

Análisis del Deployment de CitaPlanner y Recomendaciones para EscalaFin

Fecha: 16 de octubre de 2025






Repositorio Analizado: <https://github.com/qhosting/citaplanner>

Objetivo: Identificar mejores prácticas y aplicarlas a EscalaFin



Resumen Ejecutivo

CitaPlanner implementa un sistema de deployment robusto y profesional usando:

-  **Dockerfile multi-stage** optimizado para producción
 -  **Script start.sh** con verificaciones exhaustivas
 -  **docker-compose.yml** con healthchecks y dependencias claras
 -  **Documentación detallada** de deployment para EasyPanel/Coolify
 -  **Manejo robusto de Prisma** (CLI, cliente, migraciones, seed)
-

Análisis Detallado

1. Dockerfile - Mejores Prácticas Identificadas

 **Estructura Multi-Stage**

```

# Stage 1: deps - Instalar dependencias
FROM node:18-alpine AS deps
RUN apk add --no-cache libc6-compat openssl
WORKDIR /app
COPY package.json package-lock.json ./
RUN npm ci --legacy-peer-deps --ignore-scripts

# Stage 2: builder - Build de la aplicación
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN npx prisma generate
ENV NEXT_TELEMETRY_DISABLED=1
ENV NODE_ENV=production
RUN npm run build

# Stage 3: public-files - Copiar archivos públicos
FROM base AS public-files
WORKDIR /app
COPY public ./public

# Stage 4: runner - Imagen de producción
FROM base AS runner
WORKDIR /app
ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

# Copiar archivos standalone con permisos correctos
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone/app ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone/.next/static ./next/
static
COPY --from=public-files --chown=nextjs:nodejs /app/public ./public

# Copiar Prisma con permisos correctos
COPY --from=builder --chown=nextjs:nodejs /app/prisma ./prisma
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/@prisma ./node_modules/
@prisma
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/.prisma ./
node_modules/.prisma

# Copiar scripts y dependencias necesarias para seed
COPY --from=builder --chown=nextjs:nodejs /app/scripts ./scripts
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/bcryptjs ./node_modules/
bcryptjs
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/tsx ./node_modules/tsx
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/dotenv ./node_modules/
dotenv
COPY --from=builder --chown=nextjs:nodejs /app/node_modules/typescript ./node_modules/
typescript

# Copiar script de inicio
COPY --chown=nextjs:nodejs start.sh ./
RUN chmod +x start.sh

# Crear directorios con permisos correctos
RUN mkdir -p node_modules/.prisma node_modules/@prisma node_modules/.bin \
&& chown -R nextjs:nodejs node_modules/.prisma node_modules/@prisma node_modules/.
bin

```

```
USER nextjs
EXPOSE 3000
ENV PORT=3000
ENV HOSTNAME=0.0.0.0

CMD [ "./start.sh" ]
```

🔑 Puntos Clave del Dockerfile:

1. **Multi-stage para reducir tamaño de imagen final**
 2. **Usuario no-root (nextjs:nodejs)** para seguridad
 3. **Permisos correctos con -chown** en todos los COPY
 4. **Copia selectiva de node_modules** necesarios (no todos)
 5. **Scripts folder incluido** para seed execution
 6. **Dependencias de seed copiadas** (bcryptjs, tsx, dotenv, typescript)
 7. **Verificaciones de Prisma** incluidas en el Dockerfile
-

