Laravel Envoy

Introduction

Laravel Envoy provides a clean, minimal syntax for defining common tasks you run on your remote servers. Using Blade style syntax, you can easily setup tasks for deployment, Artisan commands, and more. Currently, Envoy only supports the Mac and Linux operating systems.

Installation

First, install Envoy using the Composer `global require` command:

```
composer global require laravel/envoy
```

Since global Composer libraries can sometimes cause package version conflicts, you may wish to consider using `cgr`, which is a drop-in replacement for the `composer global require` command. The `cgr` library's installation instructions can be found on GitHub.

> Make sure to place the `~/.composer/vendor/bin` directory in your PATH so the `envoy` executable is found when running the `envoy` command in your terminal.

Updating Envoy

You may also use Composer to keep your Envoy installation up to date. Issuing the `composer global update` command will update all of your globally installed Composer packages:

```
composer global update
```

Writing Tasks

All of your Envoy tasks should be defined in an `Envoy.blade.php` file in the root of your project. Here's an example to get you started:

```
@servers(['web' => ['user@192.168.1.1']])

@task('foo', ['on' => 'web'])
    ls -la
@endtask
```

As you can see, an array of `@servers` is defined at the top of the file, allowing you to reference these servers in the `on` option of your task declarations. Within your `@task` declarations, you should place the Bash code that should run on your server when the task is executed.

You can force a script to run locally by specifying the server's IP address as `127.0.0.1`:

```
@servers(['localhost' => '127.0.0.1'])
```

Setup

Sometimes, you may need to execute some PHP code before executing your Envoy

tasks. You may use the `@setup` directive to declare variables and do other general PHP work before any of your other tasks are executed:

```
@setup
    $now = new DateTime();

    $environment = isset($env) ? $env : "testing";
@endsetup
```

If you need to require other PHP files before your task is executed, you may use the `@include` directive at the top of your `Envoy.blade.php` file:

```
@include('vendor/autoload.php')

@task('foo')
    # ...
@endtask
```

You may also import other Envoy files so their stories and tasks are added to yours. After they have been imported, you may execute the tasks in those files as if they were defined in your own. You should use the `@import` directive at the top of your `Envoy.blade.php` file:

```
@import('package/Envoy.blade.php')
```

Variables

If needed, you may pass option values into Envoy tasks using the command line:

```
envoy run deploy --branch=master
```

You may access the options in your tasks via Blade's "echo" syntax. You may also use `if` statements and loops within your tasks. For example, let's verify the presence of the `$branch` variable before executing the `git pull` command:

```
@servers(['web' => '192.168.1.1'])

@task('deploy', ['on' => 'web'])
    cd site

    @if ($branch)
        git pull origin {{ $branch }}
    @endif

    php artisan migrate
@endtask
```

Stories

Stories group a set of tasks under a single, convenient name, allowing you to group small, focused tasks into large tasks. For instance, a `deploy` story may run the `git` and `composer` tasks by listing the task names within its definition:

```
@servers(['web' => '192.168.1.1'])

@story('deploy')
    git
    composer
@endstory

@task('git')
    git pull origin master
@endtask

@task('composer')
    composer install
@endtask
```

Once the story has been written, you may run it just like a typical task:

```
envoy run deploy
```

Multiple Servers

Envoy allows you to easily run a task across multiple servers. First, add additional servers to your `@servers` declaration. Each server should be assigned a unique name. Once you have defined your additional servers, list each of the servers in the task's `on`

array:

```
@servers(['web-1' => '192.168.1.1', 'web-2' => '192.168.1.2'])

@task('deploy', ['on' => ['web-1', 'web-2']])
    cd site
    git pull origin {{ $branch }}
    php artisan migrate
@endtask
```

Parallel Execution

By default, tasks will be executed on each server serially. In other words, a task will finish running on the first server before proceeding to execute on the second server. If you would like to run a task across multiple servers in parallel, add the `parallel` option to your task declaration:

```
@servers(['web-1' => '192.168.1.1', 'web-2' => '192.168.1.2'])

@task('deploy', ['on' => ['web-1', 'web-2'], 'parallel' => true])
    cd site
    git pull origin {{ $branch }}
    php artisan migrate
@endtask
```

Running Tasks

To run a task or story that is defined in your `Envoy.blade.php` file, execute Envoy's `run` command, passing the name of the task or story you would like to execute. Envoy will run the task and display the output from the servers as the task is running:

```
envoy run deploy
```

Confirming Task Execution

If you would like to be prompted for confirmation before running a given task on your servers, you should add the `confirm` directive to your task declaration. This option is particularly useful for destructive operations:

```
@task('deploy', ['on' => 'web', 'confirm' => true])
    cd site
    git pull origin {{ $branch }}
    php artisan migrate
@endtask
```

Notifications

Slack

Envoy also supports sending notifications to Slack after each task is executed. The `@slack` directive accepts a Slack hook URL and a channel name. You may retrieve your webhook URL by creating an "Incoming WebHooks" integration in your Slack control panel. You should pass the entire webhook URL into the `@slack` directive:

```
@finished
    @slack('webhook-url', '#bots')
@endfinished
```

You may provide one of the following as the channel argument:

- To send the notification to a channel: `#channel`
- To send the notification to a user: `@user`

Discord

Envoy also supports sending notifications to Discord after each task is executed. The `@discord` directive accepts a Discord hook URL and a message. You may retrieve your webhook URL by creating a "Webhook" in your Server Settings and choosing which channel the webhook should post to. You should pass the entire Webhook URL into the `@discord` directive:

```
@finished
    @discord('discord-webhook-url')
@endfinished
```

Become a Laravel Partner

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

Our Partners

Laravel

Highlights

- Release Notes
- Getting Started
- Routing
- Blade Templates
- Authentication
- Authorization

Resources

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

Partners

- Vehikl
- Tighten Co.
- Kirschbaum
- Byte 5
- 64Robots
- Cubet

Ecosystem

- Vapor
- Forge
- Envoyer
- Horizon
- Lumen
- Nova

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