



Solución al Error de Build en Coolify

Fecha: 16 de octubre de 2025

Error: Build falla durante instalación de dependencias con exit code 1

Causa: Dockerfile.coolify usaba lógica condicional Yarn/NPM inestable



Problema Identificado

El error ocurría en esta sección del `Dockerfile.coolify`:

```
# ❌ CÓDIGO PROBLEMÁTICO (versión anterior)
RUN if [ -f yarn.lock ]; then \
    yarn install --frozen-lockfile --network-timeout 300000 --production=false; \
else \
    npm ci --legacy-peer-deps; \
fi
```

Por qué fallaba:

- **Yarn en Alpine Linux** puede ser inestable
- **Lógica condicional** añade complejidad innecesaria
- **yarn.lock como symlink** puede causar problemas en algunos contextos
- **npm ci** puede fallar si package-lock.json no está perfectamente sincronizado



Solución Aplicada

1. Actualizado `Dockerfile.coolify`

ANTES (líneas 23-28):

```
RUN if [ -f yarn.lock ]; then \
    yarn install --frozen-lockfile --network-timeout 300000 --production=false; \
else \
    npm ci --legacy-peer-deps; \
fi
```

DESPUÉS (versión 11.0):

```
# Instalar dependencias SOLO con NPM
RUN echo "=== Instalando dependencias con NPM ===" && \
    echo "Limpiando cache..." && \
    npm cache clean --force && \
    echo "Instalando todas las dependencias (dev + prod)..." && \
    npm install --legacy-peer-deps --prefer-offline --no-audit --progress=false && \
    echo "✅ Dependencias instaladas correctamente"
```

2. Cambios en el Build

ANTES:

```
RUN yarn build
```

DESPUÉS:

```
RUN echo "=== Building Next.js ===" && \  
  npm run build && \  
  echo "✅ Build completado"
```

Cómo Aplicar la Solución

Opción A: Push desde local (Recomendado)

```
cd /home/ubuntu/escalafin_mvp  
  
# 1. Verificar cambios  
git diff Dockerfile.coolify  
  
# 2. Agregar cambios  
git add Dockerfile.coolify  
  
# 3. Commit  
git commit -m "fix: actualizar Dockerfile.coolify para usar solo NPM (v11.0)"  
  
# 4. Push al repositorio  
git push origin main
```

Opción B: Re-deploy desde Coolify

1. Ir a **Coolify** (adm.escalafin.com)
2. **Seleccionar el proyecto** EscalaFin
3. **Hacer Pull** del repositorio actualizado
4. **Re-deploy** la aplicación

El nuevo Dockerfile será usado automáticamente.



Ventajas de la Nueva Versión

Aspecto	Antes	Después
Gestor de paquetes	Yarn/NPM (condicional)	NPM (único)
Estabilidad	Media	Alta
Velocidad de build	Variable	Consistente
Debugging	Difícil	Fácil (logs claros)
Cache	No optimizado	Optimizado con <code>--prefer-offline</code>
Complejidad	Alta (if/else)	Baja (comandos directos)



Verificación Post-Deployment

Después del nuevo deployment, verificar:

1. Logs de Build

```
# En Coolify, revisar logs de build
# Debe mostrar:
✓ === Instalando dependencias con NPM ===
✓ Limpiando cache...
✓ Instalando todas las dependencias (dev + prod)...
✓ [✓] Dependencias instaladas correctamente
✓ === Generando Prisma Client ===
✓ [✓] Prisma Client generado
✓ === Building Next.js ===
✓ [✓] Build completado
```

2. Verificar Aplicación

```
# Verificar que el contenedor esté corriendo
curl -I https://demo.escalafin.com

# Debe retornar:
HTTP/2 200
```

Si Persiste el Error

1. Verificar que Git esté actualizado

```
# En el servidor de Coolify
cd /ruta/al/repo
git log -1 --oneline

# Debe mostrar el commit "fix: actualizar Dockerfile.coolify..."
```

2. Limpiar Build Cache

En Coolify:




- Settings → Build Settings → Clear Build Cache
- Re-deploy la aplicación

3. Verificar Variables de Entorno

Asegurar que estén configuradas:

- DATABASE_URL
- NEXTAUTH_URL
- NEXTAUTH_SECRET
- AWS_* variables
- OPENPAY_* variables

Archivos Modificados

-  Dockerfile.coolify - Actualizado a v11.0 (solo NPM)
 -  Dockerfile - Ya estaba correcto (v10.0)
 -  Dockerfile.simple - Alternativa simple (existente)
-

Dockerfile.coolify - Versión Completa Actualizada

```

# ESCALAFIN MVP - DOCKERFILE OPTIMIZADO PARA COOLIFY
# Versión: 11.0 - Solo NPM (más estable en Docker)
# Fecha: 2025-10-16

FROM node:18-alpine AS base

# Install system dependencies
RUN apk add --no-cache \
    libc6-compat \
    curl \
    dumb-init \
    openssl \
    && rm -rf /var/cache/apk/*

# Create app directory and user
WORKDIR /app
RUN addgroup --system --gid 1001 nodejs && \
    adduser --system --uid 1001 nextjs

# Set environment variables
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV PORT=3000
ENV HOSTNAME="0.0.0.0"

# ===== STAGE 1: Instalar dependencias =====
FROM base AS deps
WORKDIR /app

# Copy package files
COPY app/package.json app/package-lock.json* app/yarn.lock* ./

# Instalar dependencias SOLO con NPM
RUN echo "=== Instalando dependencias con NPM ===" && \
    echo "Limpiando cache..." && \
    npm cache clean --force && \
    echo "Instalando todas las dependencias (dev + prod)..." && \
    npm install --legacy-peer-deps --prefer-offline --no-audit --progress=false && \
    echo "✅ Dependencias instaladas correctamente"

# ===== STAGE 2: Build =====
FROM base AS builder
WORKDIR /app

# Copy dependencies
COPY --from=deps /app/node_modules ./node_modules

# Copy source code
COPY app/ .

# Variables de entorno para el build
ENV SKIP_ENV_VALIDATION=true

# Generate Prisma client
RUN echo "=== Generando Prisma Client ===" && \
    npx prisma generate && \
    echo "✅ Prisma Client generado"

# Build Next.js
RUN echo "=== Building Next.js ===" && \
    npm run build && \
    echo "✅ Build completado"

```

```
# ===== STAGE 3: Production =====
FROM base AS runner
WORKDIR /app

# Copy built application
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./next/static
COPY --from=builder --chown=nextjs:nodejs /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/.prisma ./
node_modules/.prisma
COPY --from=builder --chown=nextjs:nodejs /app/prisma ./prisma

# Create necessary directories
RUN mkdir -p /app/uploads && \
    chown -R nextjs:nodejs /app/uploads





# Switch to nextjs user
USER nextjs

# Expose port
EXPOSE 3000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=60s --retries=3 \
    CMD curl -f http://localhost:3000/api/health || exit 1

# Start application
CMD ["dumb-init", "node", "server.js"]
```

Próximos Pasos

1.  **Hacer push** del Dockerfile.coolify actualizado
 2.  **Re-deploy** en Coolify
 3.  **Verificar** logs de build
 4.  **Probar** la aplicación desplegada
-

Nota: Este mismo enfoque (usar solo NPM) se aplicó previamente a los otros Dockerfiles (Dockerfile, Dockerfile.simple) y ha demostrado ser mucho más estable en entornos de producción.