Testing

Test the code that sends and receives mail against a real mail server in Docker, with the base class and the server of Derafu Mail

Anonymous · 05/11/2026

Testing with a real mail server

Code that sends and receives mail is only proven when a mail goes through a server and is read from it. `Derafu\Mail\Testing` has what a project needs for that:

- `GreenMailServer`: a mail server (SMTP and IMAP) in a Docker container. It uses GreenMail, a server for tests: the mails that it receives by SMTP can be read by IMAP, and nothing leaves the container.
- `MailServerTestCase`: the base class of the tests. It starts the server before the first test of the class, deletes the mails before each test and stops the server after the last one.

The project needs Docker, and `phpunit/phpunit`, that Derafu Mail only suggests.

A test

A class of tests extends `MailServerTestCase` and gets the server with `server()`. The server has the users `ana` and `beto`, with the password `secret` and the domain `example.com`, and gives the options of the transport of each one:

```
use Derafu\Mail\Component\Exchange\Worker\Receiver\Handler\ReceiveHandler;
use Derafu\Mail\Component\Exchange\Worker\Receiver\Strategy\ImapStrategy;
use Derafu\Mail\Component\Exchange\Worker\ReceiverWorker;
use Derafu\Mail\Component\Exchange\Worker\Sender\Handler\SendHandler;
use Derafu\Mail\Component\Exchange\Worker\Sender\Strategy\SmtpStrategy;
use Derafu\Mail\Component\Exchange\Worker\SenderWorker;
use Derafu\Mail\Factory\EnvelopeFactory;
use Derafu\Mail\Model\Envelope;
use Derafu\Mail\Model\Message;
use Derafu\Mail\Model\Postman;
use Derafu\Mail\Testing\MailServerTestCase;
use PHPUnit\Framework\Attributes\Group;
use Symfony\Component\Mime\Address;

#[Group('docker')]
final class InvoiceMailTest extends MailServerTestCase
{
```

```
    private SenderWorker $sender;

    private ReceiverWorker $receiver;

    protected function setUp(): void
    {
        parent::setUp();

        $this->sender = new SenderWorker([
            'send' => new SendHandler(['smtp' => new SmtpStrategy()]),
        ]);
        $this->receiver = new ReceiverWorker([
            'receive' => new ReceiveHandler(['imap' => new ImapStrategy(new
EnvelopeFactory())]),
        ]);
    }

    public function testTheInvoiceReachesTheCustomer(): void
    {
        $message = new Message();
        $message->subject('Your invoice');
        $message->text('Attached.');
        $envelope = new Envelope(new Address('ana@example.com'), [new
Address('beto@example.com')]);
        $envelope->addMessage($message);

        // Send as `ana`.
        $postman = new Postman(['transport' => self::server()->smtpTransport('ana')]);
        $postman->addEnvelope($envelope);
        $this->sender->send($postman);

        // Read as `beto`.
        $postman = new Postman(['transport' =>
self::server()->imapTransport('beto')]);
        $received = $this->receiver->receive($postman);

        $this->assertCount(1, $received);
    }
}
```

`smtpTransport()` and `imapTransport()` return the options of `transport` of the SMTP and IMAP strategies, so the code that is tested does not know that the server is not a real one. `imapTransport()` receives as its second argument the options of `search`. `imapDsn()` gives the DSN of the IMAP server for a `Mailbox`.

A class that overrides `setUp()`, `setUpBeforeClass()` or `tearDownAfterClass()` calls the parent.

Without Docker

Without Docker the class fails with the reason, it does not skip its tests: a test that is skipped without anybody noticing is a test that does not exist.

PHPUnit 13 does not inherit the group of a parent class, so each class declares `#[Group('docker')]`. With it, the tests of a project run without the ones that need Docker with:

```
vendor/bin/phpunit --exclude-group docker
```

The server on its own

`GreenMailServer` is used without the base class when the test needs another set of users, or a server for a part of the tests:

```
use Derafu\Mail\Testing\GreenMailServer;

if (!GreenMailServer::isAvailable()) {
    // Docker is not installed or its daemon does not answer.
}

$server = GreenMailServer::start(
    users: ['carla', 'dario'],
    password: 'secret',
    domain: 'example.org',
);

$server->smtpTransport('carla');
$server->imapTransport('dario');
$server->purge(); // Deletes the mails of all the users.
$server->stop();  // Stops the container, in the background.
```

`start()` waits until the server is ready (`readyTimeout`, 60 seconds by default) and fails with a `MailException` if Docker is not available, if the container does not start or if the server is not ready in time. The `image` of GreenMail is an argument too.

A server that is already running, for example one that a service of the CI provides, is used with the constructor, giving its ports. `stop()` does nothing with a server that the object did not start:

```
$server = new GreenMailServer(
    container: '',
    smtpPort: 3025,
    imapPort: 3143,
    apiPort: 8080,
);
```

Last updated on 05/11/2026