# Solución a Problemas de Memoria y Deployment en Next.js

## 🔴 Problema Identificado

El proyecto INMOVA tiene problemas conocidos de empaquetado para deployment público relacionados con:

1. **Memoria insuficiente durante el build** (>4GB necesarios)
2. **Bundle demasiado grande** (múltiples dependencias pesadas)
3. **Módulos problemáticos** que causan errores de parsing
4. **Chunks muy grandes** que superan los límites

### Dependencias Problemáticas Detectadas:

- 📊 **Charts**: `plotly.js`, `react-plotly.js`, `recharts` (~5MB)
- 🎨 **UI**: 20+ librerías de `@radix-ui` (~3MB)
- 🔐 **Auth y Payments**: `stripe`, `next-auth`, `@prisma/client` (~4MB)
- 📄 **Procesamiento**: `tesseract.js`, `pdf-parse`, `mammoth`, `jspdf` (~8MB)
- 📅 **Utilidades**: `lodash`, `moment`, `date-fns`, `dayjs` (~2MB)
- 🎭 **Animaciones**: `framer-motion` (~1.5MB)

**Total estimado**: ~23MB+ de dependencias

---

## ✅ Solución 1: Optimización del Build (RECOMENDADO)

### Paso 1: Aplicar la Configuración Optimizada

```
cd /home/ubuntu/homming_vidaro/nextjs_space

# 1. Respaldar configuración actual
cp next.config.js next.config.js.backup

# 2. Copiar configuración optimizada
cp next.config.optimized.js next.config.js

# 3. Instalar dependencia necesaria
yarn add null-loader -D
```

### Paso 2: Aumentar Memoria para el Build

Actualiza los scripts en `package.json`:

```
{
  "scripts": {
    "dev": "next dev",
    "build": "NODE_OPTIONS='--max-old-space-size=6144' next build",
    "build:prod": "NODE_OPTIONS='--max-old-space-size=8192' next build",
    "start": "next start",
    "analyze": "ANALYZE=true NODE_OPTIONS='--max-old-space-size=6144' next build"
  }
}
```

### Paso 3: Probar el Build Optimizado

```
# Limpiar cachés anteriores
rm -rf .next
rm -rf node_modules/.cache

# Build con más memoria
NODE_OPTIONS="--max-old-space-size=6144" yarn build
```

### Paso 4: Analizar el Bundle (Opcional)

```
# Instalar analizador
yarn add -D @next/bundle-analyzer

# Ejecutar análisis
ANALYZE=true yarn build
```

Agrega al `next.config.js`:

```
const withBundleAnalyzer = require('@next/bundle-analyzer')({
  enabled: process.env.ANALYZE === 'true',
});

module.exports = withBundleAnalyzer(nextConfig);
```

---

## ✅ Solución 2: Optimizaciones Adicionales del Código

### 2.1. Lazy Loading Agresivo

Ya implementado parcialmente. Verifica que TODOS los charts usen lazy loading:

```
// ✅ CORRECTO
import dynamic from 'next/dynamic';

const RechartsComponents = dynamic(
  () => import('@/components/ui/lazy-charts-extended'),
  { ssr: false, loading: () <LoadingSpinner /> }
);

// ❌ EVITAR importaciones directas
import { BarChart } from 'recharts'; // NO
```

### 2.2. Code Splitting Manual

Para páginas pesadas, usa dynamic imports:

```
// En lugar de:
import HeavyComponent from '@/components/HeavyComponent';

// Usa:
const HeavyComponent = dynamic(
  () => import('@/components/HeavyComponent'),
  { loading: () => <Skeleton /> }
);
```

### 2.3. Eliminar Dependencias No Usadas

Revisa y elimina:

```
# Verificar dependencias no usadas
npx depcheck

# Eliminar duplicados
yarn dedupe
```

### 2.4. Optimizar Importaciones de Lodash

```
// ❌ MAL - importa toda la librería
import _ from 'lodash';

// ✅ BIEN - importación específica
import debounce from 'lodash/debounce';
import throttle from 'lodash/throttle';
```

---

## ✅ Solución 3: Alternativas de Deployment

### Opción A: Vercel (RECOMENDADO para Next.js)

**Ventajas**:
- Optimizado específicamente para Next.js
- Build automático con más memoria (8GB)
- CDN global incluido
- Zero-config deployment
- Mejor para el equipo (más familiar)

**Pasos**:

```
# 1. Instalar Vercel CLI
npm i -g vercel

# 2. Hacer login
vercel login

# 3. Deploy
cd /home/ubuntu/homming_vidaro/nextjs_space
vercel --prod
```

**Configuración de dominio personalizado**:

```
# En Vercel dashboard:
# Settings > Domains > Add Domain > inmova.app
# Agregar registros DNS:
# A    @       76.76.21.21
# CNAME www     cname.vercel-dns.com
```

**Variables de entorno**:
- Todas las del `.env` deben agregarse en Vercel Dashboard > Settings > Environment Variables

---

## Opción B: Railway

**Ventajas**:
- Muy fácil de usar
- Deployment desde GitHub
- Base de datos PostgreSQL incluida
- $5/mes plan Starter

**Pasos**:

1. Crea cuenta en Railway.app (https://railway.app)
2. Conecta tu repo de GitHub
3. Railway detecta Next.js automáticamente
4. Configura variables de entorno
5. Deploy con un click

