route::post('/audio', function () {
        //
    });

    Route::post('/video', function () {
        //
    });
});
```

To learn more about rate limiting, please consult the routing documentation.

Improved Maintenance Mode

Maintenance mode improvements were contributed by Taylor Otwell with inspiration from Spatie.

In previous releases of Laravel, the `php artisan down` maintenance mode feature may be bypassed using an "allow list" of IP addresses that were allowed to access the application. This feature has been removed in favor of a simpler "secret" / token solution.

While in maintenance mode, you may use the `secret` option to specify a maintenance mode bypass token:

```
php artisan down --secret="1630542a-246b-4b66-afa1-dd72a4c43515"
```

After placing the application in maintenance mode, you may navigate to the application URL matching this token and Laravel will issue a maintenance mode bypass cookie to your browser:

```
https://example.com/1630542a-246b-4b66-afa1-dd72a4c43515
```

When accessing this hidden route, you will then be redirected to the `/` route of the application. Once the cookie has been issued to your browser, you will be able to browse the application normally as if it was not in maintenance mode.

Pre-Rendering The Maintenance Mode View

If you utilize the `php artisan down` command during deployment, your users may still occasionally encounter errors if they access the application while your Composer dependencies or other infrastructure components are updating. This occurs because a significant part of the Laravel framework must boot in order to determine your application is in maintenance mode and render the maintenance mode view using the templating engine.

For this reason, Laravel now allows you to pre-render a maintenance mode view that will be returned at the very beginning of the request cycle. This view is rendered before any of your application's dependencies have loaded. You may pre-render a template of your choice using the `down` command's `render` option:

```
php artisan down --render="errors::503"
```

Closure Dispatch / Chain `catch`

Catch improvements were contributed by Mohamed Said.

Using the new `catch` method, you may now provide a Closure that should be executed if a queued Closure fails to complete successfully after exhausting all of your queue's configured retry attempts:

```
use Throwable;

dispatch(function () use ($podcast) {
    $podcast->publish();
})->catch(function (Throwable $e) {
    // This job has failed...
});
```

Dynamic Blade Components

Dynamic Blade components were contributed by Taylor Otwell.

Sometimes you may need to render a component but not know which component should be rendered until runtime. In this situation, you may now use Laravel's built-in `dynamic-component` component to render the component based on a runtime value or variable:

```
<x-dynamic-component :component="$componentName" class="mt-4" />
```

To learn more about Blade components, please consult the Blade documentation.

Event Listener Improvements

Event listener improvements were contributed by Taylor Otwell.

Closure based event listeners may now be registered by only passing the Closure to the `Event::listen` method. Laravel will inspect the Closure to determine which type of event the listener handlers:

```
use App\Events\PodcastProcessed;
use Illuminate\Support\Facades\Event;

Event::listen(function (PodcastProcessed $event) {
    //
});
```

In addition, Closure based event listeners may now be marked as queueable using the `Illuminate\Events\queueable` function:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;

Event::listen(queueable(function (PodcastProcessed $event) {
    //
}));
```

Like queued jobs, you may use the `onConnection`, `onQueue`, and `delay` methods to customize the execution of the queued listener:

```
Event::listen(queueable(function (PodcastProcessed $event) {
    //
})->onConnection('redis')->onQueue('podcasts')->delay(now()->addSeconds(10)));
```

If you would like to handle anonymous queued listener failures, you may provide a Closure to the `catch` method while defining the `queueable` listener:

```
use App\Events\PodcastProcessed;
use function Illuminate\Events\queueable;
use Illuminate\Support\Facades\Event;
use Throwable;

Event::listen(queueable(function (PodcastProcessed $event) {
    //
})->catch(function (PodcastProcessed $event, Throwable $e) {
    // The queued listener failed...
}));
```

Time Testing Helpers

Time testing helpers were contributed by Taylor Otwell with inspiration from Ruby on Rails.

When testing, you may occasionally need to modify the time returned by helpers such as `now` or `Illuminate\Support\Carbon::now()`. Laravel's base feature test class now includes helpers that allow you to manipulate the current time:

```
public function testTimeCanBeManipulated()
{
    // Travel into the future...
    $this->travel(5)->milliseconds();
    $this->travel(5)->seconds();
    $this->travel(5)->minutes();
    $this->travel(5)->hours();
    $this->travel(5)->days();
    $this->travel(5)->weeks();
    $this->travel(5)->years();

    // Travel into the past...
    $this->travel(-5)->hours();

    // Travel to an explicit time...
    $this->travelTo(now()->subHours(6));

    // Return back to the present time...
    $this->travelBack();
}
```

Artisan `serve` Improvements

Artisan `serve` improvements were contributed by Taylor Otwell.

The Artisan `serve` command has been improved with automatic reloading when environment variable changes are detected within your local `.env` file. Previously, the command had to be manually stopped and restarted.

Tailwind Pagination Views

The Laravel paginator has been updated to use the Tailwind CSS framework by default. Tailwind CSS is a highly customizable, low-level CSS framework that gives you all of the building blocks you need to build bespoke designs without any annoying opinionated styles you have to fight to override. Of course, Bootstrap 3 and 4 views remain available as well.

Upgrade Guide

High Impact Changes

Medium Impact Changes

Upgrading To 8.0 From 7.x

Estimated Upgrade Time: 15 Minutes

> ⚠⚠ We attempt to document every possible breaking change. Since some of these breaking changes are in obscure parts of the framework only a portion of these changes may actually affect your application. ⚠⚠

PHP 7.3.0 Required

Likelihood Of Impact: Medium

The new minimum PHP version is now 7.3.0.

Updating Dependencies

Update the following dependencies in your `composer.json` file:

- `laravel/framework` to `^8.0`
- `nunomaduro/collision` to `^5.0`
- `guzzlehttp/guzzle` to `^7.0.1`
- `facade/ignition` to `^2.3.6`

The following first-party packages have new major releases to support Laravel 8. If applicable, you should read their individual upgrade guides before upgrading:

- Horizon v5.0
- Passport v10.0
- Socialite v5.0
- Telescope v4.0

Finally, examine any other third-party packages consumed by your application and verify you are using the proper version for Laravel 8 support.

Collections

The `isset` Method

Likelihood Of Impact: Low

To be consistent with typical PHP behavior, the `offsetExists` method of `Illuminate\Support\Collection` has been updated to use `isset` instead of `array_key_exists`. This may present a change in behavior when dealing with collection items that have a value of `null`:

```
$collection = collect([null]);

// Laravel 7.x - true
isset($collection[0]);

// Laravel 8.x - false
isset($collection[0]);
```

Eloquent

Model Factories

Likelihood Of Impact: High

Laravel's model factories feature has been totally rewritten to support classes and is not compatible with Laravel 7.x style factories. However, to ease the upgrade process, a new `laravel/legacy-factories` package has been created to continue using your existing factories with Laravel 8.x. You may install this package via Composer:

```
composer require laravel/legacy-factories
```

The `Castable` Interface

Likelihood Of Impact: Low

The `castUsing` method of the `Castable` interface has been updated to accept an array of arguments. If you are implementing this interface you should update your implementation accordingly:

```
public static function castUsing(array $arguments);
```

Increment / Decrement Events

Likelihood Of Impact: Low

Proper "update" and "save" related model events will now be dispatched when executing the `increment` or `decrement` methods on Eloquent model instances.

Events

The `Dispatcher` Contract

Likelihood Of Impact: Low

The `listen` method of the `Illuminate\Contracts\Events\Dispatcher` contract has been updated to make the `$listener` property optional. This change was made to support automatic detection of handled event types via reflection. If you are manually implementing this interface, you should update your implementation accordingly:

```
public function listen($events, $listener = null);
```

Framework

Maintenance Mode Updates

Likelihood Of Impact: Optional

The maintenance mode feature of Laravel has been improved in Laravel 8.x. Pre-rendering the maintenance mode template is now supported and eliminates the chances of end users encountering errors during maintenance mode. However, to support this, the following lines must be added to your `public/index.php` file. These lines should be placed directly under the existing `LARAVEL_START` constant definition:

```
define('LARAVEL_START', microtime(true));

if (file_exists(__DIR__.'/../storage/framework/maintenance.php')) {
    require __DIR__.'/../storage/framework/maintenance.php';
}
```

Manager `$app` Property

Likelihood Of Impact: Low

The previously deprecated `$app` property of the `Illuminate\Support\Manager` class has been removed. If you were relying on this property, you should use the `$container` property instead.

The `elixir` Helper

Likelihood Of Impact: Low

The previously deprecated `elixir` helper has been removed. Applications still using this method are encouraged to upgrade to Laravel Mix.

Mail

The `sendNow` Method

Likelihood Of Impact: Low

The previously deprecated `sendNow` method has been removed. Instead, please use the `send` method.

Pagination

Pagination Defaults

Likelihood Of Impact: High

The paginator now uses the Tailwind CSS framework for its default styling. In order to keep using Bootstrap, you should add the following method call to the `boot` method of your application's `AppServiceProvider`:

```
use Illuminate\Pagination\Paginator;

Paginator::useBootstrap();
```

Queue

The `retryAfter` Method

Likelihood Of Impact: High

For consistency with other features of Laravel, the `retryAfter` method and `retryAfter` property of queued jobs, mailers, notifications, and listeners has been renamed to `backoff`. You should update the name of this method / property in the relevant classes in your application.

The `timeoutAt` Property

Likelihood Of Impact: High

The `timeoutAt` property of queued jobs, notifications, and listeners has been renamed to `retryUntil`. You should update the name of this property in the relevant classes in your application.

Failed Jobs Table Batch Support

Likelihood Of Impact: Optional

If you plan to use the job batching features of Laravel 8.x, your `failed_jobs` database table will need to be updated. First, a new `uuid` column should be added to your table:

```
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

Schema::table('failed_jobs', function (Blueprint $table) {
    $table->string('uuid')->after('id')->unique();
});
```

Next, the `failed.driver` configuration option within your `queue` configuration file should be updated to `database-uuids`.

