Laravel with Docker & Sail: The Right Way

Laravel with Docker & Sail: The Right Way

Laravel Sail is the official Docker development environment for Laravel. It gives you a ready-to-use Docker Compose setup with PHP, MySQL/Postgres, Redis, Mailhog, and more. In this article, we'll walk step-by-step through setting up Sail, customizing services, adding extensions, debugging containers, and preparing for production. This is the "right way" to embrace Docker without losing Laravel's simplicity.

1 — Install Sail

Laravel Sail ships with new projects, but you can add it to any Laravel 12 app. Require it via Composer, then publish the `docker-compose.yml` file.

```bash
# in your Laravel app root
composer require laravel/sail --dev
php artisan sail:install

# start Sail with Docker Compose
./vendor/bin/sail up -d
```
Code language: Bash (bash)

`sail:install` lets you choose services (MySQL, Redis, Meilisearch, Mailhog, Selenium). The generated `docker-compose.yml` defines containers. Running `sail up -d` launches them in the background.

2 — docker-compose.yml Overview

The default file defines services like `laravel.test`, `mysql`, `redis`, `mailhog`. You can customize it (ports, volumes, versions).

```
version: '3'
services:
  laravel.test:
    build:
      context: ./vendor/laravel/sail/runtimes/8.3
    ports:
      - '${APP_PORT:-80}:80'
    volumes:
      - '.:/var/www/html'
    environment:
      WWWGROUP: '${WWWGROUP}'
    depends_on:
      - mysql
      - redis
      - mailhog

  mysql:
    image: 'mysql:8.0'
    environment:
      MYSQL_DATABASE: '${DB_DATABASE}'
      MYSQL_USER: '${DB_USERNAME}'
      MYSQL_PASSWORD: '${DB_PASSWORD}'
      MYSQL_ROOT_PASSWORD: '${DB_PASSWORD}'
    ports:
      - '3306:3306'

  redis:
    image: 'redis:alpine'
    ports:
      - '6379:6379'

  mailhog:
    image: 'mailhog/mailhog:latest'
```

```yaml
    ports:
      - '8025:8025'
```
Code language: YAML (yaml)

This setup runs Laravel in `laravel.test` container, with MySQL, Redis, and Mailhog. Ports are mapped to your host for development: `http://localhost` for app, `8025` for Mailhog UI.

3 — Customizing PHP & Extensions

You can add PHP extensions by editing the Sail runtime Dockerfile. Example: enabling `imagick`.

```dockerfile
# vendor/laravel/sail/runtimes/8.3/Dockerfile
FROM laravelsail/php83-composer

# Install Imagick
RUN apt-get update && apt-get install -y libmagickwand-dev --no-install-recommends \
    && pecl install imagick \
    && docker-php-ext-enable imagick
```
Code language: Dockerfile (dockerfile)

Rebuild Sail after editing:

```bash
./vendor/bin/sail build --no-cache
```
Code language: Bash (bash)

This ensures your PHP container now has Imagick. You can repeat the same process for other system libs or extensions.

4 — Running Artisan, Composer, and NPM

Sail wraps Docker Compose commands. Instead of `php artisan`, prefix with `sail`. Same for Composer, NPM, PHPUnit.

```bash
# Artisan
./vendor/bin/sail artisan migrate

# Composer
./vendor/bin/sail composer require spatie/laravel-permission

# NPM
./vendor/bin/sail npm run dev

# Testing
./vendor/bin/sail testCode language: Bash (bash)
```

This ensures all commands run inside the PHP container, not on your host machine, so versions are consistent across the team.

5 — Debugging Containers

You can "exec" into running containers or check logs directly.

```bash
# enter PHP container shell
./vendor/bin/sail shell

# check logs for app container
./vendor/bin/sail logs -f laravel.testCode language: Bash (bash)
```

`sail shell` drops you into a bash session inside the PHP container, where you can run artisan tinker or inspect files. `sail logs -f` streams container logs (good for debugging queues or errors).

6 — Preparing Sail for Production

Sail is intended for development, but its Docker Compose setup can inspire production images. For production:

- Build a lean PHP-FPM image with `composer install --no-dev` & cached config/routes/views.
- Use Nginx as a separate container, not inside `laravel.test`.
- Use AWS RDS/DO Managed DB instead of container DB.
- Use Redis container or managed ElastiCache/DO Redis for queues (see #42 Queues).
- Run Horizon in its own service (see #45 Horizon).

For scaling beyond a single host, migrate to Kubernetes or ECS Fargate. See #52 AWS Guide for container deployments.

7 — Developer UI: Sail Status Page

For teams new to Docker, a tiny UI can display which services are running inside Sail. This helps onboarding without needing to know Docker commands.

```
// routes/web.php
Route::get('/sail-status', function () {
    $services = [
        'MySQL' => env('DB_HOST').':'.env('DB_PORT'),
        'Redis' => env('REDIS_HOST').':'.env('REDIS_PORT'),
        'Mailhog' => 'http://localhost:8025'
    ];
    return view('sail.status', compact('services'));
});
```
Code language: PHP (php)

This route builds a simple array of services (DB, Redis, Mailhog) from your `.env` and passes it to a Blade view.

```
<!-- resources/views/sail/status.blade.php -->
@extends('layouts.app')
@section('content')
<div class="container">
  <h1 class="mb-4">Sail Services</h1>
  <ul class="list-group">
    @foreach($services as $name => $url)
      <li class="list-group-item">
        <strong>{{ $name }}:</strong> <a href="{{ $url }}"
target="_blank">{{ $url }}</a>
      </li>
    @endforeach
  </ul>
</div>
@endsection
```
Code language: HTML, XML (xml)

The UI helps developers verify connections quickly. For real monitoring in production, use Horizon (#45) and Telescope (#48).

Wrapping Up

Laravel Sail is the fastest way to onboard developers with Docker: one command launches a full stack (PHP, DB, Redis, Mailhog). By customizing Dockerfiles, adding extensions, and using the Sail CLI, your team enjoys consistent environments without local setup headaches. For production, evolve the setup into lean images with proper Nginx, managed DB/Redis, Horizon for queues, and CI/CD pipelines. This hybrid approach keeps development simple but sets you up for scalable deployments.

[Laravel Starter Kits]

What’s Next

- Laravel and Docker: Setting Up a Scalable Dev Environment — dive deeper into Docker for both dev & production (Article #46).
- CI/CD for Laravel Projects with GitHub Actions — automate build & deploy pipelines (Article #54).
- Deploying Laravel on AWS: Complete Guide (2025) — containerize and run your app on ECS or EC2 (Article #52).