Dockerfile Simplificado para Debugging

Cambio de Estrategia

Después de múltiples intentos con captura de exit codes y PIPESTATUS, decidí **simplificar completamente** el Dockerfile para que los errores sean más visibles.

Versión Anterior (Compleja)

```
SHELL ["/bin/bash", "-c"]
RUN npm run build 2>&1 | tee build.log; \
    BUILD_EXIT_CODE=${PIPESTATUS[0]}; \
    if [ $BUILD_EXIT_CODE -ne 0 ]; then \
        echo "  Build failed!"; \
        tail -50 build.log; \
        exit 1; \
    fi && \
    # más verificaciones...
```

Problemas: - Demasiado complejo - Dependía de bash-specific features - Los errores no eran claros - Difícil de debuggear

---

Versión Nueva (Simplificada)

```
# Generate Prisma client first
RUN echo "  Generating Prisma client..." && \
    npx prisma generate && \
    echo "  Prisma client generated"

# Build Next.js (simplified - let npm handle errors)
RUN echo "  Building Next.js application (NORMAL mode, no standalone)..." && \
    npm run build && \
    echo "  Build completed successfully!"

# Verify build output
RUN echo "  Verifying build output..." && \
    if [ -f ".next/BUILD_ID" ]; then \
        echo "  Build ID found: $(cat .next/BUILD_ID)"; \
    else \
        echo "  BUILD_ID not found!"; \
        echo "Contents of .next directory:"; \
        ls -la .next/ || echo "No .next directory!"; \
```

```
        exit 1; \
    fi
```

Ventajas: 1. **Simple y directo**: Un RUN para build, uno para verificación 2. **Output claro**: Docker muestra todo el output de `npm run build` 3. **Errores visibles**: Si falla, verás exactamente qué línea y por qué 4. **Fácil de debuggear**: Cada paso es independiente

---

Cómo Funciona Ahora

Paso 1: Generar Prisma Client

```
RUN echo "  Generating Prisma client..." && \
    npx prisma generate && \
    echo "  Prisma client generated"
```

- Si falla: Docker detiene aquí y muestra el error de Prisma

Paso 2: Build de Next.js

```
RUN echo "  Building Next.js application..." && \
    npm run build && \
    echo "  Build completed successfully!"
```

- Si falla: Docker detiene aquí y muestra **TODO** el output de Next.js
- No hay redirección, no hay captura de output
- Verás exactamente qué archivo o paso causó el error

Paso 3: Verificar BUILD_ID

```
RUN echo "  Verifying build output..." && \
    if [ -f ".next/BUILD_ID" ]; then \
        echo "  Build ID found: $(cat .next/BUILD_ID)"; \
    else \
        echo "  BUILD_ID not found!"; \
        ls -la .next/; \
        exit 1; \
    fi
```

- Solo se ejecuta si el build tuvo éxito
- Si BUILD_ID no existe, muestra el contenido de `.next/`

Logs Esperados

Build Exitoso

```
Step X: RUN echo "  Generating Prisma client..."
```

```
  Generating Prisma client...
Prisma schema loaded from prisma/schema.prisma
  Prisma client generated

Step Y: RUN echo "  Building Next.js application..."
  Building Next.js application (NORMAL mode, no standalone)...
  Next.js 14.2.28
    Creating an optimized production build ...
   Compiled successfully
   Generating static pages (26/26)
  Build completed successfully!

Step Z: RUN echo "  Verifying build output..."
  Verifying build output...
  Build ID found: Pd5YqrYg5fXUh8wYTj4kq
```

Build Fallido

```
Step Y: RUN echo "  Building Next.js application..."
  Building Next.js application (NORMAL mode, no standalone)...
  Next.js 14.2.28
    Creating an optimized production build ...
Error: Cannot find module '@/lib/something'
    at Module._resolveFilename (node:internal/modules/cjs/loader.js:...)
    [stack trace completo aquí]

ERROR: failed to build: exit code: 1
```

Ahora verás exactamente qué causó el error!

Por Qué Esta Versión Es Mejor

Aspecto	Versión Anterior	Versión Nueva
Complejidad	Alta (PIPESTATUS, pipes, etc)	Baja (comandos directos)
Visibilidad de errores	Parcial (solo últimas 50 líneas)	Total (todo el output)
Debugging	Difícil	Fácil
Dependencias	Requiere Bash	Funciona con sh o bash
Mantenimiento	Difícil de entender	Claro y simple

Próximos Pasos

1. **Redeploy en Coolify** con esta versión simplificada
2. **Observa los logs** - ahora deberían ser mucho más claros
3. **Si falla**, comparte **TODO el output** desde "Step X" hasta el error
4. Con el output completo, podré identificar el problema exacto

---

Lección Aprendida

A veces, la solución más simple es la mejor. En lugar de intentar capturar y manejar errores de forma compleja, es mejor dejar que las herramientas (npm, Next.js, Docker) muestren sus propios errores de forma natural.

KISS Principle: Keep It Simple, Stupid!

Fecha: 2025-10-11