Laravel Docker Production Deployment Guide

Complete Workflow for DigitalOcean VPS with Ubuntu

Table of Contents

Prerequisites

Before starting, ensure you have:

- A Laravel project ready for deployment
- A DigitalOcean account
- A domain name (optional but recommended)
- Basic knowledge of terminal/command line
- Git repository with your Laravel project

VPS Setup

Step 1: Create DigitalOcean Droplet

1. **Log into DigitalOcean Dashboard**
    - Go to digitalocean.com and sign in
    - Click "Create" → "Droplets"
2. **Configure Droplet Settings**

```
Image: Ubuntu 22.04 (LTS) x64
Plan: Basic
CPU options: Regular Intel ($12/month minimum recommended)
Datacenter: Choose closest to your users
Additional Options:
✓ IPv6
✓ Monitoring
```

3. **Add SSH Key (Recommended)**
   - Generate SSH key on your local machine:

```bash
ssh-keygen -t rsa -b 4096 -c "your-email@example.com"
```

- Add the public key to DigitalOcean

4. **Create Droplet**
   - Choose a hostname (e.g., `laravel-production`)
   - Click "Create Droplet"

Step 2: Initial Server Setup

1. **Connect to Your Server**

```bash
ssh root@your_server_ip
```

2. **Update System Packages**

```bash
apt update && apt upgrade -y
```

3. **Create Non-Root User**

```bash
adduser deployer
usermod -aG sudo deployer
```

4. **Configure SSH for New User**

```bash
rsync --archive --chown=deployer:deployer ~/.ssh /home/deployer
```

5. **Test New User Access**

```bash
ssh deployer@your_server_ip
```

Docker Installation

Step 3: Install Docker and Docker Compose

1. **Install Required Packages**

```bash
sudo apt install apt-transport-https ca-certificates curl software-properties-common -y
```

2. **Add Docker Repository**

```bash
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable
```

3. **Install Docker**

```bash
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io -y
```

4. **Install Docker Compose**

```bash
```

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(una
sudo chmod +x /usr/local/bin/docker-compose
```

5. **Add User to Docker Group**

```bash
sudo usermod -aG docker $USER
```

6. **Verify Installation**

```bash
docker --version
docker-compose --version
```

Laravel Project Preparation

Step 4: Prepare Your Laravel Project

1. **Clone Your Project**

```bash
cd /home/deployer
git clone https://github.com/your-username/your-laravel-project.git
cd your-laravel-project
```

2. **Create Production Environment File**

```bash
cp .env.example .env.production
```

3. **Configure Production Environment** Edit `.env.production`:

```env
```

```
APP_NAME="Your App Name"
APP_ENV=production
APP_KEY=base64:your-generated-key
APP_DEBUG=false
APP_URL=https://yourdomain.com

LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=error

DB_CONNECTION=pgsql
DB_HOST=postgres
DB_PORT=5432
DB_DATABASE=laravel_production
DB_USERNAME=laravel_user
DB_PASSWORD=your_secure_password

CACHE_DRIVER=redis
FILESYSTEM_DISK=local
QUEUE_CONNECTION=redis
SESSION_DRIVER=redis

REDIS_HOST=redis
REDIS_PASSWORD=null
REDIS_PORT=6379
```

Docker Configuration

Step 5: Create Docker Files

1. **Create Dockerfile** Create `Dockerfile` in project root:

```dockerfile
```

```
# Multi-stage build for production
FROM php:8.2-fpm-alpine AS base

# Install system dependencies
RUN apk add --no-cache \
    postgresql-dev \
    zip \
    unzip \
    git \
    curl \
    libpng-dev \
    libjpeg-turbo-dev \
    freetype-dev \
    oniguruma-dev \
    libxml2-dev \
    nginx \
    supervisor

# Install PHP extensions
RUN docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) \
    pdo_pgsql \
    mbstring \
    exif \
    pcntl \
    bcmath \
    gd \
    xml \
    zip

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

# Set working directory
WORKDIR /var/www/html

# Copy composer files
COPY composer.json composer.lock ./

# Install PHP dependencies
RUN composer install --no-dev --optimize-autoloader --no-scripts

# Copy application code
```

```
COPY . .

