🔧 Solución al Error de Docker Build - EscalaFin MVP

Fecha: 16 de octubre de 2025
Versión: 1.0
Estado: ✅ Resuelto

📋 Problema Identificado

Error Original

```
ERROR: failed to build: failed to solve: process "/bin/sh -c echo \"=== Instalando dependencias ===\" && ..." did not complete successfully: exit code: 1
```

Causa Raíz

El proceso de instalación de dependencias estaba fallando debido a:

1. **Uso de Yarn en Docker**: El proyecto tiene tanto `yarn.lock` como `package-lock.json`, y el Dockerfile intentaba usar Yarn, lo cual es menos estable en entornos containerizados.
2. **Yarn v4 Lockfile**: El `yarn.lock` usa la versión 8 de metadata, que requiere Yarn v4, agregando complejidad innecesaria al build.
3. **Timeouts y Cache**: Problemas potenciales con timeouts de red y cache corrupto.

✅ Solución Implementada

Cambios en el Dockerfile Principal

Antes:

```
RUN echo "=== Instalando dependencias ===" && \
    if [ -f yarn.lock ]; then \
        echo "Usando Yarn..." && \
        corepack enable && \
        yarn install --frozen-lockfile --network-timeout 300000; \
    elif [ -f package-lock.json ]; then \
        echo "Usando NPM..." && \
        npm ci --legacy-peer-deps; \
    else \
        echo "Usando NPM install..." && \
        npm install --legacy-peer-deps; \
    fi
```

Después:

```
RUN echo "=== Instalando dependencias con NPM ===" && \
    echo "Limpiando cache de npm..." && \
    npm cache clean --force && \
    echo "Instalando dependencias..." && \
    npm install --legacy-peer-deps --prefer-offline --no-audit --progress=false 2>&1
| tee /tmp/install.log && \
    echo "✅ Dependencias instaladas correctamente"
```

Mejoras Aplicadas

1. **NPM Exclusivo**: Se cambió a usar solo NPM, que es más estable y predecible en Docker
2. **Limpieza de Cache**: Se limpia el cache antes de instalar para evitar problemas
3. **Flags Optimizados**:
   - `--legacy-peer-deps`: Resuelve conflictos de dependencias
   - `--prefer-offline`: Reduce dependencia de la red
   - `--no-audit`: Acelera la instalación
   - `--progress=false`: Reduce el output verbose
4. **Logging Mejorado**: Se guarda el log completo en `/tmp/install.log`

📁 Archivos Creados/Modificados

1. `/Dockerfile` (Modificado)

- Actualizado para usar solo NPM
- Mejor manejo de errores
- Logging mejorado

2. `/Dockerfile.simple` (Nuevo)

Versión simplificada y más robusta del Dockerfile:
- Solo NPM, sin lógica condicional
- Estructura más simple y mantenible
- Menos propenso a errores

3. Este documento (`SOLUCION_ERROR_DOCKER_BUILD.md`)

Documentación completa del problema y la solución

🚀 Cómo Usar

Opción 1: Dockerfile Principal (Actualizado)

```
cd /ruta/a/escalafin_mvp
docker build -t escalafin:latest -f Dockerfile .
```

Opción 2: Dockerfile Simplificado (Recomendado)

```
cd /ruta/a/escalafin_mvp
docker build -t escalafin:latest -f Dockerfile.simple .
```

Opción 3: Con Docker Compose

```
cd /ruta/a/escalafin_mvp
docker-compose build
docker-compose up -d
```

🔍 Verificación del Build

Durante el Build

Busca estos mensajes de éxito:

```
=== Instalando dependencias con NPM ===
Limpiando cache de npm...
Instalando dependencias...
✅ Dependencias instaladas correctamente
```

Después del Build

Verifica que la imagen se creó correctamente:

```
docker images | grep escalafin
```

Ejecutar Contenedor de Prueba

```
docker run -p 3000:3000 \
  -e DATABASE_URL="tu-url-de-bd" \
  -e NEXTAUTH_SECRET="tu-secret" \
  escalafin:latest
```

🐛 Troubleshooting

Si Continúa Fallando

1. Verificar Espacio en Disco

```
df -h
```

Asegúrate de tener al menos 5GB libres.