Scheduling

The `cron-expression` Library

Likelihood Of Impact: Low

Laravel's dependency on `dragonmantank/cron-expression` has been updated from `2.x` to `3.x`. This should not cause any breaking change in your application unless you are interacting with the `cron-expression` library directly. If you are interacting with this library directly, please review its change log.

Session

The `Session` Contract

Likelihood Of Impact: Low

The `Illuminate\Contracts\Session\Session` contract has received a new `pull` method. If you are implementing this contract manually, you should update your implementation accordingly:

```
/**
 * Get the value of a given key and then forget it.
 *
 * @param  string  $key
 * @param  mixed  $default
 * @return mixed
 */
public function pull($key, $default = null);
```

Testing

The `assertExactJson` Method

Likelihood Of Impact: Medium

The `assertExactJson` method now requires numeric keys of compared arrays to match and be in the same order. If you would like to compare JSON against an array without requiring numerically keyed arrays to have the same order, you may use the `assertSimilarJson` method instead.

Validation

Database Rule Connections

Likelihood Of Impact: Low

The `unique` and `exists` rules will now respect the specified connection name (accessed via the model's `getConnectionName` method) of Eloquent models when performing queries.

Miscellaneous

We also encourage you to view the changes in the `laravel/laravel` GitHub repository. While many of these changes are not required, you may wish to keep these files in sync with your application. Some of these changes will be covered in this upgrade guide, but others, such as changes to configuration files or comments, will not be. You can easily view the changes with the GitHub comparison tool and choose which updates are important to you.

Contribution Guide

Bug Reports

To encourage active collaboration, Laravel strongly encourages pull requests, not just bug reports. "Bug reports" may also be sent in the form of a pull request containing a failing test.

However, if you file a bug report, your issue should contain a title and a clear description of the issue. You should also include as much relevant information as possible and a code sample that demonstrates the issue. The goal of a bug report is to make it easy for yourself - and others - to replicate the bug and develop a fix.

Remember, bug reports are created in the hope that others with the same problem will be able to collaborate with you on solving it. Do not expect that the bug report will automatically see any activity or that others will jump to fix it. Creating a bug report serves to help yourself and others start on the path of fixing the problem. If you want to chip in, you can help out by fixing any bugs listed in our issue trackers.

The Laravel source code is managed on GitHub, and there are repositories for each of the Laravel projects:

- Laravel Application
- Laravel Art
- Laravel Documentation
- Laravel Dusk
- Laravel Cashier Stripe
- Laravel Cashier Paddle
- Laravel Echo
- Laravel Envoy
- Laravel Framework
- Laravel Homestead
- Laravel Homestead Build Scripts
- Laravel Horizon
- Laravel Passport
- Laravel Sanctum
- Laravel Scout
- Laravel Socialite
- Laravel Telescope
- Laravel Website
- Laravel UI

Support Questions

Laravel's GitHub issue trackers are not intended to provide Laravel help or support. Instead, use one of the following channels:

- GitHub Discussions
- Laracasts Forums
- Laravel.io Forums
- StackOverflow
- Discord
- Larachat
- IRC

Core Development Discussion

You may propose new features or improvements of existing Laravel behavior in the Laravel Ideas issue board. If you propose a new feature, please be willing to implement at least some of the code that would be needed to complete the feature.

Informal discussion regarding bugs, new features, and implementation of existing features takes place in the `#internals` channel of the Laravel Discord server. Taylor Otwell, the maintainer of Laravel, is typically present in the channel on weekdays from 8am-5pm (UTC-06:00 or America/Chicago), and sporadically present in the channel at other times.

Which Branch?

All bug fixes should be sent to the latest stable branch or to the current LTS branch. Bug fixes should **never** be sent to the `master` branch unless they fix features that exist only in the upcoming release.

Minor features that are **fully backward compatible** with the current release may be sent to the latest stable branch.

Major new features should always be sent to the `master` branch, which contains the upcoming release.

If you are unsure if your feature qualifies as a major or minor, please ask Taylor Otwell in the `#internals` channel of the Laravel Discord server.

Compiled Assets

If you are submitting a change that will affect a compiled file, such as most of the files in `resources/sass` or `resources/js` of the `laravel/laravel` repository, do not commit the compiled files. Due to their large size, they cannot realistically be reviewed by a maintainer. This could be exploited as a way to inject malicious code into Laravel. In order to defensively prevent this, all compiled files will be generated and committed by Laravel maintainers.

Security Vulnerabilities

If you discover a security vulnerability within Laravel, please send an email to Taylor Otwell at taylor@laravel.com. All security vulnerabilities will be promptly addressed.

Coding Style

Laravel follows the PSR-2 coding standard and the PSR-4 autoloading standard.

PHPDoc

Below is an example of a valid Laravel documentation block. Note that the `@param` attribute is followed by two spaces, the argument type, two more spaces, and finally the variable name:

```
/**
 * Register a binding with the container.
 *
 * @param  string|array  $abstract
 * @param  \Closure|string|null  $concrete
 * @param  bool  $shared
 * @return void
 *
 * @throws \Exception
 */
public function bind($abstract, $concrete = null, $shared = false)
{
    //
}
```

StyleCI

Don't worry if your code styling isn't perfect! StyleCI will automatically merge any style fixes into the Laravel repository after pull requests are merged. This allows us to focus on the content of the contribution and not the code style.

Code of Conduct

The Laravel code of conduct is derived from the Ruby code of conduct. Any violations of the code of conduct may be reported to Taylor Otwell (taylor@laravel.com):

- Participants will be tolerant of opposing views.
- Participants must ensure that their language and actions are free of personal attacks and disparaging personal remarks.
- When interpreting the words and actions of others, participants should always assume good intentions.
- Behavior that can be reasonably considered harassment will not be tolerated.

Getting-Started

Installation

Installation

Server Requirements

The Laravel framework has a few system requirements. All of these requirements are satisfied by the Laravel Homestead virtual machine, so it's highly recommended that you use Homestead as your local Laravel development environment.

However, if you are not using Homestead, you will need to make sure your server meets the following requirements:

- PHP >= 7.3
- BCMath PHP Extension
- Ctype PHP Extension
- Fileinfo PHP extension
- JSON PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PDO PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

Installing Laravel

Laravel utilizes Composer to manage its dependencies. So, before using Laravel, make sure you have Composer installed on your machine.

Via Laravel Installer

First, download the Laravel installer using Composer:

```
composer global require laravel/installer
```

Make sure to place Composer's system-wide vendor bin directory in your `$PATH` so the laravel executable can be located by your system. This directory exists in different locations based on your operating system; however, some common locations include:

- macOS: `$HOME/.composer/vendor/bin`
- Windows: `%USERPROFILE%\AppData\Roaming\Composer\vendor\bin`

- GNU / Linux Distributions: `$HOME/.config/composer/vendor/bin` or `$HOME/.composer/vendor/bin`

You could also find the composer's global installation path by running `composer global about` and looking up from the first line.

Once installed, the `laravel new` command will create a fresh Laravel installation in the directory you specify. For instance, `laravel new blog` will create a directory named `blog` containing a fresh Laravel installation with all of Laravel's dependencies already installed:

```
laravel new blog
```

Via Composer Create-Project

Alternatively, you may also install Laravel by issuing the Composer `create-project` command in your terminal:

```
composer create-project --prefer-dist laravel/laravel blog
```

Local Development Server

If you have PHP installed locally and you would like to use PHP's built-in development server to serve your application, you may use the `serve` Artisan command. This command will start a development server at `http://localhost:8000`:

```
php artisan serve
```

More robust local development options are available via Homestead and Valet.

Configuration

Public Directory

After installing Laravel, you should configure your web server's document / web root to be the `public` directory. The `index.php` in this directory serves as the front controller for all HTTP requests entering your application.

Configuration Files

All of the configuration files for the Laravel framework are stored in the `config` directory. Each option is documented, so feel free to look through the files and get familiar with the options available to you.

Directory Permissions

After installing Laravel, you may need to configure some permissions. Directories within the `storage` and the `bootstrap/cache` directories should be writable by your web server or Laravel will not run. If you are using the Homestead virtual machine, these permissions should already be set.

Application Key

The next thing you should do after installing Laravel is set your application key to a random string. If you installed Laravel via Composer or the Laravel installer, this key has already been set for you by the `php artisan key:generate` command.

Typically, this string should be 32 characters long. The key can be set in the `.env` environment file. If you have not copied the `.env.example` file to a new file named `.env`, you should do that now. **If the application key is not set, your user sessions and other encrypted data will not be secure!**

Additional Configuration

Laravel needs almost no other configuration out of the box. You are free to get started developing! However, you may wish to review the `config/app.php` file and its documentation. It contains several options such as `timezone` and `locale` that you may wish to change according to your application.

You may also want to configure a few additional components of Laravel, such as:

- Cache
- Database
- Session

Web Server Configuration

Directory Configuration

Laravel should always be served out of the root of the "web directory" configured for your web server. You should not attempt to serve a Laravel application out of a subdirectory of the "web directory". Attempting to do so could expose sensitive files present within your application.

Pretty URLs

Apache

Laravel includes a `public/.htaccess` file that is used to provide URLs without the `index.php` front controller in the path. Before serving Laravel with Apache, be sure to enable the `mod_rewrite` module so the `.htaccess` file will be honored by the server.

