Complete Docker Setup Guide for Laravel + React + MySQL

📁 Project Structure

Create your project with this structure:

```
your-project/
├── docker-compose.yml
├── deploy.sh
├── nginx/
│   └── nginx.conf
├── backend/ (Laravel application)
│   ├── Dockerfile
│   ├── .env
│   └── ... (Laravel files)
├── frontend/ (React application)
│   ├── Dockerfile
│   ├── .env
│   └── ... (React files)
└── mysql/
    └── init/ (optional SQL files)
```

🚀 Quick Start

1. Clone or Create Your Applications

For Laravel:

bash

```bash
# If you don't have a Laravel app yet
composer create-project laravel/laravel backend
cd backend
composer install
```

For React:

bash

```bash
# If you don't have a React app yet
npx create-react-app frontend
cd frontend
npm install
```

2. Copy Docker Files

Copy the provided files to their respective locations:

- `docker-compose.yml` → root directory
- Laravel `Dockerfile` → `backend/Dockerfile`
- React `Dockerfile` → `frontend/Dockerfile`
- `nginx.conf` → `nginx/nginx.conf`
- Laravel `.env` → `backend/.env` (modify as needed)

3. Make Deploy Script Executable

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

4. Run Deployment

```bash
./deploy.sh
```

🔧 Manual Setup Steps

If you prefer to run commands manually:

1. Start the services

```bash
docker-compose up -d --build
```

2. Wait for MySQL to initialize (30 seconds)

3. Setup Laravel

bash

```bash
# Generate app key
docker-compose exec laravel php artisan key:generate

# Run migrations
docker-compose exec laravel php artisan migrate

# Cache configuration
docker-compose exec laravel php artisan config:cache

# Create storage link
docker-compose exec laravel php artisan storage:link
```

🌎 Environment Configuration

Laravel (.env)

Update your `backend/.env` file:

env

```env
DB_HOST=mysql
DB_DATABASE=laravel_db
DB_USERNAME=laravel_user
DB_PASSWORD=laravel_password
REDIS_HOST=redis
```

React (.env)

Create `frontend/.env`:

env

```env
REACT_APP_API_URL=http://localhost/api
```

📡 API Integration

In your React app, make API calls like this:

javascript

```javascript
// Example API call
const fetchData = async () => {
  try {
    const response = await fetch(`${process.env.REACT_APP_API_URL}/users`);
    const data = await response.json();
    return data;
  } catch (error) {
    console.error('API Error:', error);
  }
};
```

🔒 Laravel API Setup

1. Enable CORS (if not using Laravel Sanctum)

Install Laravel CORS:

bash

```bash
docker-compose exec laravel composer require fruitcake/laravel-cors
```

2. API Routes

In `backend/routes/api.php`:

php

```php
<?php

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::middleware('api')->group(function () {
    Route::get('/users', function () {
        return response()->json(['message' => 'Hello from Laravel API!']);
    });

    // Add your API routes here
});
```

🗄️ Database Management

Access MySQL directly:

bash

```
docker-compose exec mysql mysql -u laravel_user -p laravel_db
# Password: laravel_password
```

Run Laravel migrations:

bash

```
docker-compose exec laravel php artisan migrate
```

Seed database:

bash

```
docker-compose exec laravel php artisan db:seed
```

📊 Useful Commands

View logs:

bash

```
# All services
docker-compose logs -f

# Specific service
docker-compose logs -f laravel
docker-compose logs -f react
docker-compose logs -f mysql
```

Restart services:

bash

```
docker-compose restart
```

Stop all services:

bash

```
docker-compose down
```

Rebuild specific service:

bash

```
docker-compose up -d --build laravel
```

Execute commands in containers:

bash

```
# Laravel container
docker-compose exec laravel bash

# React container
docker-compose exec react sh

# MySQL container
docker-compose exec mysql bash
```

🌐 Production Deployment

1. Server Setup

On your server, install Docker using the Linux installation steps provided earlier.

2. Environment Variables

Update your `.env` files for production:

Laravel:

env

```
APP_ENV=production
APP_DEBUG=false
APP_URL=https://your-domain.com
```

React:

env

```
REACT_APP_API_URL=https://your-domain.com/api
```

3. SSL Configuration

Uncomment the HTTPS server block in `nginx.conf` and add your SSL certificates to `nginx/ssl/`.

4. Deploy

bash

```bash
# Clone your repository
git clone your-repo-url
cd your-project

# Make deploy script executable
chmod +x deploy.sh

# Run deployment
./deploy.sh
```

🔍 Troubleshooting

Common Issues:

1. **Port conflicts:** Change ports in `docker-compose.yml` if needed
2. **Permission issues:** Run `docker-compose exec laravel chown -R www-data:www-data /var/www/html/storage`
3. **Database connection:** Ensure MySQL container is fully started before Laravel
4. **CORS issues:** Check Laravel CORS configuration
5. **React not loading:** Verify `REACT_APP_API_URL` environment variable

Health Checks:

bash

```bash
# Check if containers are running
docker-compose ps

# Check container health
docker-compose exec laravel php artisan --version
docker-compose exec react npm --version
docker-compose exec mysql mysql --version
```

🔄 Development Workflow

1. Make changes to your Laravel/React code
2. Changes are automatically reflected (volumes are mounted)
3. For Laravel config changes, run: `docker-compose exec laravel php artisan config:cache`
4. For React dependency changes, rebuild: `docker-compose up -d --build react`

🎯 Next Steps

1. Set up Laravel Sanctum for API authentication
2. Configure email services (SMTP/Mailgun)
3. Set up automated backups for MySQL
4. Implement CI/CD pipeline
5. Configure monitoring and logging
6. Set up SSL certificates for production

Your application is now fully containerized and ready for development and deployment! 🚀