🔍 Fix v9.3 - Logs Detallados y Variables de Entorno

🐛 Problema Reportado

El build de Next.js estaba fallando sin logs claros:

```
ERROR: failed to solve: process "/bin/sh -c npm run build ..."
did not complete successfully: exit code: 1
```

Sin información sobre **qué** estaba causando el fallo.

🔍 Diagnóstico

El problema tenía dos aspectos:

1. Falta de Logs Detallados

El Dockerfile anterior ejecutaba:

```
RUN npm run build && verificaciones...
```

Si `npm run build` fallaba, Docker mostraba solo el código de salida sin los logs del error.

2. Variables de Entorno Incompletas

Next.js durante el build puede intentar acceder a variables de entorno que se usan en el código. Si están `undefined`, puede causar errores en tiempo de build.

Las variables que faltaban:
- AWS_BUCKET_NAME
- AWS_FOLDER_PREFIX
- AWS_REGION
- OPENPAY_ (5 variables)
- EVOLUTION_API_ (3 variables)

✅ Solución Implementada

Mejora 1: Logs Capturados

Antes (v9.2):

```
RUN npm run build && \
    echo "=== Verificando build standalone ===" && \
    ls -la .next/ && \
    ...
```

Después (v9.3):

```
# Build separado con captura de logs
RUN echo "=== Iniciando build de Next.js ===" && \
    npm run build 2>&1 | tee /tmp/build.log || \
    (cat /tmp/build.log && exit 1)

# Verificación separada
RUN echo "=== Verificando build standalone ===" && \
    ls -la .next/ && \
    if [ ! -d ".next/standalone" ]; then \
        echo "❌ ERROR: standalone output no generado"; \
        echo "=== Contenido de .next: ===" ; \
        find .next -type d -maxdepth 2 ; \
        exit 1; \
    fi && \
    echo "✅ Standalone output verificado"
```

Beneficios:
- `tee /tmp/build.log`: Captura la salida en un archivo
- Si falla, `cat /tmp/build.log`: Muestra los logs completos antes de salir
- Pasos separados: Más fácil identificar dónde falla exactamente

Mejora 2: Variables de Entorno Completas

Agregadas (v9.3):

```
ENV AWS_BUCKET_NAME="placeholder-bucket"
ENV AWS_FOLDER_PREFIX="placeholder/"
ENV AWS_REGION="us-east-1"
ENV OPENPAY_MERCHANT_ID="placeholder"
ENV OPENPAY_PRIVATE_KEY="placeholder"
ENV OPENPAY_PUBLIC_KEY="placeholder"
ENV OPENPAY_BASE_URL="https://sandbox-api.openpay.mx/v1"
ENV EVOLUTION_API_URL="http://localhost:8080"
ENV EVOLUTION_API_TOKEN="placeholder"
ENV EVOLUTION_INSTANCE_NAME="placeholder"
```

Nota importante: Estas son **placeholders para el build**. Las variables reales de runtime se configuran en EasyPanel.

Mejora 3: Logs de Prisma

```
RUN echo "=== Generando Prisma Client ===" && \
    npx prisma generate && \
    echo "✅ Prisma Client generado"
```

Ahora se ve claramente cuándo Prisma termina de generar el cliente.

📊 Impacto

Aspecto	Antes (v9.2)	Después (v9.3)
Logs del build	❌ No visibles	✅ Completos
Diagnóstico de errores	😕 Difícil	✅ Fácil
Variables de entorno	⚠️ Incompletas	✅ Completas
Identificación del problema	⏰ Lento	✅ Inmediato

🔍 Cómo Usar los Nuevos Logs

Si el Build Falla

Los logs ahora mostrarán exactamente dónde falló:

Error en Prisma:

```
=== Generando Prisma Client ===
Error: Cannot find schema.prisma
```

Error en Next.js:

```
=== Iniciando build de Next.js ===
./app/page.tsx
Type error: Cannot find module 'xyz'
```

Error en Standalone:

```
=== Verificando build standalone ===
❌ ERROR: standalone output no generado
=== Contenido de .next: ===
.next/cache
.next/server
```

Interpretar los Logs

1. **Busca "==="**: Marca el inicio de cada etapa
2. **Busca "✅"**: Marca el éxito de cada etapa
3. **Busca "❌"**: Marca fallos
4. **Lee el error específico**: Justo después del fallo

🎯 Variables de Entorno: Build vs Runtime

Variables para Build (en Dockerfile)

Propósito: Permitir que Next.js compile sin errores

Valores: Placeholders (valores dummy)

Por qué: Next.js puede intentar acceder a estas variables durante el build para optimizaciones estáticas

Variables para Runtime (en EasyPanel)

Propósito: Configurar la aplicación en producción

Valores: Reales (tus credenciales y configuraciones)

Por qué: La aplicación las usa cuando está ejecutándose

¿Cuáles son cuáles?

