tuts+

Advertisement

CODE > LARAVEL 5

# Task Scheduling in Laravel

by Sajal Soni 4 Dec 2017

Difficulty: Beginner Length: Medium Languages: English

Laravel 5 PHP Web Development

In this article, we'll go through one of the exciting features of the Laravel web framework—task scheduling. Throughout the course of this article, we'll look at how Laravel allows you to manage scheduled tasks in your application. Moreover, we'll also end up creating our own custom scheduled tasks for demonstration purposes.

The Laravel framework allows you to set up scheduled tasks so that you don't have to worry about setting them up at the system level. You can get rid of that complex cron syntax while setting up scheduled tasks since Laravel allows you to define them in a user-friendly way.

We'll start the article with how you are used to setting up traditional cron jobs, and following that we'll explore the Laravel way of achieving it. In the latter half of the article, we'll give it a try by creating couple of custom scheduled tasks that should provide hands-on insight into the subject.

## Traditional Scheduled Task Setup

In your day-to-day application development, you often face a situation that requires you to execute certain scripts or commands periodically. If you're working with the *nix system, you are probably aware that cron jobs handle these commands. On the other hand, they're known as scheduled tasks on Windows-based systems.

Let's have a quick look at a simple example of the *nix based cron job.

```
*/5 * * * * /web/statistics.sh
```

Pretty simple—it runs the `statistics.sh` file every five minutes!

Although that was a pretty simple use case, you often find yourself in a situation that requires you to implement more complex use cases. On the other hand, a complex system requires you to define multiple cron jobs that run at different time intervals.

Let's see some tasks a complex web application has to perform periodically in the back-end.

- Clean up the unnecessary data from the database back-end.
- Update the front-end caching indexes to keep it up-to-date.
- Calculate the site statistics.
- Send emails.
- Back up different site elements.
- Generate reports.
- And more.

So, as you can see, there's plenty of stuff out there waiting to be run periodically and also at different time intervals. If you're a seasoned system admin, it's a cake walk for you to define the cron jobs for all these tasks, but sometimes we as developers wish that there was an easier way around.

Luckily, Laravel comes with a built-in *Task Scheduling* API that allows you to define scheduled tasks like never before. And yes, the next section is all about that—the basics of Laravel task scheduling.

## The Laravel Way

In the earlier section, we went through the traditional way of setting up cron jobs. In this section, we'll go through the specifics of Laravel in the context of the Task Scheduling API.

Before we go ahead, the important thing to understand is that the scheduling feature provided by Laravel is just like any other feature and won't be invoked automatically. So if you're thinking that you don't need to do anything at the system level then you're out of luck, I'd say.

In fact, the first thing you should do should you wish to use the Laravel scheduling system is to set up the cron job that runs every minute and calls the artisan command shown in the following snippet.

```
* * * * * php /path-to-your-project/artisan schedule:run >> /dev/null 2>&1
```

The above artisan command calls the Laravel scheduler, and that in turn executes all the pending cron jobs defined in your application.

Of course, we are yet to see how to define the scheduled tasks in your Laravel application, and that's the very next thing we'll dive into.

It's the `schedule` method of the `App\Console\Kernel` class that you need to use should you wish to define application-specific scheduled tasks.

Go ahead and grab the contents of the `app/Console/Kernel.php` file.

```
<?php namespace App\Console;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel {
  /**
```

```
   * The Artisan commands provided by your application.
   *
   * @var array
   */
  protected $commands = [
    'App\Console\Commands\Inspire',
  ];

  /**
   * Define the application's command schedule.
   *
   * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
   * @return void
   */
  protected function schedule(Schedule $schedule)
  {
    $schedule->command('inspire')->hourly();
  }
}
```

As you can see, the core code itself provides a useful example. In the above example, Laravel runs the `inspire` artisan command hourly. Don't you think that the syntax is so intuitive in the first place?

In fact, there are a couple of different ways in which Laravel allows you to define schedule tasks:

- Use the closure/callable.
- Call the artisan command.
- Execute the shell command.

Moreover, there are plenty of built-in scheduling frequencies you could choose from:

- every minute/every five minutes
- hourly/daily/weekly/quarterly/yearly
- at a specific time of the day
- and many more

In fact, I would say that it provides a complete set of routines so that you don't ever need to touch the shell to create your custom cron jobs!

Yes I can tell that you're eager to know how to implement your custom scheduled tasks, and that is what I also promised at the beginning of the article.

## Create Your First Scheduled Task in Laravel

As we discussed, there are different ways in which Laravel allows you to define scheduled tasks. Let's go through each to understand how it works.

### The Closure/Callable Method

The scheduling API provides the `call` method that allows you to execute a callable or a closure function. Let's revise the `app/Console/Kernel.php` file with the following code.

```
<?php
```

```
<?php
namespace App\Console;

use Illuminate\Support\Facades\DB;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
  /**
   * Define the application's command schedule.
   *
   * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
   * @return void
   */
  protected function schedule(Schedule $schedule)
  {
    // the call method
    $schedule->call(function () {
      $posts = DB::table('posts')
        ->select('user_id', DB::raw('count(*) as total_posts'))
        ->groupBy('user_id')
        ->get();

      foreach($posts as $post)
      {
        DB::table('users_statistics')
          ->where('user_id', $post->user_id)
          ->update(['total_posts' => $post->total_posts]);
      }
    })->everyThirtyMinutes();
  }
}
```

As you can see, we've passed the closure function as the first argument of the `call` method. Also, we've set the frequency to every 30 minutes, so it'll execute the closure function every 30 minutes!

