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 Requirements

The Laravel framework has a few system requirements. You should ensure that your web server has the following minimum PHP version and extensions:

- PHP >= 8.2
- Ctype PHP Extension
- cURL PHP Extension
- DOM PHP Extension
- Fileinfo PHP Extension
- Filter PHP Extension
- Hash PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PCRE PHP Extension
- PDO PHP Extension
- Session PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

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 fully-managed Laravel platform like Laravel Cloud.**

Please ensure, like the configuration below, your web server directs all requests to your application's `public/index.php` file. You should never attempt to move the `index.php` file to your project's root, as serving the application from the project root will expose many sensitive configuration files to the public Internet:

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

    add_header X-Frame-Options "SAMEORIGIN";
    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 ~ ^/index\.php(/|$) {
        fastcgi_pass unix:/var/run/php/php8.2-fpm.sock;
        fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
        include fastcgi_params;
        fastcgi_hide_header X-Powered-By;
    }

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

FrankenPHP

FrankenPHP may also be used to serve your Laravel applications. FrankenPHP is a modern PHP application server written in Go. To serve a Laravel PHP application using FrankenPHP, you may simply invoke its `php-server` command:

```
frankenphp php-server -r public/
```

To take advantage of more powerful features supported by FrankenPHP, such as its Laravel Octane integration, HTTP/3, modern compression, or the ability to package Laravel applications as standalone binaries, please consult FrankenPHP's Laravel documentation.

Directory Permissions

Laravel will need to write to the `bootstrap/cache` and `storage` directories, so you should ensure the web server process owner has permission to write to these directories.

Optimization

When deploying your application to production, there are a variety of files that should be cached, including your configuration, events, routes, and views. Laravel provides a single, convenient `optimize` Artisan command that will cache all of these files. This command should typically be invoked as part of your application's deployment process:

```
php artisan optimize
```

The `optimize:clear` method may be used to remove all of the cache files generated by the `optimize` command as well as all keys in the default cache driver:

```
php artisan optimize:clear
```

In the following documentation, we will discuss each of the granular optimization commands that are executed by the `optimize` command.

Caching Configuration

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.

> [!WARNING] 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 for `.env` variables will return `null`.

Caching Events

You should cache your application's auto-discovered event to listener mappings during your deployment process. This can be accomplished by invoking the `event:cache` Artisan command during deployment:

```
php artisan event:cache
```

Caching Routes

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.

Caching Views

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.

Debug Mode

The debug option in your `config/app.php` configuration file determines how much information about an error is actually displayed to the user. By default, this option is set to respect the value of the `APP_DEBUG` environment variable, which is stored in your application's `.env` file.

> [!WARNING] **In your production environment, this value should always be `false`. If the `APP_DEBUG` variable is set to `true` in production, you risk exposing sensitive configuration values to your application's end users.**

The Health Route

Laravel includes a built-in health check route that can be used to monitor the status of your application. In production, this route may be used to report the status of your application to an uptime monitor, load balancer, or orchestration system such as Kubernetes.

By default, the health check route is served at `/up` and will return a 200 HTTP response if the application has booted without exceptions. Otherwise, a 500 HTTP response will be returned. You may configure the URI for this route in your application's `bootstrap/app` file:

```
->withRouting(
    web: __DIR__.'/../routes/web.php',
    commands: __DIR__.'/../routes/console.php',
    health: '/up', // [tl! remove]
    health: '/status', // [tl! add]
)
```

When HTTP requests are made to this route, Laravel will also dispatch a `Illuminate\Foundation\Events\DiagnosingHealth` event, allowing you to perform additional health checks relevant to your application. Within a listener for this event, you may check your application's database or cache status. If you detect a problem with your application, you may simply throw an exception from the listener.

Deploying With Laravel Cloud or Forge

Laravel Cloud

If you would like a fully-managed, auto-scaling deployment platform tuned for Laravel, check out Laravel Cloud. Laravel Cloud is a robust deployment platform for Laravel, offering managed compute, databases, caches, and object storage.

Launch your Laravel application on Cloud and fall in love with the scalable simplicity. Laravel Cloud 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.

Laravel Forge

If you prefer to manage your own servers but aren't comfortable configuring all of the various services needed to run a robust Laravel application, Laravel Forge is a VPS server management platform for Laravel applications.

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.