**Dockerfile optimizado** (crear en raíz del proyecto):

```
FROM node:18-alpine AS base

# Instalar dependencias
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* ./
RUN \
  if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
  elif [ -f package-lock.json ]; then npm ci; \
  elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i --frozen-lockfile; \
  else echo "Lockfile not found." && exit 1; \
  fi

# Builder
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Generar Prisma client
RUN npx prisma generate

# Build con más memoria
ENV NODE_OPTIONS="--max-old-space-size=6144"
RUN yarn build

# Runner
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000

CMD ["node", "server.js"]
```

Agrega a `next.config.js`:

```
output: 'standalone',
```

---

## Opción C: Netlify

**Ventajas**:
- Deploy automático desde Git

- CDN global
- HTTPS automático
- Plan gratuito generoso

**netlify.toml** (crear en raíz):

```
[build]
  command = "NODE_OPTIONS='--max-old-space-size=6144' yarn build"
  publish = ".next"

[build.environment]
  NODE_VERSION = "18"
  NPM_FLAGS = "--legacy-peer-deps"

[[plugins]]
  package = "@netlify/plugin-nextjs"

[[headers]]
  for = "/*"
  [headers.values]
    X-Frame-Options = "SAMEORIGIN"
    X-Content-Type-Options = "nosniff"
```

---

## Opción D: AWS Amplify

**Ventajas**:
- Infraestructura AWS completa
- Escalabilidad automática
- Integración con otros servicios AWS

**amplify.yml** (crear en raíz):

```
version: 1
frontend:
  phases:
    preBuild:
      commands:
        - yarn install
        - npx prisma generate
    build:
      commands:
        - NODE_OPTIONS="--max-old-space-size=6144" yarn build
  artifacts:
    baseDirectory: .next
    files:
      - '**/*'
  cache:
    paths:
      - node_modules/**/*
      - .next/cache/**/*
```

---

## ✅ Solución 4: Optimización de Base de Datos

Si usas Prisma, optimiza:

```
# Generar cliente optimizado
npx prisma generate --no-engine

# En schema.prisma, habilita:
generator client {
  provider = "prisma-client-js"
  binaryTargets = ["native"]
  output = "../node_modules/.prisma/client"
}
```

---

## 📊 Comparación de Alternativas

| Plataforma | Memoria Build | Precio | Dificultad | Velocidad | Recomendación |
|---|---|---|---|---|---|
| **Vercel** | 8GB | $20/mes | ⭐ Muy Fácil | ⚡ Rápido | 🥇 **MEJOR** |
| Railway | 8GB | $5-20/mes | ⭐ Fácil | ⚡ Rápido | 🥈 Buena |
| Netlify | 8GB | Gratis-$19 | ⭐ Fácil | 🟡 Medio | 🥉 Aceptable |
| AWS Amplify | Configurable | Variable | ⭐⭐ Medio | ⚡ Rápido | 💼 Enterprise |
| Build Manual | Depende del servidor | Variable | ⭐⭐⭐ Difícil | 🟡 Lento | ⚠️ No recomendado |

---

## 🎯 Plan de Acción Recomendado

### Para Resolver AHORA (Urgente):

```
# 1. Aplicar configuración optimizada
cd /home/ubuntu/homming_vidaro/nextjs_space
cp next.config.optimized.js next.config.js
yarn add null-loader -D

# 2. Aumentar memoria y probar
NODE_OPTIONS="--max-old-space-size=8192" yarn build

# 3. Si falla, usar Vercel
vercel --prod
```

### Para Mejorar a MEDIANO PLAZO:

1. ✅ Analizar bundle con `@next/bundle-analyzer`
2. ✅ Implementar lazy loading en todos los componentes pesados

3. ✅ Optimizar importaciones (especialmente lodash)
4. ✅ Eliminar dependencias no usadas
5. ✅ Considerar reemplazar plotly.js con recharts
6. ✅ Migrar a Vercel para deployment

### Para Optimizar a LARGO PLAZO:

1. Implementar Server Components de Next.js 14
2. Usar App Router para mejor tree-shaking
3. Implementar ISR (Incremental Static Regeneration)
4. Separar el proyecto en micro-frontends
5. Usar CDN para assets estáticos

---

## 🚨 Troubleshooting

### Error: “JavaScript heap out of memory”

```
# Aumenta la memoria
NODE_OPTIONS="--max-old-space-size=8192" yarn build
```

### Error: “Module parse failed: Unexpected token”

```
# Verifica que next.config.js tiene la regla null-loader
# y que null-loader está instalado
yarn add null-loader -D
```

### Error: “Webpack build exceeded the allowed time”

```
# Habilita paralelización
export NODE_OPTIONS="--max-old-space-size=8192"
export CI=true
yarn build
```

### Bundle demasiado grande

```
# Analiza y optimiza
ANALYZE=true yarn build
# Revisa el informe HTML generado
```

---

## 📞 Soporte

Si los problemas persisten:

1. 📧 Contactar al equipo de Abacus.AI
2. 💬 Abrir issue en el repositorio
3. 📖 Consultar documentación de Next.js: https://nextjs.org/docs/advanced-features/debugging (https://nextjs.org/docs/advanced-features/debugging)

---

**Actualizado**: Diciembre 2025
**Autor**: DeepAgent - Abacus.AI
**Proyecto**: INMOVA