

# Instrucciones para Probar el Build Corregido

**Fecha:** 16 de octubre de 2025

**Estado:**  Listo para probar

## Resumen del Problema y Solución

### El Problema

Tu build de Docker estaba fallando con el error:

```
ERROR: failed to solve: process ... did not complete successfully: exit code: 1
```

### La Solución

Hemos actualizado el Dockerfile para:

1. **Usar solo NPM** (más estable en Docker que Yarn)
2. **Limpiar cache** antes de instalar dependencias
3. **Optimizar flags** de instalación ( `--legacy-peer-deps` , `--prefer-offline` , etc.)
4. **Crear un Dockerfile simplificado** alternativo más robusto






## Cómo Probar el Build (3 Opciones)

### Opción 1: Usar el Script Automático (RECOMENDADO)

El script te guiará paso a paso:

```
cd /ruta/a/escalafin_mvp
./test-build-quick.sh
```

El script:

-  Verifica prerequisites (Docker, espacio en disco)
-  Te permite elegir qué Dockerfile usar
-  Construye la imagen con logging detallado
-  Guarda logs en archivos para debug
-  Verifica que el build fue exitoso

### Opción 2: Build Manual con Dockerfile.simple (MÁS CONFIABLE)



Este es el más simple y robusto:

```
cd /ruta/a/escalafin_mvp

# Build
docker build -f Dockerfile.simple -t escalafin:latest .

# Verificar
docker images | grep escalafin

# Ejecutar (ajusta las variables de entorno)
docker run -p 3000:3000 \
  -e DATABASE_URL="postgresql://user:pass@localhost:5432/escalafin" \
  -e NEXTAUTH_SECRET="tu-secret-seguro-min-32-caracteres" \
  -e NEXTAUTH_URL="http://localhost:3000" \
  escalafin:latest
```

### Opción 3: Build Manual con Dockerfile Principal

Si prefieres usar el Dockerfile principal actualizado:

```
cd /ruta/a/escalafin_mvp

# Build con logs detallados
docker build --progress=plain -t escalafin:main . 2>&1 | tee build-main.log

# Verificar
docker images | grep escalafin
```

### Archivos Actualizados/Creados

Archivo	Descripción	Estado
Dockerfile	Dockerfile principal actualizado (solo NPM)	✓ Actualizado
Dockerfile.simple	Versión simplificada y más robusta	✓ Nuevo
SOLUCION_ERROR_DOCKER_BUILD.md	Documentación completa del problema	✓ Nuevo
test-build-quick.sh	Script de prueba automático	✓ Nuevo
Este archivo	Instrucciones rápidas	✓ Nuevo

## Qué Buscar Durante el Build

### ✓ Señales de Éxito

Durante el build, deberías ver:

```
=== Instalando dependencias con NPM ===
Limpiando cache de npm...
Instalando dependencias...
added 347 packages in 45s
✓ Dependencias instaladas correctamente
```

Y al final:

```
=== Verificando build standalone ===
✓ Standalone output verificado

[+] Building 245.6s (23/23) FINISHED
```

### ✗ Si Continúa Fallando

#### 1. Verifica el log completo:

```
bash
cat build-simple.log # o build-main.log
```

#### 2. Busca la línea exacta del error:

```
bash
grep -i "error" build-simple.log
```

#### 3. Verifica espacio en disco:

```
bash
df -h
```

Necesitas al menos 5GB libres.

#### 4. Limpia recursos de Docker:

```
bash
docker system prune -a --volumes
```

## Pruebas Post-Build

Una vez que el build sea exitoso, ejecuta estas pruebas:

### 1. Verificar la Imagen

```
# Ver imágenes creadas
docker images | grep escalafin

# Inspeccionar layers
docker history escalafin:latest

# Ver tamaño
docker images escalafin:latest --format "{{.Size}}"
```

## 2. Ejecutar Contenedor de Prueba

```
# Con variables mínimas
docker run -p 3000:3000 \
  -e DATABASE_URL="postgresql://escalafin:password@host:5432/escalafin" \
  -e NEXTAUTH_SECRET="tu-secret-muy-seguro-de-minimo-32-caracteres" \
  -e NEXTAUTH_URL="http://localhost:3000" \
  escalafin:latest
```

## 3. Verificar Health Check

En otra terminal:

```
# Espera 30 segundos para que arranque
sleep 30

# Verifica el health endpoint
curl http://localhost:3000/api/health

# Deberías ver: {"status":"ok"}
```