2. Limpiar Recursos de Docker

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

3. Build con Más Memoria

```
docker build --memory=4g --memory-swap=4g -t escalafin:latest .
```

4. Build con Logs Detallados

```
docker build --progress=plain --no-cache -t escalafin:latest . 2>&1 | tee build.log
```

Errores Comunes

Error	Causa	Solución
`ENOENT: no such file`	Symlinks rotos	Verificar que todos los archivos sean reales
`npm ERR! code ERESOLVE`	Conflictos de dependencias	Ya resuelto con `--legacy-peer-deps`
`exit code: 137`	Memoria insuficiente	Aumentar memoria de Docker
`timeout`	Red lenta	Usar `--prefer-offline` (ya incluido)

📊 Comparación de Dockerfiles

Aspecto	Dockerfile Original	Dockerfile (Actualizado)	Dockerfile.simple
Gestores	Yarn y NPM	Solo NPM	Solo NPM
Complejidad	Alta	Media	Baja
Estabilidad	Media	Alta	Muy Alta
Mantenibilidad	Baja	Media	Alta
Recomendado	❌	✅	✅✅

🎯 Recomendaciones

Para Desarrollo Local

- Puedes seguir usando `yarn` o `npm` según prefieras
- Los lockfiles de ambos se mantienen actualizados

Para Docker/Producción

- **Usa** `Dockerfile.simple` para builds más confiables
- **Mantén** `package-lock.json` actualizado
- El `yarn.lock` puede mantenerse pero será ignorado en Docker

Para CI/CD

```
# Ejemplo para GitHub Actions
- name: Build Docker Image
  run: |
    docker build -f Dockerfile.simple -t escalafin:${{ github.sha }} .
```

✅ Checklist de Verificación

Antes de hacer push o deploy:

- [ ] El Dockerfile ha sido actualizado
- [ ] Se ha probado el build localmente
- [ ] La imagen construye sin errores
- [ ] El contenedor arranca correctamente
- [ ] Las migraciones de Prisma funcionan
- [ ] El health check responde
- [ ] Las variables de entorno están configuradas
- [ ] El `.dockerignore` está optimizado

📚 Recursos Adicionales

Documentación Relacionada

- `COOLIFY_DEPLOYMENT_GUIDE.md`: Guía de despliegue con Coolify
- `MULTI_INSTANCE_GUIDE.md`: Despliegue multi-instancia
- `docker-compose.yml`: Configuración de servicios

Comandos Útiles

```
# Ver logs del build
docker build --progress=plain -t escalafin:test . 2>&1 | tee build.log

# Inspeccionar layers
docker history escalafin:latest

# Ejecutar shell en el contenedor
docker run -it --entrypoint sh escalafin:latest

# Ver tamaño de la imagen
docker images escalafin:latest --format "{{.Repository}}:{{.Tag}} - {{.Size}}"
```

🎉 Resultado Esperado

Después de aplicar estos cambios, deberías ver:

```
[+] Building 245.6s (23/23) FINISHED
 => [deps 1/1] RUN echo "=== Instalando dependencias con NPM ===" ...
 => [builder 1/7] COPY --from=deps /app/node_modules ./node_modules
 => [builder 2/7] COPY app/ ./
 => [builder 3/7] RUN npx prisma generate
 => [builder 4/7] RUN sed -i ...
 => [builder 5/7] RUN npm run build
 => [builder 6/7] RUN test -d .next/standalone ...
 => [runner 1/8] RUN addgroup --system ...
 => [runner 2/8] COPY --from=builder /app/.next/standalone ./
 => exporting to image
 => => naming to docker.io/library/escalafin:latest
✅ Build completado exitosamente!
```

📞 Soporte

Si continúas experimentando problemas:

1. Revisa los logs completos del build
2. Verifica que todas las dependencias en `package.json` estén disponibles
3. Comprueba la versión de Docker: `docker --version` (recomendado: 20.10+)
4. Consulta el archivo `build.log` generado durante el build

Última actualización: 16 de octubre de 2025
Mantenedor: EscalaFin DevOps Team
Estado: ✅ Producción Ready