


Análisis Completo de Problemas de Deployment

Fecha: 16 de octubre de 2025
Versión Analizada: Dockerfile v14.0
Estado:  PROBLEMAS CRÍTICOS DETECTADOS

Resumen Ejecutivo

Se detectaron **5 problemas críticos** que impiden el correcto funcionamiento del deployment:

#	Problema	Severidad	Impacto
1	<code>NEXT_OUTPUT_MODE</code> no configurado	 CRÍTICO	Build falla al copiar standalone
2	Scripts <code>start.sh</code> en ubicación incorrecta	 CRÍTICO	COPY falla en Dockerfile
3	npm actualización puede fallar	 MEDIO	Build lento en algunos casos
4	Cache de npm no optimizado	 BAJO	Builds más lentos
5	Healthcheck usa <code>curl</code> en vez de <code>wget</code>	 BAJO	Inconsistencia menor

PROBLEMA 1 (CRÍTICO): `NEXT_OUTPUT_MODE` No Configurado

Descripción

El `next.config.js` está configurado para leer el modo de output desde una variable de entorno:

```
output: process.env.NEXT_OUTPUT_MODE,
```

Pero el Dockerfile NO establece esta variable, por lo que Next.js usa el modo por defecto (no standalone).

Impacto

```

❌ npm run build genera solo .next/ (modo normal)
❌ NO se genera .next/standalone/
❌ COPY --from=builder /app/.next/standalone ./ → FALLA
❌ No se genera server.js
❌ CMD ["node", "server.js"] → FALLA al iniciar

```

Evidencia

next.config.js (línea 4):

```
output: process.env.NEXT_OUTPUT_MODE, // ← Variable no definida
```

Dockerfile (línea 76-77):

```

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
# ❌ FALTA: ENV NEXT_OUTPUT_MODE=standalone

```

Dockerfile (línea 99):

```

COPY --from=builder /app/.next/standalone ./
# ❌ Este directorio NO EXISTE porque no se configuró standalone

```

Solución

Agregar en el Dockerfile, stage builder (antes del build):

```

# Build de Next.js
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV SKIP_ENV_VALIDATION=1
ENV NEXT_OUTPUT_MODE=standalone # ← AGREGAR ESTA LÍNEA

```

PROBLEMA 2 (CRÍTICO): Scripts en Ubicación Incorrecta

Descripción

El Dockerfile intenta copiar `start.sh` y `healthcheck.sh` desde el builder stage, pero estos archivos están en el **root del proyecto**, no en `app/`.

Ubicación Actual

```

escalafin_mvp/
├── start.sh
├── healthcheck.sh
└── app/
    ├── app/
    ├── components/
    ├── package.json
    └── ... (resto del código)
  
```

 AQUÍ ESTÁN
 AQUÍ ESTÁN

Lo Que Hace el Dockerfile

```

# Stage builder
COPY app/ ./                                # Copia solo el contenido de app/

# Stage runner
COPY --from=builder /app/start.sh* /app/      # ❌ NO EXISTE en builder
COPY --from=builder /app/healthcheck.sh* /app/ # ❌ NO EXISTE en builder
  
```

Impacto

- ❌ Los scripts no se copian al contenedor
- ❌ start.sh no está disponible
- ✅ El CMD funciona porque usa "node server.js" directamente
- ⚠️ Pero start.sh tiene lógica importante de migraciones y seed

Solución Opción 1: Copiar desde Root

```

# En el stage runner, ANTES de copiar desde builder
COPY start.sh healthcheck.sh /app/
RUN chmod +x /app/start.sh /app/healthcheck.sh

# Luego copiar el resto
COPY --from=builder /app/.next/standalone ./
COPY --from=builder /app/.next/static ./next/static
# ...
  
