

PLAN DE ACCIÓN INMEDIATA - DEPLOY EASYPANEL

SITUACIÓN ACTUAL

Problema: Múltiples intentos de deploy en EasyPanel han fallado, no está claro en qué etapa exacta falla (backend, frontend, o configuración).

Solución: Estrategia de debugging sistemático paso a paso.

LO QUE ACABO DE CREAR PARA TI

1. 3 Dockerfiles Incrementales (para debugging paso a paso):

- `Dockerfile.step1-backend` → Solo prueba Prisma/DB
- `Dockerfile.step2-frontend` → Solo prueba Next.js build
- `Dockerfile.step3-full` → Build completo integrado

2. Script de Testing Automatizado:





- `test-dockerfiles.sh` → Ejecuta todos los tests localmente

3. Documentación Completa:

- `ESTRATEGIA_DEBUG_EASYPANEL.md` → Estrategia completa de debugging
 - `CONFIGURACION_EASYPANEL_CORRECTA.md` → Configuración exacta para EasyPanel
-

OPCIÓN 1: TESTING LOCAL PRIMERO (RECOMENDADO)

Por qué es mejor esta opción:

-  Detectas errores ANTES de gastar tiempo en EasyPanel
-  Identificas exactamente qué capa falla (backend vs frontend)
-  Builds locales son mucho más rápidos que remote
-  No contaminas production con deploys fallidos

Pasos:

1. Ejecutar script de testing:

```
cd /home/ubuntu/escalafin_mvp
./test-dockerfiles.sh
```

Este script:

- ✓ Construye `Dockerfile.step1-backend` (backend/Prisma)
- ✓ Construye `Dockerfile.step2-frontend` (Next.js build)

- ✓ Construye Dockerfile.step3-full (completo)
- ✓ Verifica healthcheck
- ✓ Verifica API health endpoint
- ✓ Te dice si está listo para production

1. Si todos los tests pasan:

```
# Push a GitHub
git add .
git commit -m "feat: dockerfiles paso a paso para deployment"
git push origin main
```

1. Configurar en EasyPanel:

- Seguir instrucciones exactas de `CONFIGURACION_EASYPANEL_CORRECTA.md`
- Usar `Dockerfile.step3-full`
- **IMPORTANTE:** Poner `/app` en "Ruta de compilación"

2. Deploy en EasyPanel con confianza

OPCIÓN 2: DEPLOY DIRECTO EN EASYPANEL (MÁS RÁPIDO PERO RIESGOSO)

Cuándo usar esta opción:

- Ya has hecho builds locales exitosos anteriormente
- Tienes prisa y quieres ir directo a production
- Confías en que el código está correcto

Pasos:

1. Push a GitHub (si aún no lo has hecho):


```
cd /home/ubuntu/escalafin_mvp
git add .
git commit -m "feat: dockerfiles optimizados para EasyPanel"
git push origin main
```

1. Configurar en EasyPanel UI:

Configuración del Repositorio:

```
Propietario: qghosting
Repositorio: escalafin-mvp
Rama: main
```

Configuración de Compilación:

```
Método de compilación: Dockerfile
Dockerfile: Dockerfile.step3-full
Ruta de compilación: /app  NO DEJAR VACÍO
```

Variables de Entorno (ejemplo mínimo):

```
DATABASE_URL=postgresql://user:pass@host:5432/escalafin?schema=public
NEXTAUTH_URL=https://tudominio.com
NEXTAUTH_SECRET=<generar-con-openssl>
NODE_ENV=production
NEXT_OUTPUT_MODE=standalone
SKIP_ENV_VALIDATION=1
```

1. **Crear PostgreSQL** en EasyPanel primero (si aún no existe)
2. **Deploy** de la app
3. **Monitorear logs** en EasyPanel UI:
 - Buscar “✅” para éxitos
 - Buscar “❌” para errores
 - Si falla, revisar EXACTAMENTE en qué línea

MI RECOMENDACIÓN PROFESIONAL

HACER ESTO:

1. **AHORA:** Ejecutar `./test-dockerfiles.sh` localmente
 - Toma 5-10 minutos
 - Te evita horas de debugging en EasyPanel
 - Identifica exactamente qué falla
2. **SI TODO PASA:** Push a GitHub y deploy en EasyPanel con confianza
3. **SI ALGO FALLA:**
 - Revisar logs de `/tmp/docker-stepX.log`
 - Corregir el problema específico
 - Re-ejecutar solo ese step

NO HACER ESTO:

- ❌ Ir directo a EasyPanel sin testear localmente
- ❌ Cambiar múltiples cosas a la vez sin saber qué funciona
- ❌ Intentar “adivinar” qué está mal
- ❌ Dejar “Ruta de compilación” vacío en EasyPanel



SI TEST-DOCKERFILES.SH FALLA

Error en Step 1 (Backend/Prisma):

Problema: Prisma schema inválido o dependencias faltantes

Solución:

```
cd app
npm install --legacy-peer-deps
npx prisma validate
npx prisma generate
```

Error en Step 2 (Frontend):

Problema: Next.js build falla o TypeScript errors

Solución:

```
cd app
npm run build
# Ver errores específicos y corregir
```

Error en Step 3 (Build completo):

Problema: Standalone no se genera o server.js falta

Solución: Verificar que `NEXT_OUTPUT_MODE=standalone` en `next.config.js`



SIGUIENTE PASO INMEDIATO

¿Qué prefieres?

A) Testing local primero (RECOMENDADO):

```
cd /home/ubuntu/escalafin_mvp
./test-dockerfiles.sh
```

Toma 5-10 minutos, pero te asegura que todo funciona ANTES de EasyPanel.

B) Ir directo a EasyPanel:

Seguir paso a paso `CONFIGURACION_EASYPANEL_CORRECTA.md`



RESPUESTA A TU PREGUNTA SOBRE “DEPLOY SISTEMATIZADO”

“el deploy se realiza de sistematizada o que me recomiendas”

Mi respuesta:

Sí, definitivamente usar el enfoque **SISTEMATIZADO** con estos Dockerfiles paso a paso.

Por qué:

- ☒ Separar backend y frontend te permite identificar EXACTAMENTE dónde falla
- ☒ No pierdes tiempo en builds largos que fallan al final
- ☒ Puedes debuggear cada capa independientemente
- ☒ Una vez que cada step pasa, el build completo tiene 99% de probabilidad de éxito

Proceso sistematizado:





1. Test Step 1 → Backend/Prisma ✓
2. Test Step 2 → Frontend/Next.js ✓
3. Test Step 3 → Build completo ✓
4. Deploy en EasyPanel con Dockerfile.step3-full ✓

Este es el approach profesional que usan equipos grandes de DevOps.

RESUMEN EJECUTIVO

Has tenido múltiples errores porque no había una forma de saber DÓNDE exactamente fallaba.

AHORA TIENES:

-  3 Dockerfiles incrementales para debugging preciso
-  Script automatizado para testear todo localmente
-  Documentación exacta para configurar EasyPanel
-  Estrategia clara: testear local → corregir → deploy production

PRÓXIMA ACCIÓN:

Ejecutar `./test-dockerfiles.sh` AHORA para ver exactamente qué funciona y qué no.

¿Quieres que ejecute el script de testing ahora mismo para ver los resultados?