If the `.htaccess` file that ships with Laravel does not work with your Apache installation, try this alternative:

```
Options +FollowSymLinks -Indexes
RewriteEngine On

RewriteCond %{HTTP:Authorization} .
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

Nginx

If you are using Nginx, the following directive in your site configuration will direct all requests to the `index.php` front controller:

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

When using Homestead or Valet, pretty URLs will be automatically configured.

Configuration

Introduction

All of the configuration files for the Laravel framework are stored in the `config` directory. Each option is documented, so feel free to look through the files and get familiar with the options available to you.

Environment Configuration

It is often helpful to have different configuration values based on the environment where the application is running. For example, you may wish to use a different cache driver locally than you do on your production server.

To make this a cinch, Laravel utilizes the DotEnv PHP library by Vance Lucas. In a fresh Laravel installation, the root directory of your application will contain a `.env.example` file. If you install Laravel via Composer, this file will automatically be copied to `.env`. Otherwise, you should copy the file manually.

Your `.env` file should not be committed to your application's source control, since each developer / server using your application could require a different environment configuration. Furthermore, this would be a security risk in the event an intruder gains access to your source control repository, since any sensitive credentials would get exposed.

If you are developing with a team, you may wish to continue including a `.env.example` file with your application. By putting placeholder values in the example configuration file, other developers on your team can clearly see which environment variables are needed to run your application. You may also create a `.env.testing` file. This file will override the `.env` file when running PHPUnit tests or executing Artisan commands with the `--env=testing` option.

> 💡💡 Any variable in your `.env` file can be overridden by external environment variables such as server-level or system-level environment variables. 💡💡

Environment Variable Types

All variables in your `.env` files are parsed as strings, so some reserved values have been created to allow you to return a wider range of types from the `env()` function:

`.env` Value	`env()` Value
true	(bool) true
(true)	(bool) true

`.env` Value	`env()` Value
false	(bool) false
(false)	(bool) false
empty	(string) ''
(empty)	(string) ''
null	(null) null
(null)	(null) null

If you need to define an environment variable with a value that contains spaces, you may do so by enclosing the value in double quotes.

```
APP_NAME="My Application"
```

Retrieving Environment Configuration

All of the variables listed in this file will be loaded into the `$_ENV` PHP super-global when your application receives a request. However, you may use the `env` helper to retrieve values from these variables in your configuration files. In fact, if you review the Laravel configuration files, you will notice several of the options already using this helper:

```
'debug' => env('APP_DEBUG', false),
```

The second value passed to the `env` function is the "default value". This value will be used if no environment variable exists for the given key.

Determining The Current Environment

The current application environment is determined via the `APP_ENV` variable from your `.env` file. You may access this value via the `environment` method on the `App` facade:

```
$environment = App::environment();
```

You may also pass arguments to the `environment` method to check if the environment matches a given value. The method will return `true` if the environment matches any of the given values:

```
if (App::environment('local')) {
    // The environment is local
}

if (App::environment(['local', 'staging'])) {
    // The environment is either local OR staging...
}
```

> 💡💡 The current application environment detection can be overridden by a server-level `APP_ENV` environment variable. This can be useful when you need to share the same application for different environment configurations, so you can set up a given host to match a given environment in your server's configurations. 💡💡

Hiding Environment Variables From Debug Pages

When an exception is uncaught and the `APP_DEBUG` environment variable is `true`, the debug page will show all environment variables and their contents. In some cases you may want to obscure certain variables. You may do this by updating the `debug_hide` option in your `config/app.php` configuration file.

Some variables are available in both the environment variables and the server / request data. Therefore, you may need to hide them for both `$_ENV` and `$_SERVER`:

```
return [

    // ...

    'debug_hide' => [
        '_ENV' => [
            'APP_KEY',
            'DB_PASSWORD',
        ],

        '_SERVER' => [
            'APP_KEY',
            'DB_PASSWORD',
        ],

        '_POST' => [
            'password',
        ],
    ],
];
```

Accessing Configuration Values

You may easily access your configuration values using the global `config` helper function from anywhere in your application. The configuration values may be accessed using "dot" syntax, which includes the name of the file and option you wish to access. A default value may also be specified and will be returned if the configuration option does not exist:

```
$value = config('app.timezone');

// Retrieve a default value if the configuration value does not exist...
$value = config('app.timezone', 'Asia/Seoul');
```

To set configuration values at runtime, pass an array to the `config` helper:

```
config(['app.timezone' => 'America/Chicago']);
```

Configuration Caching

To give your application a speed boost, you should cache all of your configuration files into a single file using the `config:cache` Artisan command. This will combine all of the configuration options for your application into a single file which will be loaded quickly by the framework.

You should typically run the `php artisan config:cache` command as part of your production deployment routine. The command should not be run during local development as configuration options will frequently need to be changed during the course of your application's development.

> ⚠⚠ If you execute the `config:cache` command during your deployment process, you should be sure that you are only calling the `env` function from within your configuration files. Once the configuration has been cached, the `.env` file will not be loaded and all calls to the `env` function will return `null`. ⚠⚠

Maintenance Mode

When your application is in maintenance mode, a custom view will be displayed for all requests into your application. This makes it easy to "disable" your application while it is updating or when you are performing maintenance. A maintenance mode check is included in the default middleware stack for your application. If the application is in maintenance mode, a `MaintenanceModeException` will be thrown with a status code of 503.

To enable maintenance mode, execute the `down` Artisan command:

```
php artisan down
```

You may also provide `message` and `retry` options to the `down` command. The `message` value may be used to display or log a custom message, while the `retry` value will be set as the `Retry-After` HTTP header's value:

```
php artisan down --message="Upgrading Database" --retry=60
```

Bypassing Maintenance Mode

Even while in maintenance mode, you may use use the `secret` option to specify a maintenance mode bypass token:

```
php artisan down --secret="1630542a-246b-4b66-afa1-dd72a4c43515"
```

After placing the application in maintenance mode, you may navigate to the application URL matching this token and Laravel will issue a maintenance mode bypass cookie to your browser:

```
https://example.com/1630542a-246b-4b66-afa1-dd72a4c43515
```

When accessing this hidden route, you will then be redirected to the `/` route of the application. Once the cookie has been issued to your browser, you will be able to browse the application normally as if it was not in maintenance mode.

Pre-Rendering The Maintenace Mode View

If you utilize the `php artisan down` command during deployment, your users may still occasionally encounter errors if they access the application while your Composer dependencies or other infrastructure components are updating. This occurs because a significant part of the Laravel framework must boot in order to determine your application is in maintenance mode and render the maintenance mode view using the templating engine.

For this reason, Laravel allows you to pre-render a maintenance mode view that will be returned at the very beginning of the request cycle. This view is rendered before any of your application's dependencies have loaded. You may pre-render a template of your choice using the `down` command's `render` option:

```
php artisan down --render="errors::503"
```

Redirecting Maintenance Mode Requests

While in maintenance mode, Laravel will display the maintenance mode view for all application URLs the user attempts to access. If you wish, you may instruct Laravel to redirect all requests to a specific URL. This may be accomplished using the `redirect` option. For example, you may wish to redirect all requests to the `/` URI:

```
php artisan down --redirect=/
```

Disabling Maintenance Mode

To disable maintenance mode, use the `up` command:

```
php artisan up
```

> 💡💡 You may customize the default maintenance mode template by defining your own template at `resources/views/errors/503.blade.php`. 💡💡

Maintenance Mode & Queues

While your application is in maintenance mode, no queued jobs will be handled. The jobs will continue to be handled as normal once the application is out of maintenance mode.

Alternatives To Maintenance Mode

Since maintenance mode requires your application to have several seconds of downtime, consider alternatives like Envoyer to accomplish zero-downtime deployment with Laravel.

Directory Structure

Introduction

The default Laravel application structure is intended to provide a great starting point for both large and small applications. But you are free to organize your application however you like. Laravel imposes almost no restrictions on where any given class is located - as long as Composer can autoload the class.

The Root Directory

The App Directory

The `app` directory contains the core code of your application. We'll explore this directory in more detail soon; however, almost all of the classes in your application will be in this directory.

The Bootstrap Directory

The `bootstrap` directory contains the `app.php` file which bootstraps the framework. This directory also houses a `cache` directory which contains framework generated files for performance optimization such as the route and services cache files.

The Config Directory

The `config` directory, as the name implies, contains all of your application's configuration files. It's a great idea to read through all of these files and familiarize yourself with all of the options available to you.

The Database Directory

The `database` directory contains your database migrations, model factories, and seeds. If you wish, you may also use this directory to hold an SQLite database.

The Public Directory

The `public` directory contains the `index.php` file, which is the entry point for all requests entering your application and configures autoloading. This directory also houses your assets such as images, JavaScript, and CSS.

The Resources Directory

The `resources` directory contains your views as well as your raw, un-compiled assets such as LESS, SASS, or JavaScript. This directory also houses all of your language files.

The Routes Directory

The `routes` directory contains all of the route definitions for your application. By default, several route files are included with Laravel: `web.php`, `api.php`, `console.php` and `channels.php`.

The `web.php` file contains routes that the `RouteServiceProvider` places in the `web` middleware group, which provides session state, CSRF protection, and cookie encryption. If your application does not offer a stateless, RESTful API, all of your routes will most likely be defined in the `web.php` file.

The `api.php` file contains routes that the `RouteServiceProvider` places in the `api` middleware group, which provides rate limiting. These routes are intended to be stateless, so requests entering the application through these routes are intended to be authenticated via tokens and will not have access to session state.

The `console.php` file is where you may define all of your Closure based console commands. Each Closure is bound to a command instance allowing a simple approach to interacting with each command's IO methods. Even though this file does not define HTTP routes, it defines console based entry points (routes) into your application.

The `channels.php` file is where you may register all of the event broadcasting channels that your application supports.

The Storage Directory

The `storage` directory contains your compiled Blade templates, file based sessions, file caches, and other files generated by the framework. This directory is segregated into `app`, `framework`, and `logs` directories. The `app` directory may be used to store any files generated by your application. The `framework` directory is used to store framework generated files and caches. Finally, the `logs` directory contains your application's log files.

The `storage/app/public` directory may be used to store user-generated files, such as profile avatars, that should be publicly accessible. You should create a symbolic link at `public/storage` which points to this directory. You may create the link using the `php artisan storage:link` command.

The Tests Directory

The `tests` directory contains your automated tests. An example PHPUnit test is provided out of the box. Each test class should be suffixed with the word `Test`. You may run your tests using the `phpunit` or `php vendor/bin/phpunit` commands.

The Vendor Directory

The `vendor` directory contains your Composer dependencies.

The App Directory

The majority of your application is housed in the `app` directory. By default, this directory is namespaced under `App` and is autoloaded by Composer using the PSR-4 autoloading standard.

The `app` directory contains a variety of additional directories such as `Console`, `Http`, and `Providers`. Think of the `Console` and `Http` directories as providing an API into the core of your application. The HTTP protocol and CLI are both mechanisms to interact with your application, but do not actually contain application logic. In other words, they are two ways of issuing commands to your application. The `Console` directory contains all of your Artisan commands, while the `Http` directory contains your controllers, middleware, and requests.

A variety of other directories will be generated inside the `app` directory as you use the `make` Artisan commands to generate classes. So, for example, the `app/Jobs` directory will not exist until you execute the `make:job` Artisan command to generate a job class.

> 💡💡 Many of the classes in the `app` directory can be generated by Artisan via commands. To review the available commands, run the `php artisan list make` command in your terminal. 💡💡

The Broadcasting Directory

The `Broadcasting` directory contains all of the broadcast channel classes for your application. These classes are generated using the `make:channel` command. This directory does not exist by default, but will be created for you when you create your first channel. To learn more about channels, check out the documentation on event broadcasting.

The Console Directory

The `Console` directory contains all of the custom Artisan commands for your application. These commands may be generated using the `make:command` command. This directory also houses your console kernel, which is where your custom Artisan commands are registered and your scheduled tasks are defined.

The Events Directory

This directory does not exist by default, but will be created for you by the `event:generate` and `make:event` Artisan commands. The `Events` directory houses event classes. Events may be used to alert other parts of your application that a given action has occurred, providing a great deal of flexibility and decoupling.

The Exceptions Directory

The `Exceptions` directory contains your application's exception handler and is also a good place to place any exceptions thrown by your application. If you would like to customize how your exceptions are logged or rendered, you should modify the `Handler` class in this directory.

The Http Directory

The `Http` directory contains your controllers, middleware, and form requests. Almost all of the logic to handle requests entering your application will be placed in this directory.

The Jobs Directory

This directory does not exist by default, but will be created for you if you execute the `make:job` Artisan command. The `Jobs` directory houses the queueable jobs for your application. Jobs may be queued by your application or run synchronously within the current request lifecycle. Jobs that run synchronously during the current request are sometimes referred to as "commands" since they are an implementation of the command pattern.

The Listeners Directory

This directory does not exist by default, but will be created for you if you execute the `event:generate` or `make:listener` Artisan commands. The `Listeners` directory contains the classes that handle your events. Event listeners receive an event instance and perform logic in response to the event being fired. For example, a `UserRegistered` event might be handled by a `SendWelcomeEmail` listener.

The Mail Directory

This directory does not exist by default, but will be created for you if you execute the `make:mail` Artisan command. The `Mail` directory contains all of your classes that represent emails sent by your application. Mail objects allow you to encapsulate all of the logic of building an email in a single, simple class that may be sent using the `Mail::send` method.

The Models Directory

The `Models` directory contains all of your Eloquent model classes. The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.

The Notifications Directory

This directory does not exist by default, but will be created for you if you execute the `make:notification` Artisan command. The `Notifications` directory contains all of the "transactional" notifications that are sent by your application, such as simple notifications about events that happen within your application. Laravel's notification features abstracts sending notifications over a variety of drivers such as email, Slack, SMS, or stored in a database.

The Policies Directory

This directory does not exist by default, but will be created for you if you execute the `make:policy` Artisan command. The `Policies` directory contains the authorization policy classes for your application. Policies are used to determine if a user can perform a given action against a resource. For more information, check out the authorization documentation.

The Providers Directory

The `Providers` directory contains all of the service providers for your application. Service providers bootstrap your application by binding services in the service container, registering events, or performing any other tasks to prepare your application for incoming requests.

In a fresh Laravel application, this directory will already contain several providers. You are free to add your own providers to this directory as needed.

The Rules Directory

This directory does not exist by default, but will be created for you if you execute the `make:rule` Artisan command. The `Rules` directory contains the custom validation rule objects for your application. Rules are used to encapsulate complicated validation logic in a simple object. For more information, check out the validation documentation.

Laravel Homestead

Introduction

Laravel strives to make the entire PHP development experience delightful, including your local development environment. Vagrant provides a simple, elegant way to manage and provision Virtual Machines.

Laravel Homestead is an official, pre-packaged Vagrant box that provides you a wonderful development environment without requiring you to install PHP, a web server, and any other server software on your local machine. No more worrying about messing up your operating system! Vagrant boxes are completely disposable. If something goes wrong, you can destroy and re-create the box in minutes!

Homestead runs on any Windows, Mac, or Linux system, and includes Nginx, PHP, MySQL, PostgreSQL, Redis, Memcached, Node, and all of the other goodies you need to develop amazing Laravel applications.

> ⚠⚠ If you are using Windows, you may need to enable hardware virtualization (VT-x). It can usually be enabled via your BIOS. If you are using Hyper-V on a UEFI system you may additionally need to disable Hyper-V in order to access VT-x. ⚠⚠

Included Software

- Ubuntu 18.04
- Git
- PHP 7.4
- PHP 7.3
- PHP 7.2
- PHP 7.1
- PHP 7.0
- PHP 5.6
- Nginx
- MySQL
- lmm for MySQL or MariaDB database snapshots
- Sqlite3
- PostgreSQL (9.6, 10, 11, 12)
- Composer
- Node (With Yarn, Bower, Grunt, and Gulp)
- Redis
- Memcached
- Beanstalkd
- Mailhog
- avahi
- ngrok
- Xdebug
- XHProf / Tideways / XHGui
- wp-cli

Optional Software

- Apache
- Blackfire
- Cassandra
- Chronograf
- CouchDB
- Crystal & Lucky Framework
- Docker
- Elasticsearch
- Gearman
- Go
- Grafana
- InfluxDB
- MariaDB
- MinIO
- MongoDB
- MySQL 8
- Neo4j
- Oh My Zsh
- Open Resty
- PM2
- Python
- RabbitMQ
- Solr
- Webdriver & Laravel Dusk Utilities

Installation & Setup

First Steps

Before launching your Homestead environment, you must install VirtualBox 6.x, VMWare, Parallels or Hyper-V as well as Vagrant. All of these software packages provide easy-to-use visual installers for all popular operating systems.

To use the VMware provider, you will need to purchase both VMware Fusion / Workstation and the VMware Vagrant plug-in. Though it is not free, VMware can provide faster shared folder performance out of the box.

To use the Parallels provider, you will need to install Parallels Vagrant plug-in. It is free of charge.

Because of Vagrant limitations, the Hyper-V provider ignores all networking settings.

Installing The Homestead Vagrant Box

Once VirtualBox / VMware and Vagrant have been installed, you should add the `laravel/homestead` box to your Vagrant installation using the following command in your terminal. It will take a few minutes to download the box, depending on your Internet connection speed:

```
vagrant box add laravel/homestead
```

If this command fails, make sure your Vagrant installation is up to date.

> ⚠⚠ Homestead periodically issues "alpha" / "beta" boxes for testing, which may interfere with the `vagrant box add` command. If you are having issues running `vagrant box add`, you may run the `vagrant up` command and the correct box will be downloaded when Vagrant attempts to start the virtual machine. ⚠⚠

Installing Homestead

You may install Homestead by cloning the repository onto your host machine. Consider cloning the repository into a `Homestead` folder within your "home" directory, as the Homestead box will serve as the host to all of your Laravel projects:

```
git clone https://github.com/laravel/homestead.git ~/Homestead
```

You should check out a tagged version of Homestead since the `master` branch may not always be stable. You can find the latest stable version on the GitHub Release Page. Alternatively, you may checkout the `release` branch which always contains the latest stable release:

```
cd ~/Homestead

git checkout release
```

Once you have cloned the Homestead repository, run the `bash init.sh` command from the Homestead directory to create the `Homestead.yaml` configuration file. The `Homestead.yaml` file will be placed in the Homestead directory:

```
// Mac / Linux...
bash init.sh

// Windows...
init.bat
```

Configuring Homestead

Setting Your Provider

The `provider` key in your `Homestead.yaml` file indicates which Vagrant provider should be used: `virtualbox`, `vmware_fusion`, `vmware_workstation`, `parallels` or `hyperv`. You may set this to the provider you prefer:

```
provider: virtualbox
```

Configuring Shared Folders

The `folders` property of the `Homestead.yaml` file lists all of the folders you wish to share with your Homestead environment. As files within these folders are changed, they will be kept in sync between your local machine and the Homestead environment. You may configure as many shared folders as necessary:

```
folders:
    - map: ~/code/project1
      to: /home/vagrant/project1
```

> ⚠⚠ Windows users should not use the `~/` path syntax and instead should use the full path to their project, such as `C:\Users\user\Code\project1`. ⚠⚠

You should always map individual projects to their own folder mapping instead of mapping your entire `~/code` folder. When you map a folder the virtual machine must keep track of all disk IO for *every* file in the folder. This leads to performance issues if you have a large number of files in a folder.

```
folders:
    - map: ~/code/project1
      to: /home/vagrant/project1

    - map: ~/code/project2
      to: /home/vagrant/project2
```

> ⚠⚠ You should never mount `.` (the current directory) when using Homestead. This causes Vagrant to not map the current folder to `/vagrant` and will break optional features and cause unexpected results while provisioning. ⚠⚠

To enable NFS, you only need to add a simple flag to your synced folder configuration:

```
folders:
    - map: ~/code/project1
      to: /home/vagrant/project1
      type: "nfs"
```

> ⚠⚠ When using NFS on Windows, you should consider installing the vagrant-winnfsd plug-in. This plug-in will maintain the correct user / group permissions for files and directories within the Homestead box. ⚠⚠

You may also pass any options supported by Vagrant's Synced Folders by listing them under the `options` key:

```
folders:
    - map: ~/code/project1
      to: /home/vagrant/project1
      type: "rsync"
      options:
          rsync__args: ["--verbose", "--archive", "--delete", "-zz"]
          rsync__exclude: ["node_modules"]
```

Configuring Nginx Sites

Not familiar with Nginx? No problem. The `sites` property allows you to easily map a "domain" to a folder on your Homestead environment. A sample site configuration is included in the `Homestead.yaml` file. Again, you may add as

many sites to your Homestead environment as necessary. Homestead can serve as a convenient, virtualized environment for every Laravel project you are working on:

```
sites:
    - map: homestead.test
      to: /home/vagrant/project1/public
```

If you change the `sites` property after provisioning the Homestead box, you should re-run `vagrant reload --provision` to update the Nginx configuration on the virtual machine.

> ⚠⚠ Homestead scripts are built to be as idempotent as possible. However, if you are experiencing issues while provisioning you should destroy and rebuild the machine via `vagrant destroy && vagrant up`. ⚠⚠

Enable / Disable Services

Homestead starts several services by default; however, you may customize which services are enabled or disabled during provisioning. For example, you may enable PostgreSQL and disable MySQL:

```
services:
    - enabled:
        - "postgresql@12-main"
    - disabled:
        - "mysql"
```

The specified services will be started or stopped based on their order in the `enabled` and `disabled` directives.

Hostname Resolution

Homestead publishes hostnames over `mDNS` for automatic host resolution. If you set `hostname: homestead` in your `Homestead.yaml` file, the host will be available at `homestead.local`. MacOS, iOS, and Linux desktop distributions include `mDNS` support by default. Windows requires installing Bonjour Print Services for Windows.

Using automatic hostnames works best for "per project" installations of Homestead. If you host multiple sites on a single Homestead instance, you may add the "domains" for your web sites to the `hosts` file on your machine. The `hosts` file will redirect requests for your Homestead sites into your Homestead machine. On Mac and Linux, this file is located at `/etc/hosts`. On Windows, it is located at `C:\Windows\System32\drivers\etc\hosts`. The lines you add to this file will look like the following:

```
192.168.10.10  homestead.test
```

Make sure the IP address listed is the one set in your `Homestead.yaml` file. Once you have added the domain to your `hosts` file and launched the Vagrant box you will be able to access the site via your web browser:

```
http://homestead.test
```

Launching The Vagrant Box

Once you have edited the `Homestead.yaml` to your liking, run the `vagrant up` command from your Homestead directory. Vagrant will boot the virtual machine and automatically configure your shared folders and Nginx sites.

To destroy the machine, you may use the `vagrant destroy --force` command.

Per Project Installation

Instead of installing Homestead globally and sharing the same Homestead box across all of your projects, you may instead configure a Homestead instance for each project you manage. Installing Homestead per project may be beneficial if you wish to ship a `Vagrantfile` with your project, allowing others working on the project to `vagrant up`.

To install Homestead directly into your project, require it using Composer:

```
composer require laravel/homestead --dev
```

Once Homestead has been installed, use the `make` command to generate the `Vagrantfile` and `Homestead.yaml` file in your project root. The `make` command will automatically configure the `sites` and `folders` directives in the `Homestead.yaml` file.

Mac / Linux:

```
php vendor/bin/homestead make
```

Windows:

```
vendor\\bin\\homestead make
```

Next, run the `vagrant up` command in your terminal and access your project at `http://homestead.test` in your browser. Remember, you will still need to add an `/etc/hosts` file entry for `homestead.test` or the domain of your choice if you are not using automatic hostname resolution.

Installing Optional Features

Optional software is installed using the "features" setting in your Homestead configuration file. Most features can be enabled or disabled with a boolean value, while some features allow multiple configuration options:

```
features:
    - blackfire:
        server_id: "server_id"
        server_token: "server_value"
        client_id: "client_id"
        client_token: "client_value"
    - cassandra: true
    - chronograf: true
    - couchdb: true
    - crystal: true
    - docker: true
    - elasticsearch:
        version: 7.9.0
    - gearman: true
```

```
    - golang: true
    - grafana: true
    - influxdb: true
    - mariadb: true
    - minio: true
    - mongodb: true
    - mysql8: true
    - neo4j: true
    - ohmyzsh: true
    - openresty: true
    - pm2: true
    - python: true
    - rabbitmq: true
    - solr: true
    - webdriver: true
```

MariaDB

Enabling MariaDB will remove MySQL and install MariaDB. MariaDB serves as a drop-in replacement for MySQL, so you should still use the `mysql` database driver in your application's database configuration.

MongoDB

The default MongoDB installation will set the database username to `homestead` and the corresponding password to `secret`.

Elasticsearch

You may specify a supported version of Elasticsearch, which may be a major version or an exact version number (major.minor.patch). The default installation will create a cluster named 'homestead'. You should never give Elasticsearch more than half of the operating system's memory, so make sure your Homestead machine has at least twice the Elasticsearch allocation.

> 💡💡 Check out the Elasticsearch documentation to learn how to customize your configuration. 💡💡

Neo4j

The default Neo4j installation will set the database username to `homestead` and corresponding password to `secret`. To access the Neo4j browser, visit `http://homestead.test:7474` via your web browser. The ports `7687` (Bolt), `7474` (HTTP), and `7473` (HTTPS) are ready to serve requests from the Neo4j client.

Aliases

You may add Bash aliases to your Homestead machine by modifying the `aliases` file within your Homestead directory:

```
alias c='clear'
alias ..='cd ..'
```

After you have updated the `aliases` file, you should re-provision the Homestead machine using the `vagrant reload --provision` command. This will ensure that your new aliases are available on the machine.

Daily Usage

Accessing Homestead Globally

Sometimes you may want to `vagrant up` your Homestead machine from anywhere on your filesystem. You can do this on Mac / Linux systems by adding a Bash function to your Bash profile. On Windows, you may accomplish this by adding a "batch" file to your `PATH`. These scripts will allow you to run any Vagrant command from anywhere on your system and will automatically point that command to your Homestead installation:

Mac / Linux

```
function homestead() {
    ( cd ~/Homestead && vagrant $* )
}
```

Make sure to tweak the `~/Homestead` path in the function to the location of your actual Homestead installation. Once the function is installed, you may run commands like `homestead up` or `homestead ssh` from anywhere on your system.

Windows

Create a `homestead.bat` batch file anywhere on your machine with the following contents:

```
@echo off

set cwd=%cd%
set homesteadVagrant=C:\Homestead

cd /d %homesteadVagrant% && vagrant %*
cd /d %cwd%

set cwd=
set homesteadVagrant=
```

Make sure to tweak the example `C:\Homestead` path in the script to the actual location of your Homestead installation. After creating the file, add the file location to your `PATH`. You may then run commands like `homestead up` or `homestead ssh` from anywhere on your system.

Connecting Via SSH

You can SSH into your virtual machine by issuing the `vagrant ssh` terminal command from your Homestead directory.

But, since you will probably need to SSH into your Homestead machine frequently, consider adding the "function" described above to your host machine to quickly SSH into the Homestead box.

Connecting To Databases

A `homestead` database is configured for both MySQL and PostgreSQL out of the box. To connect to your MySQL or PostgreSQL database from your host machine's database client, you should connect to `127.0.0.1` and port `33060` (MySQL) or `54320` (PostgreSQL). The username and password for both databases is `homestead` / `secret`.

> ⚠⚠ You should only use these non-standard ports when connecting to the databases from your host machine. You will use the default 3306 and 5432 ports in your Laravel database configuration file since Laravel is running *within* the virtual machine. ⚠⚠

Database Backups

Homestead can automatically backup your database when your Vagrant box is destroyed. To utilize this feature, you must be using Vagrant 2.1.0 or greater. Or, if you are using an older version of Vagrant, you must install the `vagrant-triggers` plug-in. To enable automatic database backups, add the following line to your `Homestead.yaml` file:

```
backup: true
```

Once configured, Homestead will export your databases to `mysql_backup` and `postgres_backup` directories when the `vagrant destroy` command is executed. These directories can be found in the folder where you cloned Homestead or in the root of your project if you are using the per project installation method.

Database Snapshots

Homestead supports freezing the state of MySQL and MariaDB databases and branching between them using Logical MySQL Manager. For example, imagine working on a site with a multi-gigabyte database. You can import the database and take a snapshot. After doing some work and creating some test content locally, you may quickly restore back to the original state.

Under the hood, LMM uses LVM's thin snapshot functionality with copy-on-write support. In practice, this means that changing a single row in a table will only cause the changes you made to be written to disk, saving significant time and disk space during restores.

Since `lmm` interacts with LVM, it must be run as `root`. To see all available commands, run `sudo lmm` inside your Vagrant box. A common workflow looks like the following:

1. Import a database into the default `master` lmm branch.
2. Save a snapshot of the unchanged database using `sudo lmm branch prod-YYYY-MM-DD`.
3. Modify the database.
4. Run `sudo lmm merge prod-YYYY-MM-DD` to undo all changes.
5. Run `sudo lmm delete <branch>` to delete unneeded branches.

Adding Additional Sites

Once your Homestead environment is provisioned and running, you may want to add additional Nginx sites for your Laravel applications. You can run as many Laravel installations as you wish on a single Homestead environment. To add an additional site, add the site to your `Homestead.yaml` file:

```
sites:
    - map: homestead.test
      to: /home/vagrant/project1/public
    - map: another.test
      to: /home/vagrant/project2/public
```

If Vagrant is not automatically managing your "hosts" file, you may need to add the new site to that file as well:

```
192.168.10.10  homestead.test
192.168.10.10  another.test
```

Once the site has been added, run the `vagrant reload --provision` command from your Homestead directory.

Site Types

Homestead supports several types of sites which allow you to easily run projects that are not based on Laravel. For example, we may easily add a Symfony application to Homestead using the `symfony2` site type:

```
sites:
    - map: symfony2.test
      to: /home/vagrant/my-symfony-project/web
      type: "symfony2"
```

The available site types are: `apache`, `apigility`, `expressive`, `laravel` (the default), `proxy`, `silverstripe`, `statamic`, `symfony2`, `symfony4`, and `zf`.

Site Parameters

You may add additional Nginx `fastcgi_param` values to your site via the `params` site directive. For example, we'll add a `FOO` parameter with a value of `BAR`:

```
sites:
    - map: homestead.test
      to: /home/vagrant/project1/public
      params:
          - key: FOO
            value: BAR
```

Environment Variables

You can set global environment variables by adding them to your `Homestead.yaml` file:

```
variables:
    - key: APP_ENV
      value: local
    - key: FOO
      value: bar
```

After updating the `Homestead.yaml`, be sure to re-provision the machine by running `vagrant reload --provision`. This will update the PHP-FPM configuration for all of the installed PHP versions and also update the environment for the `vagrant` user.

Wildcard SSL

Homestead configures a self-signed SSL certificate for each site defined in the `sites:` section of your `Homestead.yaml` file. If you would like to generate a wildcard SSL certificate for a site you may add a `wildcard` option to that site's configuration. By default, the site will use the wildcard certificate *instead* of the specific domain certificate:

```
- map: foo.domain.test
  to: /home/vagrant/domain
  wildcard: "yes"
```

If the `use_wildcard` option is set to `no`, the wildcard certificate will be generated but will not be used:

```
- map: foo.domain.test
  to: /home/vagrant/domain
  wildcard: "yes"
  use_wildcard: "no"
```

Configuring Cron Schedules

Laravel provides a convenient way to schedule Cron jobs by scheduling a single `schedule:run` Artisan command to be run every minute. The `schedule:run` command will examine the job schedule defined in your `App\Console\Kernel` class to determine which jobs should be run.

If you would like the `schedule:run` command to be run for a Homestead site, you may set the `schedule` option to `true` when defining the site:

```
sites:
    - map: homestead.test
      to: /home/vagrant/project1/public
      schedule: true
```

The Cron job for the site will be defined in the `/etc/cron.d` folder of the virtual machine.

Configuring Mailhog

Mailhog allows you to easily catch your outgoing email and examine it without actually sending the mail to its recipients. To get started, update your `.env` file to use the following mail settings:

```
MAIL_MAILER=smtp
MAIL_HOST=localhost
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
```

Once Mailhog has been configured, you may access the Mailhog dashboard at `http://localhost:8025`.

Configuring Minio

Minio is an open source object storage server with an Amazon S3 compatible API. To install Minio, update your `Homestead.yaml` file with the following configuration option in the features section:

```
minio: true
```

By default, Minio is available on port 9600. You may access the Minio control panel by visiting `http://localhost:9600/`. The default access key is `homestead`, while the default secret key is `secretkey`. When accessing Minio, you should always use region `us-east-1`.

In order to use Minio you will need to adjust the S3 disk configuration in your `config/filesystems.php` configuration file. You will need to add the `use_path_style_endpoint` option to the disk configuration, as well as change the `url` key to `endpoint`:

```
's3' => [
    'driver' => 's3',
    'key' => env('AWS_ACCESS_KEY_ID'),
    'secret' => env('AWS_SECRET_ACCESS_KEY'),
    'region' => env('AWS_DEFAULT_REGION'),
    'bucket' => env('AWS_BUCKET'),
    'endpoint' => env('AWS_URL'),
    'use_path_style_endpoint' => true,
]
```

Finally, ensure your `.env` file has the following options:

```
AWS_ACCESS_KEY_ID=homestead
AWS_SECRET_ACCESS_KEY=secretkey
AWS_DEFAULT_REGION=us-east-1
AWS_URL=http://localhost:9600
```

To provision buckets, add a `buckets` directive to your Homestead configuration file:

```
buckets:
    - name: your-bucket
      policy: public
    - name: your-private-bucket
      policy: none
```

Supported `policy` values include: `none`, `download`, `upload`, and `public`.

Ports

By default, the following ports are forwarded to your Homestead environment:

- **SSH:** 2222 → Forwards To 22
- **ngrok UI:** 4040 → Forwards To 4040
- **HTTP:** 8000 → Forwards To 80
- **HTTPS:** 44300 → Forwards To 443
- **MySQL:** 33060 → Forwards To 3306
- **PostgreSQL:** 54320 → Forwards To 5432
- **MongoDB:** 27017 → Forwards To 27017
- **Mailhog:** 8025 → Forwards To 8025
- **Minio:** 9600 → Forwards To 9600

Forwarding Additional Ports

If you wish, you may forward additional ports to the Vagrant box, as well as specify their protocol:

```
ports:
    - send: 50000
      to: 5000
    - send: 7777
      to: 777
      protocol: udp
```

Sharing Your Environment

Sometimes you may wish to share what you're currently working on with coworkers or a client. Vagrant has a built-in way to support this via `vagrant share`; however, this will not work if you have multiple sites configured in your `Homestead.yaml` file.

To solve this problem, Homestead includes its own `share` command. To get started, SSH into your Homestead machine via `vagrant ssh` and run `share homestead.test`. This will share the `homestead.test` site from your `Homestead.yaml` configuration file. You may substitute any of your other configured sites for `homestead.test`:

```
share homestead.test
```

After running the command, you will see an Ngrok screen appear which contains the activity log and the publicly accessible URLs for the shared site. If you would like to specify a custom region, subdomain, or other Ngrok runtime option, you may add them to your `share` command:

```
share homestead.test -region=eu -subdomain=laravel
```

> ⚠⚠ Remember, Vagrant is inherently insecure and you are exposing your virtual machine to the Internet when running the `share` command. ⚠⚠

Multiple PHP Versions

Homestead 6 introduced support for multiple versions of PHP on the same virtual machine. You may specify which version of PHP to use for a given site within your `Homestead.yaml` file. The available PHP versions are: "5.6", "7.0", "7.1", "7.2", "7.3", and "7.4" (the default):

```
sites:
    - map: homestead.test
      to: /home/vagrant/project1/public
      php: "7.1"
```

In addition, you may use any of the supported PHP versions via the CLI:

```
php5.6 artisan list
php7.0 artisan list
php7.1 artisan list
php7.2 artisan list
php7.3 artisan list
php7.4 artisan list
```

You may also update the default CLI version by issuing the following commands from within your Homestead virtual machine:

```
php56
php70
php71
php72
php73
php74
```

Web Servers

Homestead uses the Nginx web server by default. However, it can install Apache if `apache` is specified as a site type. While both web servers can be installed at the same time, they cannot both be *running* at the same time. The `flip` shell command is available to ease the process of switching between web servers. The `flip` command automatically determines which web server is running, shuts it off, and then starts the other server. To use this command, SSH into your Homestead machine and run the command in your terminal:

```
flip
```

Mail

Homestead includes the Postfix mail transfer agent, which is listening on port `1025` by default. So, you may instruct your application to use the `smtp` mail driver on `localhost` port `1025`. Then, all sent mail will be handled by Postfix and caught by Mailhog. To view your sent emails, open http://localhost:8025 in your web browser.

Debugging & Profiling

Debugging Web Requests With Xdebug

Homestead includes support for step debugging using Xdebug. For example, you can load a web page from a browser, and PHP will connect to your IDE to allow inspection and modification of the running code.

By default Xdebug is already running and ready to accept connections. If you need to enable Xdebug on the CLI run the `sudo phpenmod xdebug` command within your Vagrant box. Next, follow your IDE's instructions to enable debugging. Finally, configure your browser to trigger Xdebug with an extension or bookmarklet.

> ⚠⚠ Xdebug causes PHP to run significantly slower. To disable Xdebug, run `sudo phpdismod xdebug` within your Vagrant box and restart the FPM service. ⚠⚠

Debugging CLI Applications

To debug a PHP CLI application, use the `xphp` shell alias inside your Vagrant box:

```
xphp path/to/script
```

Autostarting Xdebug

When debugging functional tests that make requests to the web server, it is easier to autostart debugging rather than modifying tests to pass through a custom header or cookie to trigger debugging. To force Xdebug to start automatically, modify `/etc/php/7.x/fpm/conf.d/20-xdebug.ini` inside your Vagrant box and add the following configuration:

```
; If Homestead.yaml contains a different subnet for the IP address, this address may be different...
xdebug.remote_host = 192.168.10.1
xdebug.remote_autostart = 1
```

Profiling Applications with Blackfire

Blackfire is a SaaS service for profiling web requests and CLI applications and writing performance assertions. It offers an interactive user interface which displays profile data in call-graphs and timelines. It is built for use in development, staging, and production, with no overhead for end users. It provides performance, quality, and security checks on code and `php.ini` configuration settings.

The Blackfire Player is an open-source Web Crawling, Web Testing and Web Scraping application which can work jointly with Blackfire in order to script profiling scenarios.

To enable Blackfire, use the "features" setting in your Homestead configuration file:

```
features:
    - blackfire:
        server_id: "server_id"
        server_token: "server_value"
        client_id: "client_id"
        client_token: "client_value"
```

Blackfire server credentials and client credentials require a user account. Blackfire offers various options to profile an application, including a CLI tool and browser extension. Please review the Blackfire documentation for more details.

Profiling PHP Performance Using XHGui

XHGui is a user interface for exploring the performance of your PHP applications. To enable XHGui, add `xhgui: 'true'` to your site configuration:

```
sites:
    -
        map: your-site.test
        to: /home/vagrant/your-site/public
        type: "apache"
        xhgui: 'true'
```

If the site already exists, make sure to run `vagrant provision` after updating your configuration.

To profile a web request, add `xhgui=on` as a query parameter to a request. XHGui will automatically attach a cookie to the response so that subsequent requests do not need the query string value. You may view your application profile results by browsing to `http://your-site.test/xhgui`.

To profile a CLI request using XHGui, prefix the command with `XHGUI=on`:

```
XHGUI=on path/to/script
```

CLI profile results may be viewed in the same way as web profile results.

Note that the act of profiling slows down script execution, and absolute times may be as much as twice as real-world requests. Therefore, always compare percentage improvements and not absolute numbers. Also, be aware the execution time includes any time spent paused in a debugger.

Since performance profiles take up significant disk space, they are deleted automatically after a few days.

Network Interfaces

The `networks` property of the `Homestead.yaml` configures network interfaces for your Homestead environment. You may configure as many interfaces as necessary:

```
networks:
    - type: "private_network"
      ip: "192.168.10.20"
```

To enable a bridged interface, configure a `bridge` setting and change the network type to `public_network`:

```
networks:
    - type: "public_network"
      ip: "192.168.10.20"
      bridge: "en1: Wi-Fi (AirPort)"
```

To enable DHCP, just remove the `ip` option from your configuration:

```
networks:
    - type: "public_network"
      bridge: "en1: Wi-Fi (AirPort)"
```

Extending Homestead

You may extend Homestead using the `after.sh` script in the root of your Homestead directory. Within this file, you may add any shell commands that are necessary to properly configure and customize your virtual machine.

When customizing Homestead, Ubuntu may ask you if you would like to keep a package's original configuration or overwrite it with a new configuration file. To avoid this, you should use the following command when installing packages to avoid overwriting any configuration previously written by Homestead:

```
sudo apt-get -y \
    -o Dpkg::Options::="--force-confdef" \
    -o Dpkg::Options::="--force-confold" \
    install your-package
```

User Customizations

When using Homestead in a team setting, you may want to tweak Homestead to better fit your personal development style. You may create a `user-customizations.sh` file in the root of your Homestead directory (The same directory containing your `Homestead.yaml`). Within this file, you may make any customization you would like; however, the `user-customizations.sh` should not be version controlled.

Updating Homestead

Before you begin updating Homestead ensure you have removed your current virtual machine by running the following command in your Homestead directory:

```
vagrant destroy
```

Next, you need to update the Homestead source code. If you cloned the repository you can run the following commands at the location you originally cloned the repository:

```
git fetch

git pull origin release
```

These commands pull the latest Homestead code from the GitHub repository, fetches the latest tags, and then checks out the latest tagged release. You can find the latest stable release version on the GitHub releases page.

If you have installed Homestead via your project's `composer.json` file, you should ensure your `composer.json` file contains `"laravel/homestead": "^11"` and update your dependencies:

```
composer update
```

Then, you should update the Vagrant box using the `vagrant box update` command:

```
vagrant box update
```

Next, you should run the `bash init.sh` command from the Homestead directory in order to update some additional configuration files. You will be asked whether you wish to overwrite your existing `Homestead.yaml`, `after.sh`, and `aliases` files:

```
// Mac / Linux...
bash init.sh

// Windows...
init.bat
```

Finally, you will need to regenerate your Homestead box to utilize the latest Vagrant installation:

```
vagrant up
```

Provider Specific Settings

VirtualBox

`natdnshostresolver`

By default, Homestead configures the `natdnshostresolver` setting to `on`. This allows Homestead to use your host operating system's DNS settings. If you would like to override this behavior, add the following lines to your `Homestead.yaml` file:

```
provider: virtualbox
natdnshostresolver: 'off'
```

Symbolic Links On Windows

If symbolic links are not working properly on your Windows machine, you may need to add the following block to your `Vagrantfile`:

```
config.vm.provider "virtualbox" do |v|
    v.customize ["setextradata", :id, "VBoxInternal2/SharedFoldersEnableSymlinksCreate/v-root", "1"]
end
```

Laravel Valet

Introduction

Valet is a Laravel development environment for Mac minimalists. No Vagrant, no `/etc/hosts` file. You can even share your sites publicly using local tunnels. *Yeah, we like it too.*

Laravel Valet configures your Mac to always run Nginx in the background when your machine starts. Then, using DnsMasq, Valet proxies all requests on the `*.test` domain to point to sites installed on your local machine.

In other words, a blazing fast Laravel development environment that uses roughly 7 MB of RAM. Valet isn't a complete replacement for Vagrant or Homestead, but provides a great alternative if you want flexible basics, prefer extreme speed, or are working on a machine with a limited amount of RAM.

Out of the box, Valet support includes, but is not limited to:

- Laravel
- Lumen
- Bedrock
- CakePHP 3
- Concrete5
- Contao
- Craft
- Drupal
- ExpressionEngine
- Jigsaw
- Joomla
- Katana
- Kirby
- Magento
- OctoberCMS
- Sculpin
- Slim
- Statamic
- Static HTML
- Symfony
- WordPress
- Zend

However, you may extend Valet with your own custom drivers.

Valet Or Homestead

As you may know, Laravel offers Homestead, another local Laravel development environment. Homestead and Valet differ in regards to their intended audience and their approach to local development. Homestead offers an entire

Ubuntu virtual machine with automated Nginx configuration. Homestead is a wonderful choice if you want a fully virtualized Linux development environment or are on Windows / Linux.

Valet only supports Mac, and requires you to install PHP and a database server directly onto your local machine. This is easily achieved by using Homebrew with commands like `brew install php` and `brew install mysql`. Valet provides a blazing fast local development environment with minimal resource consumption, so it's great for developers who only require PHP / MySQL and do not need a fully virtualized development environment.

Both Valet and Homestead are great choices for configuring your Laravel development environment. Which one you choose will depend on your personal taste and your team's needs.

Installation

Valet requires macOS and Homebrew. Before installation, you should make sure that no other programs such as Apache or Nginx are binding to your local machine's port 80.

- Install or update Homebrew to the latest version using `brew update`.
- Install PHP 7.4 using Homebrew via `brew install php`.
- Install Composer.
- Install Valet with Composer via `composer global require laravel/valet`. Make sure the `~/.composer/vendor/bin` directory is in your system's "PATH".
- Run the `valet install` command. This will configure and install Valet and DnsMasq, and register Valet's daemon to launch when your system starts.

Once Valet is installed, try pinging any `*.test` domain on your terminal using a command such as `ping foobar.test`. If Valet is installed correctly you should see this domain responding on `127.0.0.1`.

Valet will automatically start its daemon each time your machine boots. There is no need to run `valet start` or `valet install` ever again once the initial Valet installation is complete.

Using Another Domain

By default, Valet serves your projects using the `.test` TLD. If you'd like to use another domain, you can do so using the `valet tld tld-name` command.

For example, if you'd like to use `.app` instead of `.test`, run `valet tld app` and Valet will start serving your projects at `*.app` automatically.

Database

If you need a database, try MySQL by running `brew install mysql@5.7` on your command line. Once MySQL has been installed, you may start it using the `brew services start mysql@5.7` command. You can then connect to the database at `127.0.0.1` using the `root` username and an empty string for the password.

PHP Versions

Valet allows you to switch PHP versions using the `valet use php@version` command. Valet will install the specified PHP version via Brew if it is not already installed:

```
valet use php@7.2
```

```
valet use php
```

> ⚠⚠ Valet only serves one PHP version at a time, even if you have multiple PHP versions installed. ⚠⚠

Resetting Your Installation

If you are having trouble getting your Valet installation to run properly, executing the `composer global update` command followed by `valet install` will reset your installation and can solve a variety of problems. In rare cases it may be necessary to "hard reset" Valet by executing `valet uninstall --force` followed by `valet install`.

Upgrading

You may update your Valet installation using the `composer global update` command in your terminal. After upgrading, it is good practice to run the `valet install` command so Valet can make additional upgrades to your configuration files if necessary.

Serving Sites

Once Valet is installed, you're ready to start serving sites. Valet provides two commands to help you serve your Laravel sites: `park` and `link`.

The `park` Command

- Create a new directory on your Mac by running something like `mkdir ~/Sites`. Next, `cd ~/Sites` and run `valet park`. This command will register your current working directory as a path that Valet should search for sites.
- Next, create a new Laravel site within this directory: `laravel new blog`.
- Open `http://blog.test` in your browser.

That's all there is to it. Now, any Laravel project you create within your "parked" directory will automatically be served using the `http://folder-name.test` convention.

The `link` Command

The `link` command may also be used to serve your Laravel sites. This command is useful if you want to serve a single site in a directory and not the entire directory.

- To use the command, navigate to one of your projects and run `valet link app-name` in your terminal. Valet will create a symbolic link in `~/.config/valet/Sites` which points to your current working directory.
- After running the `link` command, you can access the site in your browser at `http://app-name.test`.

To see a listing of all of your linked directories, run the `valet links` command. You may use `valet unlink app-name` to destroy the symbolic link.

> 💡💡 You can use `valet link` to serve the same project from multiple (sub)domains. To add a subdomain or another domain to your project run `valet link subdomain.app-name` from the project folder. 💡💡

Securing Sites With TLS

By default, Valet serves sites over plain HTTP. However, if you would like to serve a site over encrypted TLS using HTTP/2, use the `secure` command. For example, if your site is being served by Valet on the `laravel.test` domain, you should run the following command to secure it:

```
valet secure laravel
```

To "unsecure" a site and revert back to serving its traffic over plain HTTP, use the `unsecure` command. Like the `secure` command, this command accepts the host name that you wish to unsecure:

```
valet unsecure laravel
```

Sharing Sites

Valet even includes a command to share your local sites with the world, providing an easy way to test your site on mobile devices or share it with team members and clients. No additional software installation is required once Valet is installed.

Sharing Sites Via Ngrok

To share a site, navigate to the site's directory in your terminal and run the `valet share` command. A publicly accessible URL will be inserted into your clipboard and is ready to paste directly into your browser or share with your team.

To stop sharing your site, hit `Control + C` to cancel the process.

> 💡💡 You may pass additional parameters to the share command, such as `valet share --region=eu`. For more information, consult the ngrok documentation. 💡💡

Sharing Sites Via Expose

If you have Expose installed, you can share your site by navigating to the site's directory in your terminal and running the `expose` command. Consult the expose documentation for additional command-line parameters it supports. After sharing the site, Expose will display the sharable URL that you may use on your other devices or amongst team members.

To stop sharing your site, hit `Control + C` to cancel the process.

Sharing Sites On Your Local Network

Valet restricts incoming traffic to the internal `127.0.0.1` interface by default. This way your development machine isn't exposed to security risks from the Internet.

If you wish to allow other devices on your local network to access the Valet sites on your machine via your machine's IP address (eg: `192.168.1.10/app-name.test`), you will need to manually edit the appropriate Nginx configuration file for that site to remove the restriction on the `listen` directive by removing the the `127.0.0.1:` prefix on the directive for ports 80 and 443.

If you have not run `valet secure` on the project, you can open up network access for all non-HTTPS sites by editing the `/usr/local/etc/nginx/valet/valet.conf` file. However, if you're serving the project site over HTTPS (you have run `valet secure` for the site) then you should edit the `~/.config/valet/Nginx/app-name.test` file.

Once you have updated your Nginx configuration, run the `valet restart` command to apply the configuration changes.

Site Specific Environment Variables

Some applications using other frameworks may depend on server environment variables but do not provide a way for those variables to be configured within your project. Valet allows you to configure site specific environment variables by adding a `.valet-env.php` file within the root of your project. These variables will be added to the `$_SERVER` global array:

```
<?php

// Set $_SERVER['key'] to "value" for the foo.test site...
return [
    'foo' => [
        'key' => 'value',
    ],
];

// Set $_SERVER['key'] to "value" for all sites...
return [
    '*' => [
        'key' => 'value',
    ],
];
```

Proxying Services

Sometimes you may wish to proxy a Valet domain to another service on your local machine. For example, you may occasionally need to run Valet while also running a separate site in Docker; however, Valet and Docker can't both bind to port 80 at the same time.

To solve this, you may use the `proxy` command to generate a proxy. For example, you may proxy all traffic from `http://elasticsearch.test` to `http://127.0.0.1:9200`:

```
valet proxy elasticsearch http://127.0.0.1:9200
```

You may remove a proxy using the `unproxy` command:

```
valet unproxy elasticsearch
```

You may use the `proxies` command to list all site configuration that are proxied:

```
valet proxies
```

Custom Valet Drivers

You can write your own Valet "driver" to serve PHP applications running on another framework or CMS that is not natively supported by Valet. When you install Valet, a `~/.config/valet/Drivers` directory is created which contains a `SampleValetDriver.php` file. This file contains a sample driver implementation to demonstrate how to write a custom driver. Writing a driver only requires you to implement three methods: `serves`, `isStaticFile`, and `frontControllerPath`.

All three methods receive the `$sitePath`, `$siteName`, and `$uri` values as their arguments. The `$sitePath` is the fully qualified path to the site being served on your machine, such as `/Users/Lisa/Sites/my-project`. The `$siteName` is the "host" / "site name" portion of the domain (`my-project`). The `$uri` is the incoming request URI (`/foo/bar`).

Once you have completed your custom Valet driver, place it in the `~/.config/valet/Drivers` directory using the `FrameworkValetDriver.php` naming convention. For example, if you are writing a custom valet driver for WordPress, your file name should be `WordPressValetDriver.php`.

Let's take a look at a sample implementation of each method your custom Valet driver should implement.

The `serves` Method

The `serves` method should return `true` if your driver should handle the incoming request. Otherwise, the method should return `false`. So, within this method you should attempt to determine if the given `$sitePath` contains a project of the type you are trying to serve.

For example, let's pretend we are writing a `WordPressValetDriver`. Our `serves` method might look something like this:

```
/**
 * Determine if the driver serves the request.
 *
 * @param  string  $sitePath
 * @param  string  $siteName
 * @param  string  $uri
 * @return bool
 */
public function serves($sitePath, $siteName, $uri)
{
    return is_dir($sitePath.'/wp-admin');
}
```

The `isStaticFile` Method

The `isStaticFile` should determine if the incoming request is for a file that is "static", such as an image or a stylesheet. If the file is static, the method should return the fully qualified path to the static file on disk. If the incoming request is not for a static file, the method should return `false`:

```
/**
 * Determine if the incoming request is for a static file.
 *
 * @param  string  $sitePath
 * @param  string  $siteName
 * @param  string  $uri
 * @return string|false
 */
public function isStaticFile($sitePath, $siteName, $uri)
{
    if (file_exists($staticFilePath = $sitePath.'/public/'.$uri)) {
        return $staticFilePath;
```

```
    }

    return false;
}
```

> ⚠⚠ The `isStaticFile` method will only be called if the `serves` method returns `true` for the incoming request and the request URI is not `/`. ⚠⚠

The `frontControllerPath` Method

The `frontControllerPath` method should return the fully qualified path to your application's "front controller", which is typically your "index.php" file or equivalent:

```
/**
 * Get the fully resolved path to the application's front controller.
 *
 * @param  string  $sitePath
 * @param  string  $siteName
 * @param  string  $uri
 * @return string
 */
public function frontControllerPath($sitePath, $siteName, $uri)
{
    return $sitePath.'/public/index.php';
}
```

Local Drivers

If you would like to define a custom Valet driver for a single application, create a `LocalValetDriver.php` in the application's root directory. Your custom driver may extend the base `ValetDriver` class or extend an existing application specific driver such as the `LaravelValetDriver`:

```
class LocalValetDriver extends LaravelValetDriver
{
    /**
     * Determine if the driver serves the request.
     *
     * @param  string  $sitePath
     * @param  string  $siteName
     * @param  string  $uri
     * @return bool
     */
    public function serves($sitePath, $siteName, $uri)
    {
        return true;
    }

    /**
     * Get the fully resolved path to the application's front controller.
     *
     * @param  string  $sitePath
     * @param  string  $siteName
     * @param  string  $uri
     * @return string
     */
    public function frontControllerPath($sitePath, $siteName, $uri)
```

```
    {
        return $sitePath.'/public_html/index.php';
    }
}
```

Other Valet Commands

Command	Description
`valet forget`	Run this command from a "parked" directory to remove it from the parked directory list.
`valet log`	View a list of logs which are written by Valet's services.
`valet paths`	View all of your "parked" paths.
`valet restart`	Restart the Valet daemon.
`valet start`	Start the Valet daemon.
`valet stop`	Stop the Valet daemon.
`valet trust`	Add sudoers files for Brew and Valet to allow Valet commands to be run without prompting for passwords.
`valet uninstall`	Uninstall Valet: Shows instructions for manual uninstall; or pass the `--force` parameter to aggressively delete all of Valet.

Valet Directories & Files

You may find the following directory and file information helpful while troubleshooting issues with your Valet environment:

File / Path	Description
`~/.config/valet/`	Contains all of Valet's configuration. You may wish to maintain a backup of this folder.
`~/.config/valet/dnsmasq.d/`	Contains DNSMasq's configuration.
`~/.config/valet/Drivers/`	Contains custom Valet drivers.
`~/.config/valet/Extensions/`	Contains custom Valet extensions / commands.
`~/.config/valet/Nginx/`	Contains all Valet generated Nginx site configurations. These files are rebuilt when running the `install`, `secure`, and `tld` commands.
`~/.config/valet/Sites/`	Contains all symbolic links for linked projects.
`~/.config/valet/config.json`	Valet's master configuration file

File / Path	Description
`~/.config/valet/valet.sock`	The PHP-FPM socket used by Valet's Nginx configuration. This will only exist if PHP is running properly.
`~/.config/valet/Log/fpm-php.www.log`	User log for PHP errors.
`~/.config/valet/Log/nginx-error.log`	User log for Nginx errors.
`/usr/local/var/log/php-fpm.log`	System log for PHP-FPM errors.
`/usr/local/var/log/nginx`	Contains Nginx access and error logs.
`/usr/local/etc/php/X.X/conf.d`	Contains `*.ini` files for various PHP configuration settings.
`/usr/local/etc/php/X.X/php-fpm.d/valet-fpm.conf`	PHP-FPM pool configuration file.
`~/.composer/vendor/laravel/valet/cli/stubs/secure.valet.conf`	The default Nginx configuration used for building site certificates.

Deployment

Introduction

When you're ready to deploy your Laravel application to production, there are some important things you can do to make sure your application is running as efficiently as possible. In this document, we'll cover some great starting points for making sure your Laravel application is deployed properly.

Server Configuration

Nginx

If you are deploying your application to a server that is running Nginx, you may use the following configuration file as a starting point for configuring your web server. Most likely, this file will need to be customized depending on your server's configuration. If you would like assistance in managing your server, consider using a service such as Laravel Forge:

```
server {
    listen 80;
    server_name example.com;
    root /srv/example.com/public;

    add_header X-Frame-Options "SAMEORIGIN";
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Content-Type-Options "nosniff";

    index index.php;

    charset utf-8;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location = /favicon.ico { access_log off; log_not_found off; }
    location = /robots.txt  { access_log off; log_not_found off; }

    error_page 404 /index.php;

    location ~ \.php$ {
        fastcgi_pass unix:/var/run/php/php7.4-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
```

```
    }

    location ~ /\.(?!well-known).* {
        deny all;
    }
}
```

Optimization

Autoloader Optimization

When deploying to production, make sure that you are optimizing Composer's class autoloader map so Composer can quickly find the proper file to load for a given class:

```
composer install --optimize-autoloader --no-dev
```

> 💡💡 In addition to optimizing the autoloader, you should always be sure to include a `composer.lock` file in your project's source control repository. Your project's dependencies can be installed much faster when a `composer.lock` file is present. 💡💡

Optimizing Configuration Loading

When deploying your application to production, you should make sure that you run the `config:cache` Artisan command during your deployment process:

```
php artisan config:cache
```

This command will combine all of Laravel's configuration files into a single, cached file, which greatly reduces the number of trips the framework must make to the filesystem when loading your configuration values.

> ⚠⚠ If you execute the `config:cache` command during your deployment process, you should be sure that you are only calling the `env` function from within your configuration files. Once the configuration has been cached, the `.env` file will not be loaded and all calls to the `env` function will return `null`. ⚠⚠

Optimizing Route Loading

If you are building a large application with many routes, you should make sure that you are running the `route:cache` Artisan command during your deployment process:

```
php artisan route:cache
```

This command reduces all of your route registrations into a single method call within a cached file, improving the performance of route registration when registering hundreds of routes.

Optimizing View Loading

When deploying your application to production, you should make sure that you run the `view:cache` Artisan command during your deployment process:

```
php artisan view:cache
```

This command precompiles all your Blade views so they are not compiled on demand, improving the performance of each request that returns a view.

Deploying With Forge / Vapor

If you aren't quite ready to manage your own server configuration or aren't comfortable configuring all of the various services needed to run a robust Laravel application, Laravel Forge is a wonderful alternative.

Laravel Forge can create servers on various infrastructure providers such as DigitalOcean, Linode, AWS, and more. In addition, Forge installs and manages all of the tools needed to build robust Laravel applications, such as Nginx, MySQL, Redis, Memcached, Beanstalk, and more.

Laravel Vapor

If you would like a totally serverless, auto-scaling deployment platform tuned for Laravel, check out Laravel Vapor. Laravel Vapor is a serverless deployment platform for Laravel, powered by AWS. Launch your Laravel infrastructure on Vapor and fall in love with the scalable simplicity of serverless. Laravel Vapor is fine-tuned by Laravel's creators to work seamlessly with the framework so you can keep writing your Laravel applications exactly like you're used to.