## 4. Acceder a la Aplicación

Abre tu navegador en: <http://localhost:3000>



## Usar con Docker Compose

Si prefieres usar Docker Compose:

```
# docker-compose.yml
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile.simple # o Dockerfile
    ports:
      - "3000:3000"
    environment:
      DATABASE_URL: postgresql://user:pass@db:5432/escalafin
      NEXTAUTH_SECRET: tu-secret-muy-seguro
      NEXTAUTH_URL: http://localhost:3000
    depends_on:
      - db

  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: escalafin
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

Luego:

```
docker-compose up --build
```

## Comparación de Dockerfiles

Característica	Dockerfile	Dockerfile.simple
Complejidad	Media	Baja ★
Estabilidad	Alta	Muy Alta ★★
Velocidad	Rápida	Rápida
Mantenibilidad	Media	Alta ★
<b>Recomendado para</b>	Producción	Desarrollo y Producción ★★

## Próximos Pasos

### Una Vez que el Build Funcione:

#### 1. Push a Docker Registry:

```
bash
docker tag escalafin:latest tu-registry.com/escalafin:latest
docker push tu-registry.com/escalafin:latest
```

#### 2. Desplegar en Coolify:

- Usa la imagen que acabas de construir
- Configura las variables de entorno en Coolify
- Sigue `COOLIFY_DEPLOYMENT_GUIDE.md`

#### 3. Automatizar con CI/CD:

```
yaml
# .github/workflows/docker-build.yml
- name: Build Docker Image
  run: docker build -f Dockerfile.simple -t escalafin:${{ github.sha }} .
```

## Troubleshooting Avanzado

### Error: “npm ERR! network”

```
# Aumenta el timeout
npm config set fetch-timeout 600000

# 0 en el Dockerfile, agrega:
RUN npm install --fetch-timeout=600000
```

### Error: “exit code 137” (Memoria insuficiente)

```
# Aumenta memoria de Docker
docker build --memory=4g --memory-swap=4g -t escalafin:latest .
```

### Error: “ENOSPC” (Sin espacio)

```
# Limpia todo
docker system prune -a --volumes

# Verifica espacio
df -h
```

## El build funciona pero el contenedor no arranca

```
# Ver logs
docker logs <container-id>

# Ejecutar shell dentro del contenedor
docker run -it --entrypoint sh escalafin:latest

# Verificar archivos
ls -la
ls -la .next/standalone/
```



## Recursos Adicionales

### Documentación del Proyecto

- `SOLUCION_ERROR_DOCKER_BUILD.md` : Documentación técnica completa
- `COOLIFY_DEPLOYMENT_GUIDE.md` : Guía de despliegue
- `MULTI_INSTANCE_GUIDE.md` : Despliegue multi-instancia
- `docker-compose.yml` : Configuración de servicios

### Comandos Útiles

```
# Ver tamaño de cada layer
docker history escalafin:latest

# Inspeccionar la imagen
docker inspect escalafin:latest

# Exportar la imagen
docker save escalafin:latest | gzip > escalafin-latest.tar.gz

# Importar en otro servidor
docker load < escalafin-latest.tar.gz
```







## Checklist de Verificación

Antes de considerar el build completo:

- ☐ El build completa sin errores
- ☐ La imagen se crea correctamente
- ☐ El contenedor arranca sin problemas
- ☐ El health check responde OK
- ☐ Puedes acceder a `http://localhost:3000`
- ☐ Las migraciones de Prisma funcionan
- ☐ Los logs no muestran errores críticos
- ☐ La aplicación es funcional

## Si Todo Funciona

¡Felicidades! Tu aplicación está lista para:

1.  **Desarrollo local** con Docker
2.  **Despliegue en Coolify**
3.  **CI/CD automático**
4.  **Múltiples instancias**

## Si Necesitas Ayuda

1. **Revisa los logs completos:** `build-*.log`
2. **Consulta la documentación:** `SOLUCION_ERROR_DOCKER_BUILD.md`
3. **Verifica prerequisites:** Docker 20.10+, 5GB+ espacio
4. **Prueba la versión simplificada:** `Dockerfile.simple`

**Última actualización:** 16 de octubre de 2025

**Versión:** 1.0

**Estado:**  Listo para usar

## Comando Rápido para Empezar

```
# Opción más simple y rápida:
cd /ruta/a/escalafin_mvp
docker build -f Dockerfile.simple -t escalafin:latest . && \
docker run -p 3000:3000 \
  -e DATABASE_URL="postgresql://user:pass@localhost:5432/escalafin" \
  -e NEXTAUTH_SECRET="$(openssl rand -base64 32)" \
  -e NEXTAUTH_URL="http://localhost:3000" \
  escalafin:latest
```

¡Ya está! 