# Set permissions
RUN chown -R www-data:www-data /var/www/html \
    && chmod -R 755 /var/www/html/storage \
    && chmod -R 755 /var/www/html/bootstrap/cache

# Run composer scripts
RUN composer run-script post-autoload-dump

# Expose port
EXPOSE 9000

CMD ["php-fpm"]
```

2. **Create Docker Compose File** Create `docker-compose.production.yml`:

```
yaml
```

```yaml
version: '3.8'

services:
  # Laravel Application
  app:
    build:
      context: .
      dockerfile: Dockerfile
    restart: unless-stopped
    volumes:
      - ./storage:/var/www/html/storage
      - ./bootstrap/cache:/var/www/html/bootstrap/cache
    networks:
      - laravel_network
    depends_on:
      - postgres
      - redis

  # Nginx Web Server
  nginx:
    image: nginx:alpine
    restart: unless-stopped
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./:/var/www/html:ro
      - ./docker/nginx/nginx.conf:/etc/nginx/nginx.conf:ro
      - ./docker/nginx/default.conf:/etc/nginx/conf.d/default.conf:ro
      - ./docker/ssl:/etc/nginx/ssl:ro
    networks:
      - laravel_network
    depends_on:
      - app

  # PostgreSQL Database
  postgres:
    image: postgres:15-alpine
    restart: unless-stopped
    environment:
      POSTGRES_DB: laravel_production
      POSTGRES_USER: laravel_user
      POSTGRES_PASSWORD: your_secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - laravel_network

  # Redis Cache
  redis:
    image: redis:alpine
    restart: unless-stopped
    networks:
      - laravel_network

# Docker Networks
networks:
  laravel_network:
    driver: bridge

# Persistent Volumes
volumes:
  postgres_data:
```

3. **Create Nginx Configuration** Create directory and files:

```bash
mkdir -p docker/nginx
```

Create `docker/nginx/nginx.conf`:

```nginx
```

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile            on;
    tcp_nopush          on;
    tcp_nodelay         on;
    keepalive_timeout   65;
    types_hash_max_size 2048;

    include             /etc/nginx/mime.types;
    default_type        application/octet-stream;

    include /etc/nginx/conf.d/*.conf;
}
```

Create `docker/nginx/default.conf`:

```nginx
```

```
server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name yourdomain.com www.yourdomain.com;

    root /var/www/html/public;
    index index.php index.html index.htm;

    # SSL Configuration
    ssl_certificate /etc/nginx/ssl/cert.pem;
    ssl_certificate_key /etc/nginx/ssl/key.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-GCM-SH
    ssl_prefer_server_ciphers off;

    # Security Headers
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header Referrer-Policy "no-referrer-when-downgrade" always;
    add_header Content-Security-Policy "default-src * data: 'unsafe-eval' 'unsafe-inline'" always;

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

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

    location ~ /\.ht {
        deny all;
    }

    location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

SSL Certificate Setup

Step 6: Setup SSL with Let's Encrypt

1. **Install Certbot**

```bash
sudo apt install snapd
sudo snap install core; sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
```

2. **Generate SSL Certificate**

```bash
sudo certbot certonly --standalone -d yourdomain.com -d www.yourdomain.com
```

3. **Copy Certificates to Docker Volume**

```bash
mkdir -p docker/ssl
sudo cp /etc/letsencrypt/live/yourdomain.com/fullchain.pem docker/ssl/cert.pem
sudo cp /etc/letsencrypt/live/yourdomain.com/privkey.pem docker/ssl/key.pem
sudo chown -R deployer:deployer docker/ssl
```

4. **Setup Auto-Renewal**

```bash
sudo crontab -e
```

Add this line:

```
  0 12 * * * /usr/bin/certbot renew --quiet && cp /etc/letsencrypt/live/yourdomain.com/fullchain.pem
/home/deployer/your-laravel-project/docker/ssl/cert.pem && cp
/etc/letsencrypt/live/yourdomain.com/privkey.pem /home/deployer/your-laravel-project/docker/ssl/key.pem
&& docker-compose -f /home/deployer/your-laravel-project/docker-compose.production.yml restart nginx
```

Deployment Process

Step 7: Deploy Your Application

1. **Build and Start Containers**

```bash
cd /home/deployer/your-laravel-project
docker-compose -f docker-compose.production.yml up -d --build
```

2. **Generate Application Key**

```bash
docker-compose -f docker-compose.production.yml exec app php artisan key:generate --env=production
```

3. **Run Database Migrations**

```bash
docker-compose -f docker-compose.production.yml exec app php artisan migrate --env=production --force
```

4. **Cache Configuration**

```bash
docker-compose -f docker-compose.production.yml exec app php artisan config:cache
docker-compose -f docker-compose.production.yml exec app php artisan route:cache
docker-compose -f docker-compose.production.yml exec app php artisan view:cache
```

5. **Create Storage Link**

```bash
docker-compose -f docker-compose.production.yml exec app php artisan storage:link
```

Step 8: Create Deployment Script

Create `deploy.sh` for easier future deployments:

```bash
#!/bin/bash

echo "Starting deployment..."