In our example, we count the total posts per user and update the statistics table accordingly.

## The Artisan Command

Apart from the closures or callables, you could also schedule an artisan command that will be executed at certain intervals. In fact, that should be the preferred approach over closures as it provides better code organization and reusability at the same time.

Go ahead and revise the contents of the `app/Console/Kernel.php` file with the following.

```
<?php
namespace App\Console;

use Illuminate\Support\Facades\Config;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
  /**
   * The Artisan commands provided by your application.
   *
   * @var array
   */
  protected $commands = [
    'App\Console\Commands\UserStatistics'
  ];

  /**
   * Define the application's command schedule.
```

```
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
      // artisan command method
      $schedule->command('statistics:user')->everyThirtyMinutes();
    }

    /**
     * Register the Closure based commands for the application.
     *
     * @return void
     */
    protected function commands()
    {
      require base_path('routes/console.php');
    }
  }
```

It's the `command` method that you would like to use should you wish to schedule an artisan command as shown in the above code snippet. You need to pass the artisan command signature as the first argument of the `command` method.

Of course, you need to define the corresponding artisan command as well at `app/Console/Commands/UserStatistics.php`.

```
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use Illuminate\Support\Facades\DB;

class UserStatistics extends Command
{
  /**
   * The name and signature of the console command.
   *
   * @var string
   */
  protected $signature = 'statistics:user';

  /**
   * The console command description.
   *
   * @var string
   */
  protected $description = 'Update user statistics';

  /**
   * Create a new command instance.
   *
   * @return void
   */
  public function __construct()
  {
    parent::__construct();
  }

  /**
   * Execute the console command.
   *
   * @return mixed
   */
  public function handle()
  {
    // calculate new statistics
    $posts = DB::table('posts')
      ->select('user_id', DB::raw('count(*) as total_posts'))
      ->groupBy('user_id')
      ->get();

    // update statistics table
    foreach($posts as $post)
    {
      DB::table('users_statistics')
      ->where('user_id', $post->user_id)
      ->update(['total_posts' => $post->total_posts]);
    }
  }
}
```

### The Exec Command

We could say that the methods we've discussed so far were specific to the Laravel application itself. Moreover, Laravel also allows you to schedule the shell commands so that you could run external applications as well.

Let's go through a quick example that demonstrates how to take a backup of your database every day.

```
<?php
namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
  /**
   * Define the application's command schedule.
   *
   * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
   * @return void
   */
  protected function schedule(Schedule $schedule)
  {
    // exec method
    $host = config('database.connections.mysql.host');
    $username = config('database.connections.mysql.username');
    $password = config('database.connections.mysql.password');
    $database = config('database.connections.mysql.database');

    $schedule->exec("mysqldump -h {$host} -u {$username} -p{$password} {$database}")
      ->daily()
      ->sendOutputTo('/backups/daily_backup.sql');
  }
}
```

It's apparent from the code that you need to use the `exec` method of the scheduler, and you need to pass the command that you would like to run as its first argument.

Apart from that, we've also used the `sendOutputTo` method that allows you to collect the output of the command. On the other hand, there's a method, `emailOutputTo`, that allows you to email the output contents!

And that brings us to the end of the article. In fact, we've just scratched the surface of the Laravel Scheduling API, and it has a lot to offer in its kitty.

## Conclusion

Today, we went through the task scheduling API in the Laravel web framework. It was fascinating to see how easily it allows you to manage tasks that need to be run periodically.

At the beginning of the article, we discussed the traditional way of setting up scheduled tasks, and following that we introduced the Laravel way of doing it. In the latter half of the article, we went through a couple of practical examples to demonstrate task scheduling concepts.

I hope that you've enjoyed the article, and you should feel more confident about setting up scheduled tasks in Laravel. For those of you who are either just getting started with Laravel or looking to expand your knowledge, site, or application with extensions, we have a variety of things you can study in Envato Market.

Should anything pop up in your mind, let's start a conversation using the feed below!

Sajal Soni
Software Engineer, INDIA

Sajal belongs to India and he loves to spend time creating websites based on open source frameworks. Apart from this, it's traveling and listening music which takes the rest of his time!

sajalsoni

Email Address

Update me weekly

**Translations**

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by native

**2 Comments** **Tuts+ Hub** 1 **Login**

Recommend 3 Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

DS • 4 months ago

Very useful tutorial. Thank you!

1 • Reply • Share ›

Patrick Kalamula • 4 months ago

Very useful. I love it. Thanks.

• Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy

Advertisement

QUICK LINKS - Explore popular categories

ENVATO TUTS+ +

JOIN OUR COMMUNITY +

HELP +

tuts+

26,486 Tutorials 1,167 Courses 28,658 Translations

Envato.com Our products Careers Sitemap

Follow Envato Tuts+