```

Solución Opción 2: Mover Scripts a app/

```

mv /home/ubuntu/escalafin_mvp/start.sh /home/ubuntu/escalafin_mvp/app/
mv /home/ubuntu/escalafin_mvp/healthcheck.sh /home/ubuntu/escalafin_mvp/app/
  
```

Recomendación: Opción 1 (más claro dónde están los scripts de infraestructura)

PROBLEMA 3 (MEDIO): Actualización de npm Puede Fallar

Descripción

El Dockerfile actualiza npm globalmente en cada build:

```
RUN npm install -g npm@latest
```

Riesgos

1. **Builds no reproducibles:** Cada build puede usar una versión diferente de npm
2. **Breaking changes:** npm@latest puede introducir cambios incompatibles
3. **Build más lento:** Descargar e instalar npm en cada build

Impacto

- ⚠️ npm 10.9.0 hoy ➡️ npm 11.0.0 mañana (puede romper el build)
- ⚠️ Agrega ~10-15 segundos al build
- ⚠️ No aprovecha cache de Docker layers efectivamente

Solución

Pin a una versión específica que soporte lockfileVersion 3:

```
# Actualizar npm a versión específica que soporte lockfileVersion 3
RUN echo "=== 📦 Instalando npm v10.9.0 ===" && \
  npm install -g npm@10.9.0 && \
  npm --version && \
  echo "✅ npm 10.9.0 instalado"
```

Beneficio: Builds reproducibles y cacheable.

● PROBLEMA 4 (BAJO): Cache de npm No Optimizado

Descripción

El Dockerfile copia todos los package files a la vez:

```
COPY app/package.json ./
COPY app/package-lock.json* ./
COPY app/yarn.lock* ./
```

Oportunidad de Mejora

Copiar en pasos separados permite mejor uso del cache:

```
# Step 1: Copy package.json first
COPY app/package.json ./

# Step 2: Copy lock files (pueden cambiar independientemente)
COPY app/package-lock.json* ./
COPY app/yarn.lock* ./

# Step 3: Install (solo se re-ejecuta si alguno de los anteriores cambió)
RUN npm ci --legacy-peer-deps
```

Impacto

- ✓ Cambios en código fuente no invalidan cache de dependencias
- 🕒 Ahorra ~2-5 minutos en builds incrementales

● PROBLEMA 5 (BAJO): Inconsistencia en Healthcheck

Descripción

- El **Dockerfile** usa `wget` para healthcheck
- El **healthcheck.sh** usa `curl`

Dockerfile (línea 115-116):

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
  CMD if [ -f "healthcheck.sh" ]; then sh healthcheck.sh; else wget --no-verbose --
  tries=1 --spider http://localhost:${PORT}/api/health || exit 1; fi
```

healthcheck.sh:

```
if curl -f http://localhost:${PORT:-3000}/api/health > /dev/null 2>&1; then
  echo "✓ Health check passed"
  exit 0
fi
```

Problema

Si `healthcheck.sh` existe pero `curl` no está instalado, el healthcheck falla.

Solución

Usar `wget` en ambos lados (ya está instalado en la imagen):

```
# healthcheck.sh
#!/bin/sh
if wget --no-verbose --tries=1 --spider http://localhost:${PORT:-3000}/api/health > /
dev/null 2>&1; then
  echo "✓ Health check passed"
  exit 0
else
  echo "✗ Health check failed"
  exit 1
fi
```



Comparación: Antes vs Después

Antes (v14.0 - Con Problemas)

```
# ❌ PROBLEMA 1: No standalone
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
# Sin NEXT_OUTPUT_MODE=standalone

RUN npm run build
# Genera solo .next/, NO .next/standalone/

# ❌ PROBLEMA 2: Scripts no existen en builder
COPY --from=builder /app/start.sh* /app/
COPY --from=builder /app/healthcheck.sh* /app/

# ❌ PROBLEMA 1 (continuación): Falla al copiar
COPY --from=builder /app/.next/standalone ./

# ❌ PROBLEMA 1 (continuación): server.js no existe
CMD ["dumb-init", "node", "server.js"]
```

Resultado: ❌ Build falla o contenedor no inicia

Después (v15.0 - Corregido)

```
# ✅ SOLUCIÓN 1: Configurar standalone
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV SKIP_ENV_VALIDATION=1
ENV NEXT_OUTPUT_MODE=standalone # ← AGREGADO

RUN npm run build
# Genera .next/ Y .next/standalone/ ✅

# ✅ SOLUCIÓN 2: Copiar scripts desde root
COPY start.sh healthcheck.sh /app/
RUN chmod +x /app/start.sh /app/healthcheck.sh