Variable	Build	Runtime	Notas
DATABASE_URL	✅ Placeholder	✅ Real	Prisma necesita para generar cliente
NEXTAUTH_URL	✅ Placeholder	✅ Real	NextAuth valida formato
NEXTAUTH_SECRET	✅ Placeholder	✅ Real	NextAuth valida longitud (>32 chars)
AWS_*	✅ Placeholder	✅ Real	Para código que usa S3
OPENPAY_*	✅ Placeholder	✅ Real	Para código de pagos
EVOLUTION_*	✅ Placeholder	✅ Real	Para código de WhatsApp
NODE_ENV	✅ production	✅ production	Controla optimizaciones
NEXT_OUTPUT_MODE	✅ standalone	-	Solo para build

🔄 Archivos Modificados

1. Dockerfile

Línea 3: Versión actualizada a 9.3

Líneas 46-61: Variables de entorno completas agregadas

Líneas 63-65: Logs de Prisma mejorados

Líneas 67-68: Build de Next.js con captura de logs

Líneas 70-78: Verificación mejorada con diagnóstico

🚀 Próximos Pasos

1. Push a GitHub

```
git add Dockerfile
git commit -m "🔍 Fix v9.3: Logs detallados + variables completas"
git push
```

2. Rebuild en EasyPanel

Ahora cuando el build corra, verás logs completos que te dirán **exactamente** qué está fallando (si es que falla).

3. Analizar los Logs

Si el build falla aún:
1. Copia los logs completos de EasyPanel
2. Busca la sección que dice “=== Iniciando build de Next.js ===”
3. Lee el error específico
4. Ese error nos dirá qué hay que corregir

🐛 Troubleshooting Mejorado

Ahora puedes diagnosticar:

✅ Errores de TypeScript

```
./app/page.tsx
Type error: Property 'x' does not exist
```

✅ Errores de Módulos

```
Module not found: Can't resolve 'package-name'
```

✅ Errores de Prisma

```
Error: Prisma schema not found
```

✅ Errores de Variables

```
Error: NEXTAUTH_SECRET must be at least 32 characters
```

✅ Errores de Configuración

```
Error: Invalid next.config.js
```

💡 Mejores Prácticas

Durante el Desarrollo

Si agregas nuevas variables de entorno al código:

1. **En EasyPanel**: Agrega la variable con su valor real
2. **En Dockerfile**: Agrega placeholder si es necesaria en build time

Durante el Debug

1. **Siempre lee los logs completos**
2. **Busca el primer error**: Los errores subsecuentes suelen ser cascada
3. **Busca las líneas "==="**: Te orientan sobre en qué etapa estás

Para Nuevas Features

Si implementas nuevas funcionalidades que usan variables de entorno:

1. Decide si la variable se necesita en build time
2. Si sí, agrégala al Dockerfile como placeholder
3. Siempre agrégala a EasyPanel con el valor real

📊 Comparación de Debugging

Antes (v9.2)

```
Build falló → "exit code: 1"
¿Por qué? → 🤷 No se sabe
Siguiente paso → 🎲 Adivinar y probar
```

Después (v9.3)

```
Build falló → Logs completos visibles
¿Por qué? → ✅ Error específico mostrado
Siguiente paso → ✅ Fix dirigido
```

✅ Beneficios de v9.3

1. **Transparencia total**: Ves exactamente qué está pasando
2. **Debugging rápido**: No más adivinanzas
3. **Menos iteraciones**: Identificas y corriges el problema en el primer intento
4. **Menos frustración**: Sabes qué está fallando y por qué

🎯 Estado Actual

- [x] Logs mejorados para Prisma
- [x] Logs completos para Next.js build
- [x] Verificación mejorada de standalone
- [x] Variables de entorno completas
- [x] Documentación creada

- [ ] **Push a GitHub** ← Próximo paso
- [ ] Rebuild con logs visibles

🔗 Referencias

- Next.js Build Output (https://nextjs.org/docs/advanced-features/output-file-tracing)
- Docker Multi-Stage Build Logs (https://docs.docker.com/build/building/multi-stage/)
- Environment Variables in Next.js (https://nextjs.org/docs/basic-features/environment-variables)

Versión: 9.3
Mejora: Logs detallados + variables completas
Objetivo: Debugging transparente
Estado: ✅ Implementado
Fecha: 2025-10-15