2. Script start.sh - Inicialización Robusta

Verificaciones y Funcionalidades

```
#!/bin/sh

echo "🚀 Iniciando CITAPLANNER..."

# 1. Configurar PATH para Prisma CLI
export PATH="$PATH:/app/node_modules/.bin"
echo "📦 PATH configurado: $PATH"

# 2. Verificar que Prisma CLI existe (con fallbacks)
echo "🔍 Verificando Prisma CLI..."
if [ -f "node_modules/.bin/prisma" ]; then
    echo "✅ Prisma CLI encontrado en node_modules/.bin/prisma"
    PRISMA_CMD="node_modules/.bin/prisma"
elif [ -f "node_modules/prisma/build/index.js" ]; then
    echo "⚠️ Usando Prisma directamente desde build/index.js"
    PRISMA_CMD="node node_modules/prisma/build/index.js"
else
    echo "❌ Prisma CLI no encontrado - intentando con npx"
    PRISMA_CMD="npx prisma"
fi

echo "🔑 Comando Prisma: $PRISMA_CMD"

# 3. Verificar cliente Prisma
echo "🔍 Verificando cliente Prisma..."
if [ -d "node_modules/@prisma/client" ]; then
    echo "✅ Cliente Prisma encontrado"
else
    echo "⚠️ Cliente Prisma no encontrado, generando..."
    $PRISMA_CMD generate || echo "❌ Error generando cliente Prisma"
fi

# 4. Aplicar migraciones
echo "🔄 Aplicando migraciones si es necesario..."
$PRISMA_CMD migrate deploy || echo "⚠️ Error en migraciones, continuando..."

# 5. Verificar estado de migraciones
echo "📋 Verificando estado de migraciones..."
$PRISMA_CMD migrate status || echo "⚠️ No se pudo verificar estado de migraciones"

# 6. Regenerar cliente Prisma en container
echo "🔑 Regenerando cliente Prisma en container..."
$PRISMA_CMD generate || echo "⚠️ Error generando cliente Prisma"

# 7. Ejecutar seed SOLO si la BD está vacía
echo "🌱 Verificando si necesita seed..."
echo "📋 Consultando tabla users..."

USER_COUNT=$(node -e "
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();
prisma.user.count()
.then(count => { console.log(count); process.exit(0); })
.catch(err => { console.error('0'); process.exit(0); })
.finally(() => prisma.$disconnect());
" 2>/dev/null || echo "0")

echo "👥 Usuarios en la base de datos: $USER_COUNT"

if [ "$USER_COUNT" = "0" ]; then
    echo "🌱 Base de datos vacía - ejecutando seed..."
    if [ -f "scripts/seed.ts" ]; then

```

```

    echo "✅ Seed script encontrado, ejecutando..."
    npm run seed || echo "⚠️ Error ejecutando seed, continuando..."
else
    echo "⚠️ Script seed.ts no encontrado en scripts/"
    echo "📁 Contenido de scripts/:"
    ls -la scripts/ 2>/dev/null || echo "Directorio scripts/ no existe"
fi
else
    echo "✅ Base de datos ya tiene usuarios, omitiendo seed"
fi

# 8. Verificar archivos de Next.js standalone
echo "🔍 Verificando archivos de Next.js standalone..."

if [ ! -f "/app/server.js" ]; then
    echo "❌ ERROR CRITICO: server.js NO ENCONTRADO en /app/server.js"
    echo "📁 Estructura del directorio /app:"
    ls -la /app/ | head -30
    echo ""
    echo "🔍 Buscando server.js en todo el filesystem:"
    find /app -name "server.js" -type f 2>/dev/null | head -10
    echo ""
    echo "❌ El Dockerfile no copió correctamente el standalone build"
    exit 1
fi

echo "✅ server.js encontrado en /app/server.js (CORRECTO)"
echo "📁 Contenido del directorio /app:"
ls -la /app/ | head -20

# 9. Iniciar la aplicación
echo ""
echo "🚀 Iniciando servidor Next.js standalone..."
echo "📁 Working directory: /app"
echo "💻 Server: /app/server.js"
echo "🌐 Hostname: 0.0.0.0"
echo "🔌 Port: 3000"
echo ""

cd /app || {
    echo "❌ ERROR: No se puede cambiar a /app"
    exit 1
}

echo "🎉 EJECUTANDO: node server.js"
exec node server.js

```

🔑 Características Destacadas del start.sh:

1. **Verificación de Prisma CLI con fallbacks** múltiples
2. **Seed condicional** - Solo ejecuta si la BD está vacía (evita duplicados)
3. **Verificación de estructura standalone** antes de iniciar
4. **Manejo de errores robusto** - Continúa aunque haya errores menores
5. **Logs informativos** - Emojis y mensajes claros para debugging
6. **Aplicación de migraciones automática**
7. **Regeneración de cliente Prisma** en el contenedor
8. **Verificación de archivos críticos** antes de iniciar el servidor

3. docker-compose.yml - Configuración Profesional

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
      args:
        - NODE_ENV=production
    ports:
      - "${APP_PORT:-3000}:3000"
    environment:
      - DATABASE_URL=${DATABASE_URL}
      - NEXTAUTH_URL=${NEXTAUTH_URL}
      - NEXTAUTH_SECRET=${NEXTAUTH_SECRET}
      - NODE_ENV=production
      - PORT=3000
    depends_on:
      postgres:
        condition: service_healthy # ★ Espera hasta que postgres esté healthy
    restart: unless-stopped
    networks:
      - app-network
    volumes:
      - app-data:/app/.next/cache # ★ Caché de Next.js persistente

  postgres:
    image: postgres:17-alpine
    environment:
      - POSTGRES_DB=${POSTGRES_DB:-citaplanner_db}
      - POSTGRES_USER=${POSTGRES_USER:-postgres}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    ports:
      - "${DB_PORT:-5432}:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - postgres_backups:/backup-citaplanner # ★ Volumen para backups
      - ./init-db.sql:/docker-entrypoint-initdb.d/init-db.sql
    restart: unless-stopped
    networks:
      - app-network
    healthcheck: # ★ Healthcheck configurado
      test: ["CMD-SHELL", "pg_isready -U ${POSTGRES_USER:-postgres}"]
      interval: 10s
      timeout: 5s
      retries: 5