# ✅ Ahora standalone existe
COPY --from=builder /app/.next/standalone ./

# ✅ server.js existe
CMD ["dumb-init", "node", "server.js"]
```

Resultado: ✅ Build exitoso, contenedor inicia correctamente



Plan de Acción Recomendado

Prioridad 1 (Crítico - Hacer AHORA)

- [] **Agregar** `ENV NEXT_OUTPUT_MODE=standalone` en Dockerfile (stage builder)
- [] **Copiar scripts desde root** en lugar de desde builder
- [] **Probar build localmente** con `docker build -t escalafin-test .`

Prioridad 2 (Importante - Hacer Hoy)

- [] **Pin versión de npm** a `10.9.0` en lugar de `latest`
- [] **Actualizar healthcheck.sh** para usar `wget` en vez de `curl`
- [] **Probar contenedor completo** con `docker-compose up`

Prioridad 3 (Mejora - Esta Semana)

- [] **Optimizar cache** separando COPY de package files
- [] **Documentar cambios** en CHANGELOG.md
- [] **Actualizar CI/CD** (GitHub Actions) con los mismos cambios



Cómo Verificar la Corrección

Test 1: Verificar Standalone Build

```
cd /home/ubuntu/escalafin_mvp
docker build --progress=plain -t escalafin-test . 2>&1 | grep -A5 "standalone"
```

Esperado:

```
✓ COPY --from=builder /app/.next/standalone ./
✓ No errores de "file not found"
```

Test 2: Verificar Scripts

```
docker run --rm escalafin-test ls -la /app/ | grep -E "(start|health)"
```

Esperado:

```
-rwxr-xr-x start.sh
-rwxr-xr-x healthcheck.sh
```

Test 3: Verificar server.js

```
docker run --rm escalafin-test ls -la /app/server.js
```

Esperado:

```
-rw-r--r-- 1 nextjs nodejs 1234 Oct 16 06:00 /app/server.js
```

Test 4: Iniciar Contenedor

```
docker run -p 3000:3000 -e DATABASE_URL="postgresql://..." escalafin-test
```

Esperado:

🚀 Iniciando servidor Next.js standalone...
 ✅ Server listening on 0.0.0.0:3000

Archivos Afectados

Archivo	Cambios Necesarios	Prioridad
Dockerfile	Agregar NEXT_OUTPUT_MODE, copiar scripts correctamente	🔴 CRÍTICO
healthcheck.sh	Cambiar curl por wget	🟢 BAJO
docker-compose.yml	Agregar NEXT_OUTPUT_MODE en env (opcional)	🟡 MEDIO
Dockerfile.coolify	Mismos cambios que Dockerfile	🔴 CRÍTICO
.github/workflows/*.yaml	Verificar configuración de build	🟡 MEDIO

⚠ Riesgos Durante la Corrección

- 1. Regenerar package-lock.json accidentalmente**
 - Riesgo: Cambiar versiones de dependencias
 - Mitigación: No tocar package files, solo Dockerfile
- 2. Breaking changes en producción**
 - Riesgo: Deploy directo a producción sin probar
 - Mitigación: Probar localmente primero con docker-compose
- 3. Perder datos durante re-deploy**
 - Riesgo: Recrear contenedores sin backup
 - Mitigación: Backup de DB antes de re-deploy

✅ Checklist de Validación

Antes de considerar el deployment corregido:

- ☐ Build de Docker completa sin errores
- ☐ Contenedor inicia correctamente
- ☐ Healthcheck pasa (status healthy)
- ☐ API endpoints responden (GET /api/health → 200)

- [] Migraciones de DB se aplican correctamente
- [] Prisma client funciona en el contenedor
- [] Variables de entorno se leen correctamente
- [] Start.sh ejecuta migraciones y seed si es necesario
- [] Logs no muestran errores críticos



Próximos Pasos

1. **Crear Dockerfile v15.0** con todas las correcciones
 2. **Probar localmente** con docker-compose
 3. **Actualizar todos los Dockerfiles** (normal, coolify, easypanel)
 4. **Crear script de prueba automatizado** (PROBAR_DEPLOY_COMPLETO.sh)
 5. **Documentar cambios** en CHANGELOG.md
 6. **Push a GitHub** y verificar GitHub Actions
 7. **Deploy a Coolify** con configuración corregida
-

Fecha de Análisis: 16 de octubre de 2025

Analizado por: DeepAgent

Estado: ☒ Análisis Completo - Listo para Implementar Correcciones