# Pull latest changes
git pull origin main

# Build and restart containers
docker-compose -f docker-compose.production.yml down
docker-compose -f docker-compose.production.yml up -d --build

# Wait for containers to be ready
sleep 30

# Run Laravel commands
docker-compose -f docker-compose.production.yml exec app php artisan migrate --env=production --force
docker-compose -f docker-compose.production.yml exec app php artisan config:cache
docker-compose -f docker-compose.production.yml exec app php artisan route:cache
docker-compose -f docker-compose.production.yml exec app php artisan view:cache

echo "Deployment completed successfully!"
```

Make it executable:

```bash
chmod +x deploy.sh
```

Monitoring and Maintenance

Step 9: Setup Monitoring

1. **View Container Status**

```bash
```

```
docker-compose -f docker-compose.production.yml ps
```

2. **View Logs**

```bash
docker-compose -f docker-compose.production.yml logs -f app
docker-compose -f docker-compose.production.yml logs -f nginx
docker-compose -f docker-compose.production.yml logs -f postgres
```

3. **Setup Log Rotation** Create `/etc/logrotate.d/docker-containers`:

```
/var/lib/docker/containers/*/*.log {
  rotate 7
  daily
  compress
  size=1M
  missingok
  delaycompress
  copytruncate
}
```

Step 10: Backup Strategy

1. **Database Backup Script** Create `backup-db.sh`:

```bash
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
docker-compose -f docker-compose.production.yml exec -T postgres pg_dump -U laravel_user laravel_productio
# Upload to cloud storage (optional)
```

2. **Setup Automated Backups**

```bash
crontab -e
```

Add:

```
0 2 * * * /home/deployer/your-laravel-project/backup-db.sh
```

Troubleshooting

Common Issues and Solutions

1. **Permission Issues**

```bash
docker-compose -f docker-compose.production.yml exec app chown -R www-data:www-data /var/www/html/st
docker-compose -f docker-compose.production.yml exec app chmod -R 755 /var/www/html/storage
```

2. **Database Connection Issues**
   - Check if PostgreSQL container is running
   - Verify environment variables
   - Check network connectivity between containers
3. **SSL Certificate Issues**
   - Ensure domain points to your server IP
   - Check certificate paths in Nginx config
   - Verify certificate permissions
4. **Container Memory Issues**

```bash
docker system prune -a
docker volume prune
```

5. **View Container Resource Usage**

```bash
docker stats
```

Useful Commands

```bash
```

```
# Restart specific service
docker-compose -f docker-compose.production.yml restart nginx

# Execute commands in containers
docker-compose -f docker-compose.production.yml exec app php artisan tinker

# Update containers without downtime
docker-compose -f docker-compose.production.yml up -d --no-deps app

# Check container health
docker-compose -f docker-compose.production.yml exec app php artisan about

# Clear all caches
docker-compose -f docker-compose.production.yml exec app php artisan optimize:clear
```

Security Best Practices

1. **Firewall Configuration**

```bash
sudo ufw allow ssh
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

2. **Regular Updates**

```bash
# Update system packages
sudo apt update && sudo apt upgrade -y

# Update Docker images
docker-compose -f docker-compose.production.yml pull
docker-compose -f docker-compose.production.yml up -d
```

3. **Environment Security**
    - Use strong passwords for database
    - Keep .env files secure and never commit them

    - Regularly rotate secrets and keys
    - Monitor access logs

Conclusion

Your Laravel application is now deployed in production with:

- ✅ Docker containerization
- ✅ PostgreSQL database
- ✅ Nginx web server
- ✅ SSL encryption
- ✅ Automated deployment script
- ✅ Backup strategy
- ✅ Monitoring setup

The application should be accessible at `https://yourdomain.com`

For ongoing maintenance, use the deployment script and monitoring tools provided. Regular backups and security updates are essential for production environments.

Support: If you encounter issues, check the troubleshooting section or review container logs for specific error messages.