How to Send Emails in Laravel with SMTP, Mailgun, and Postmark

How to Send Emails in Laravel with SMTP, Mailgun, and Postmark

Sending emails is a core feature for many applications—whether it's delivering account confirmations, password reset links, or notifications. Laravel 12 makes email delivery simple with its built-in `Mail` system and support for multiple drivers such as **SMTP**, **Mailgun**, and **Postmark**. In this guide, you'll learn how to configure each driver, send your first email, use a mailable inside a controller, trigger emails from events, and test email delivery in your project.

Configuring SMTP in Laravel

SMTP (Simple Mail Transfer Protocol) is one of the most common ways to send emails. Laravel provides an SMTP driver out of the box. Open your `.env` file and add the following configuration:

```bash
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=587
MAIL_USERNAME=your_smtp_username
MAIL_PASSWORD=your_smtp_password
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=no-reply@example.com
MAIL_FROM_NAME="My App"
```
Code language: Bash (bash)

This sample uses Mailtrap for safe testing. In production, replace with your provider (e.g., Gmail SMTP, SendGrid). `MAIL_FROM_*` values define the default sender for outgoing mail.

Sending Emails with Mailables

Laravel uses **Mailables** to structure, render, and send emails. Generate one with Markdown support:

```bash
php artisan make:mail WelcomeMail --markdown=emails.welcome
```
Code language: Bash (bash)

This command creates a mailable and a Markdown Blade view so you can style emails quickly and consistently.

```php
namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class WelcomeMail extends Mailable
{
    use Queueable, SerializesModels;

    public $user;

    public function __construct($user)
    {
        $this->user = $user;
    }

    public function build()
    {
        return $this->subject('Welcome to My App')
                    ->markdown('emails.welcome')
                    ->with([
                        'name' => $this->user->name,
                    ]);
    }
}
```
Code language: PHP (php)

`build()` sets the subject, selects the Markdown template, and passes data (`name`) to the view for personalization.

Using a Mailable Inside a Controller

Send the `WelcomeMail` right after user registration from your controller:

```
namespace App\Http\Controllers;

use App\Mail\WelcomeMail;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Mail;

class RegisterController extends Controller
{
    public function store(Request $request)
    {
        $user = User::create($request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|email|unique:users',
            'password' => 'required|min:8'
        ]));

        Mail::to($user->email)->send(new WelcomeMail($user));

        return redirect('/home')->with('success', 'Account created and
welcome email sent!');
```

```
    }
}
```
Code language: PHP (php)

This keeps the flow simple: create the user, then send the email. For larger apps, consider events to decouple email logic from controllers.

Using a Mailable with Events

Events and listeners separate concerns and make registration extensible (send email, log analytics, etc.). Generate an event and a listener:

```
php artisan make:event UserRegistered
php artisan make:listener SendWelcomeEmail --event=UserRegistered
```
Code language: Bash (bash)

These commands scaffold a `UserRegistered` event and a `SendWelcomeEmail` listener that will react to the event.

```
// app/Events/UserRegistered.php
namespace App\Events;

use App\Models\User;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

class UserRegistered
{
    use Dispatchable, SerializesModels;

    public $user;

    public function __construct(User $user)
    {
        $this->user = $user;
    }
}Code language: PHP (php)
```

The event carries the `User` instance so listeners have everything needed to perform follow-up actions.

```
// app/Listeners/SendWelcomeEmail.php
namespace App\Listeners;

use App\Events\UserRegistered;
use App\Mail\WelcomeMail;
use Illuminate\Support\Facades\Mail;

class SendWelcomeEmail
{
    public function handle(UserRegistered $event): void
    {
        Mail::to($event->user->email)
            ->send(new WelcomeMail($event->user));
    }
}Code language: PHP (php)
```

The listener's `handle()` method receives the event, then dispatches the `WelcomeMail` to the new user's email address.

```
// app/Providers/EventServiceProvider.php
namespace App\Providers;

use Illuminate\Foundation\Support\Providers\EventServiceProvider as
ServiceProvider;
use App\Events\UserRegistered;
use App\Listeners\SendWelcomeEmail;

class EventServiceProvider extends ServiceProvider
{
    protected $listen = [
        UserRegistered::class => [
            SendWelcomeEmail::class,
        ],
    ];
}Code language: PHP (php)
```

Registering the mapping in `$listen` tells Laravel to run `SendWelcomeEmail` whenever `UserRegistered` is dispatched.

```
// app/Http/Controllers/RegisterController.php
use App\Events\UserRegistered;

public function store(Request $request)
{
    $user = User::create($request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|unique:users',
        'password' => 'required|min:8'
    ]));

    UserRegistered::dispatch($user);

    return redirect('/home')->with('success', 'Account created
successfully!');
}Code language: PHP (php)
```

Now the controller only fires an event. The listener handles email sending, improving testability and maintainability.

Mailgun Configuration

Use Mailgun for transactional delivery by setting these variables in `.env`:

```
MAIL_MAILER=mailgun
MAILGUN_DOMAIN=your-domain.com
```

```
MAILGUN_SECRET=your-mailgun-key
MAILGUN_ENDPOINT=api.mailgun.net
```
Code language: Bash (bash)

With `MAIL_MAILER=mailgun`, all `Mail::to(...)->send(...)` calls route through Mailgun's API transport.

Postmark Configuration

Postmark focuses on high deliverability. Configure it like this:

```
MAIL_MAILER=postmark
POSTMARK_TOKEN=your-postmark-server-token
MAIL_FROM_ADDRESS=hello@yourdomain.com
MAIL_FROM_NAME="My App"
```
Code language: Bash (bash)

Once active, Laravel will send via Postmark—helpful for critical emails like password resets and receipts.

Testing Emails with PHPUnit

Use fakes to test behavior without sending real mail. Here we verify that the event triggers the welcome email:

```
namespace Tests\Feature;

use Tests\TestCase;
use App\Events\UserRegistered;
use App\Mail\WelcomeMail;
use App\Models\User;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Facades\Event;
use Illuminate\Foundation\Testing\RefreshDatabase;

class MailTest extends TestCase
{
    use RefreshDatabase;

    public function test_event_triggers_welcome_email()
    {
        Mail::fake();
        Event::fake();

        $user = User::factory()->create();

        UserRegistered::dispatch($user);

        Event::assertDispatched(UserRegistered::class);

        Mail::assertSent(WelcomeMail::class, function ($mail) use
($user) {
            return $mail->hasTo($user->email);
        });
    }
}
```
Code language: PHP (php)

We fake both subsystems, assert the event was dispatched, and confirm the expected mailable was sent to the right recipient.

```
PHPUnit 10.*/Laravel Test Runner

   PASS  Tests\Feature\MailTest
  ✓ test_event_triggers_welcome_email

  Tests:  1 passed
  Assertions: 2
  Time: 0.41s
```
Code language: Bash (bash)

This output indicates a successful run—your event fired and your mailable was sent, all without hitting external mail services.

Wrapping Up

You configured SMTP, Mailgun, and Postmark; created a reusable mailable; sent it from a controller; refactored to events and listeners for clean separation; and tested the flow with fakes. This foundation ensures reliable, maintainable email delivery in production.

What’s Next

Explore more advanced communication and notification patterns:

- Mastering Laravel Notifications: Mail, SMS, and Slack
- Laravel Events and Listeners: A Complete Guide
- How to Queue Emails in Laravel for Faster Delivery