volumes:
  postgres_data:
  postgres_backups:
  app-data:

networks:
  app-network:
    driver: bridge
```

🔑 Mejores Prácticas del docker-compose.yml:

1. **Healthcheck en PostgreSQL** - Asegura que la BD esté lista antes de iniciar la app

2. **depends_on con condition: service_healthy** - App espera hasta que postgres esté funcionando
3. **Volumen para caché de Next.js** - Mejora rendimiento en rebuilds
4. **Volumen para backups** - Facilita respaldos de la BD
5. **Script de inicialización** - `init-db.sql` se ejecuta al crear la BD
6. **Red aislada** - app-network para comunicación entre servicios
7. **restart: unless-stopped** - Reinicia automáticamente en caso de fallo
8. **Variables con valores por defecto** - `${APP_PORT:-3000}` facilita configuración

Recomendaciones para EscalaFin

✓ Prioridad ALTA - Implementar Inmediatamente

1. Mejorar el script start.sh

Problema Actual: EscalaFin probablemente no tiene un `start.sh` tan robusto

Solución: Crear/actualizar `start.sh` con:

- ✓ Verificaciones de Prisma CLI
- ✓ Seed condicional (solo si BD vacía)
- ✓ Aplicación automática de migraciones
- ✓ Verificaciones de estructura standalone
- ✓ Logs informativos con emojis

Archivo a crear: `/home/ubuntu/escalafin_mvp/start.sh`

2. Optimizar el Dockerfile

Problema Actual: Posibles permisos incorrectos o falta de dependencias para seed

Solución: Actualizar Dockerfile para incluir:

- ✓ Usuario no-root (nextjs:nodejs) con UID 1001
- ✓ Permisos correctos (`-chown`) en todos los COPY
- ✓ Copiar scripts folder para seed
- ✓ Copiar dependencias necesarias (bcryptjs, tsx, dotenv, typescript)
- ✓ Crear directorios de Prisma con permisos correctos

3. Mejorar docker-compose.yml

Problema Actual: Posiblemente faltan healthchecks y volúmenes optimizados

Solución: Actualizar docker-compose.yml con:



- ✓ Healthcheck en PostgreSQL
- ✓ `depends_on` con `condition: service_healthy`
- ✓ Volumen para caché de Next.js
- ✓ Volumen para backups de PostgreSQL
- ✓ Variables de entorno con valores por defecto

4. Agregar Verificaciones de Inicialización

Problema Actual: No hay logs claros cuando algo falla durante el inicio

Solución: Implementar:

- ✓ Logs con emojis para fácil identificación
- ✓ Verificaciones paso a paso (PATH, Prisma CLI, BD, server.js)









-  Diagnósticos automáticos si algo falla
-  Continuación con warnings en errores no críticos

Prioridad MEDIA - Implementar Pronto

5. Documentación de Deployment


Crear: `COOLIFY_DEPLOYMENT_GUIDE.md` similar al de CitaPlanner pero adaptado para Coolify

Contenido:

-  Requisitos previos
-  Paso 1: Crear proyecto en Coolify
-  Paso 2: Configurar PostgreSQL
-  Paso 3: Desplegar la aplicación
-  Paso 4: Variables de entorno requeridas
-  Paso 5: Configurar dominios y SSL
-  Paso 6: Monitoreo y logs
-  Solución de problemas comunes

6. Script de Backup Automático

Crear: Script para backups automáticos de PostgreSQL

```
#!/bin/bash
# backup-db.sh
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
docker compose exec -T postgres pg_dump -U postgres escalafin_db > "/backup-escalafin/
backup_${TIMESTAMP}.sql"
echo " Backup creado: backup_${TIMESTAMP}.sql"
```

7. Health Check Endpoint

Crear: `/app/app/api/health/route.ts` para monitoreo

