Laravel and Docker: Setting Up a Scalable Dev Environment

Laravel and Docker: Setting Up a Scalable Dev Environment

Docker has become the standard for creating consistent, portable development environments. Instead of “it works on my machine,” Docker ensures your Laravel app runs the same way everywhere—on your laptop, staging server, or production cluster. In this guide, we’ll containerize Laravel, configure services like MySQL and Redis, and run the app in a scalable setup.

1 – Install Docker & Docker Compose

First, ensure you have Docker Engine and Docker Compose installed. Docker Compose lets you run multiple containers (Laravel app, MySQL, Redis, Nginx) together.

```
# Check Docker
docker -v
# Check Compose
docker compose versionCode language: Bash (bash)
```

If these commands print version numbers, you’re good to go. Otherwise, download from Docker Desktop.

2 – Create a Dockerfile for Laravel

The Dockerfile defines how the Laravel container is built.

```
# Dockerfile
FROM php:8.3-fpm

# Install system dependencies
RUN apt-get update && apt-get install -y \
    git curl libpng-dev libonig-dev libxml2-dev zip unzip \
    && docker-php-ext-install pdo_mysql mbstring exif pcntl bcmath gd

# Install Composer
COPY --from=composer:2.7 /usr/bin/composer /usr/bin/composer

WORKDIR /var/www

COPY . .

RUN composer install --no-dev --optimize-autoloader

CMD ["php-fpm"]Code language: Dockerfile (dockerfile)
```

This builds a PHP-FPM container with Composer and required extensions. The app code is copied inside `/var/www`, ready to run.

3 – Define docker-compose.yml

Docker Compose orchestrates multiple containers: Laravel (PHP-FPM), MySQL, Redis, and Nginx as a reverse proxy.

```
# docker-compose.yml
```

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - .:/var/www
    networks:
      - laravel
    depends_on:
      - db
      - redis

  db:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: laravel
      MYSQL_ROOT_PASSWORD: root
    volumes:
      - dbdata:/var/lib/mysql
    networks:
      - laravel

  redis:
    image: redis:alpine
    networks:
      - laravel

  nginx:
    image: nginx:alpine
    volumes:
      - .:/var/www
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "8000:80"
    networks:
```

```
      - laravel
    depends_on:
      - app

volumes:
  dbdata:

networks:
  laravel:Code language: YAML (yaml)
```

This defines four services: `app` (Laravel), `db` (MySQL), `redis`, and `nginx`. The app depends on Redis and DB to start correctly. Nginx proxies requests to the app container.

4 – Configure Nginx

Add an Nginx config to route traffic into the Laravel container.

```
# nginx.conf
server {
    listen 80;
    index index.php index.html;
    root /var/www/public;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        fastcgi_pass app:9000;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_index index.php;
```

```
    }
}Code language: Nginx (nginx)
```

This config sends PHP requests to the `app` container (running PHP-FPM) and serves static assets directly. It mirrors production best practices.

5 – Run Containers

Start your Laravel environment with:

```
docker compose up -d --buildCode language: Bash (bash)
```

This builds containers and runs them in the background. Visit `http://localhost:8000` to access your Laravel app inside Docker.

6 – Scaling with Docker

One of Docker's strengths is scaling services horizontally. You can run multiple Laravel containers behind Nginx:

```
docker compose up --scale app=3 -dCode language: Bash (bash)
```

This runs three Laravel app containers in parallel, balancing requests through Nginx. Perfect for high-concurrency setups, especially when combined with Octane.

Wrapping Up

You've containerized Laravel with Docker, set up MySQL, Redis, and Nginx, and even scaled app containers horizontally. This setup ensures every developer runs the same environment and makes production deployment smoother. For even more scalability, integrate Docker with AWS ECS, Kubernetes, or DigitalOcean Droplets.

What's Next

- 10 Proven Ways to Optimize Laravel for High Traffic — combine Docker scaling with caching and queues.
- Optimizing Laravel for High Concurrency with Octane — supercharge Dockerized apps with in-memory request handling.
- Optimizing Laravel for AWS Deployment (Step-by-Step) — deploy your Dockerized Laravel app in the cloud.