```
import { NextResponse } from 'next/server';
import { db } from '@lib/db';

export async function GET() {
  try {
    // Verificar conexión a BD
    await db.$queryRaw`SELECT 1`;

    return NextResponse.json({
      status: 'healthy',
      timestamp: new Date().toISOString(),
      database: 'connected'
    });
  } catch (error) {
    return NextResponse.json({
      status: 'unhealthy',
      timestamp: new Date().toISOString(),
      error: 'Database connection failed'
    }, { status: 503 });
  }
}
```

✓ **Prioridad BAJA - Mejoras Futuras**

8. Monitoring y Observability

- Implementar logs estructurados (JSON)
- Agregar métricas de rendimiento
- Configurar alertas para errores críticos

9. CI/CD Pipeline

- Crear GitHub Actions workflow para testing
- Implementar deployment automático a staging
- Configurar rollback automático en caso de fallo

10. Escalabilidad

- Configurar load balancing
 - Implementar caché distribuido (Redis)
 - Optimizar para múltiples instancias
-



Comparación: CitaPlanner vs EscalaFin

Característica	CitaPlanner	EscalaFin (Actual)	Prioridad
Dockerfile Multi-Stage	✓ Optimizado	⚠ Mejorable	ALTA
Script start.sh Robusto	✓ Completo	⚠ Básico/Inexistente	ALTA
Healthchecks	✓ Configurados	✗ Faltantes	ALTA
Seed Condicional	✓ Solo si BD vacía	✗ Siempre ejecuta	ALTA
Permisos Correctos	✓ nextjs:nodejs	⚠ Posiblemente root	ALTA
Verificaciones de Prisma	✓ Múltiples fallbacks	✗ Básicas	MEDIA
Volúmenes Optimizados	✓ Caché + Backups	⚠ Solo datos	MEDIA
Documentación Deployment	✓ Detallada	⚠ Básica	MEDIA
Health Endpoint	✓ Implementado	✗ Faltante	MEDIA
Logs Informativos	✓ Con emojis	⚠ Básicos	BAJA
Backups Automáticos	✓ Configurados	✗ Manuales	BAJA
CI/CD	✗ No tiene	✗ Básico	BAJA



Plan de Acción Sugerido

Fase 1: Correcciones Críticas (1-2 horas)

1. **Crear script start.sh robusto** con todas las verificaciones
2. **Actualizar Dockerfile** para incluir permisos correctos y dependencias
3. **Mejorar docker-compose.yml** con healthchecks y volúmenes

Fase 2: Mejoras Importantes (2-3 horas)

1. **Agregar health endpoint** para monitoreo
2. **Crear documentación de deployment** para Coolify
3. **Implementar seed condicional** para evitar duplicados

Fase 3: Optimizaciones (1-2 horas)

1. **Agregar script de backup** automático
2. **Mejorar logs** con emojis y estructura clara
3. **Documentar proceso de rollback**

Fase 4: Pruebas y Validación (1 hora)

1. **Probar deployment completo** desde cero
2. **Verificar todos los escenarios** (BD vacía, BD existente, errores)
3. **Validar rollback** funciona correctamente

Tiempo Total Estimado: 6-8 horas



Conclusiones

CitaPlanner implementa un sistema de deployment **profesional y robusto** que sigue las mejores prácticas de la industria:








Fortalezas de CitaPlanner:

1. **Dockerfile multi-stage optimizado** - Reduce tamaño de imagen
2. **Script start.sh con verificaciones exhaustivas** - Previene errores
3. **Seed condicional** - Evita duplicados y corruption de datos
4. **Healthchecks configurados** - Asegura disponibilidad
5. **Documentación detallada** - Facilita deployment por otros desarrolladores
6. **Manejo robusto de errores** - Continúa operación cuando es posible



Aplicabilidad a EscalaFin:

TODAS estas prácticas son **directamente aplicables** a EscalaFin y deberían implementarse para garantizar:

-  **Deployments más confiables** - Menos errores en producción
-  **Mejor experiencia de desarrollo** - Logs claros y debugging fácil
-  **Escalabilidad mejorada** - Preparado para crecimiento
-  **Mantenimiento simplificado** - Documentación y scripts claros
-  **Seguridad mejorada** - Usuario no-root, permisos correctos









Referencias

- **Repositorio CitaPlanner:** <https://github.com/qhosting/citaplanner>
- **Next.js Standalone Output:** <https://nextjs.org/docs/advanced-features/output-file-tracing>
- **Docker Multi-Stage Builds:** <https://docs.docker.com/build/building/multi-stage/>
- **PostgreSQL Healthchecks:** <https://docs.docker.com/compose/compose-file/compose-file-v3/#healthcheck>
- **Prisma Best Practices:** <https://www.prisma.io/docs/guides/performance-and-optimization>

Próximos Pasos

¿Te gustaría que implemente alguna de estas mejoras en EscalaFin? Puedo empezar por:

1.  Crear el script `start.sh` robusto
2.  Actualizar el `Dockerfile` con las mejores prácticas
3.  Mejorar el `docker-compose.yml` con healthchecks
4.  Crear la documentación de deployment para Coolify
5.  Implementar el health endpoint

Solo dime por dónde quieres que empiece. 

Documento generado el: 16 de octubre de 2025

Autor: DeepAgent - Análisis de Mejores Prácticas

Versión: 1.0