🚀 MEJORAS DE BUILD & DEPLOY - INMOVA

Fecha: 7 de Diciembre, 2025
Auditor: Arquitecto de Software Senior & DevOps Engineer
Aplicación: INMOVA - Sistema de Gestión de Propiedades
Infraestructura: Abacus.AI Hosted + AWS (Postgres, S3)

📋 RESUMEN EJECUTIVO

✅ Configuración Actual

Plataforma de Deploy: Abacus.AI Hosted
Dominio: `inmova.app`
Build System: Next.js 15 Standalone Output
Database: PostgreSQL (Hosted by Abacus.AI)
File Storage: AWS S3
Node Version: 20.x

🚨 PROBLEMAS IDENTIFICADOS

1. ❌ **Build Commands Manuales**: Comandos de deploy complejos y frágiles
2. ❌ **No hay CI/CD**: Sin pipeline automatizado
3. ❌ **Falta Health Checks**: Sin verificación post-deploy
4. ❌ **Environment Variables Inseguras**: `.env` contiene secrets en repo
5. ❌ **No hay Rollback Strategy**: Sin plan de reversión
6. ❌ **Build Time Largo**: ~3-5 minutos
7. ⚠️ **Docker Config Básica**: Dockerfile sin optimizaciones
8. ⚠️ **No hay Staging Environment**: Deploy directo a producción

---

🔴 1. CONFIGURACIÓN DE BUILD MEJORADA

1.1 Optimizar next.config.js

Archivo actual:

```
// next.config.js (líneas 4-17)
const nextConfig = {
  distDir: process.env.NEXT_DIST_DIR || '.next',
  output: process.env.NEXT_OUTPUT_MODE,
  experimental: {
    outputFileTracingRoot: path.join(__dirname, '../'),
  },
  eslint: {
    ignoreDuringBuilds: true, // ❌ PROBLEMA
  },
  typescript: {
    ignoreBuildErrors: false,
  },
  images: { unoptimized: true }, // ❌ PROBLEMA
};
```

✅ **Configuración Mejorada:**

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

/** @type {import('next').NextConfig} */
const nextConfig = {
  // Build output
  distDir: process.env.NEXT_DIST_DIR || '.build',
  output: process.env.NEXT_OUTPUT_MODE || 'standalone',

  // File tracing
  experimental: {
    outputFileTracingRoot: path.join(__dirname, '../'),
    // ✅ Optimizaciones experimentales
    optimizeCss: true,
    optimizePackageImports: ['lucide-react', 'lodash'],
  },

  // ✅ Code quality
  eslint: {
    ignoreDuringBuilds: false, // Forzar ESLint
    dirs: ['app', 'components', 'lib', 'hooks'],
  },
  typescript: {
    ignoreBuildErrors: false,
    tsconfigPath: './tsconfig.build.json',
  },

  // ✅ Optimización de imágenes
  images: {
    unoptimized: false,
    formats: ['image/avif', 'image/webp'],
    deviceSizes: [640, 750, 828, 1080, 1200, 1920, 2048],
    imageSizes: [16, 32, 48, 64, 96, 128, 256, 384],
    minimumCacheTTL: 60 * 60 * 24 * 30, // 30 días
    remotePatterns: [
      {
        protocol: 'https',
        hostname: process.env.AWS_BUCKET_NAME + '.s3.' + process.env.AWS_REGION + '.am
azonaws.com',
      },
    ],
  },

  // ✅ Webpack optimizations
  webpack: (config, { dev, isServer }) => {
    // Producción optimizations
    if (!dev) {
      config.optimization = {
        ...config.optimization,
        minimize: true,
        splitChunks: {
          chunks: 'all',
          cacheGroups: {
            default: false,
            vendors: false,
            // Vendor chunk
            vendor: {
              name: 'vendor',
              chunks: 'all',
              test: /node_modules/,
              priority: 20,
            },
            // Common chunk
            common: {
              name: 'common',
              minChunks: 2,
              chunks: 'all',
              priority: 10,
              reuseExistingChunk: true,
              enforce: true,
            },
          },
        },
      };
    }

    // Aliases
    config.resolve.alias = {
      ...config.resolve.alias,
      '@': path.resolve(__dirname),
    };

    return config;
  },

  // ✅ Compiler optimizations
  compiler: {
    removeConsole: process.env.NODE_ENV === 'production' ? {
      exclude: ['error', 'warn'],
    } : false,
  },

  // ✅ Headers de seguridad
  async headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'X-DNS-Prefetch-Control',
            value: 'on',
          },
          {
            key: 'Strict-Transport-Security',
            value: 'max-age=31536000; includeSubDomains; preload',
          },
          {
            key: 'X-Content-Type-Options',
            value: 'nosniff',
          },
          {
            key: 'X-Frame-Options',
            value: 'DENY',
          },
          {
            key: 'X-XSS-Protection',
            value: '1; mode=block',
          },
          {
            key: 'Referrer-Policy',
            value: 'strict-origin-when-cross-origin',
          },
        ],
      },
    ];
  },

  // ✅ Redirects
  async redirects() {
    return [
      {
        source: '/admin',
        destination: '/admin/users',
        permanent: false,
      },
    ];
  },

  // ✅ Rewrites para APIs externas
  async rewrites() {
    return [
      {
        source: '/api/stripe/:path*',
        destination: 'https://api.stripe.com/:path*',
      },
    ];
  },
};

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

1.2 Optimizar Dockerfile

Dockerfile actual (líneas 1-55):
- ✅ Multi-stage build (bueno)
- ❌ No usa build cache eficientemente
- ❌ No separa dependencias dev/prod
- ❌ No usa .dockerignore optimizado

✅ **Dockerfile Mejorado:**

```dockerfile
# syntax=docker/dockerfile:1.4

# ====================
# STAGE 1: Dependencies
# ====================
FROM node:20-alpine AS deps
RUN apk add --no-cache libc6-compat openssl
WORKDIR /app

# Copy package files
COPY package.json yarn.lock* .yarnrc.yml ./
COPY .yarn ./.yarn

# Install dependencies with cache mount
RUN --mount=type=cache,target=/root/.yarn \
    yarn install --frozen-lockfile --production=false

# ====================
# STAGE 2: Builder
# ====================
FROM node:20-alpine AS builder
WORKDIR /app

# Copy dependencies from deps stage
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Environment variables para build
ENV NEXT_TELEMETRY_DISABLED=1
ENV NODE_ENV=production
ENV SKIP_ENV_VALIDATION=1

# Generate Prisma Client
RUN yarn prisma generate

# Build application
RUN --mount=type=cache,target=/app/.next/cache \
    NODE_OPTIONS="--max-old-space-size=4096" yarn build

# ====================
# STAGE 3: Runner (Production)
# ====================
FROM node:20-alpine AS runner
WORKDIR /app

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1
ENV PORT=3000
ENV HOSTNAME="0.0.0.0"

# Security: Create non-root user
RUN addgroup --system --gid 1001 nodejs && \
    adduser --system --uid 1001 nextjs && \
    apk add --no-cache tini curl

# Copy only necessary files from builder
COPY --from=builder --chown=nextjs:nodejs /app/.build/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.build/static ./.build/static
COPY --from=builder --chown=nextjs:nodejs /app/public ./public
COPY --from=builder --chown=nextjs:nodejs /app/prisma ./prisma

# Create directories
```

```
RUN mkdir -p /app/uploads /app/logs && \
    chown -R nextjs:nodejs /app/uploads /app/logs

USER nextjs

EXPOSE 3000

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
  CMD curl -f http://localhost:3000/api/health || exit 1

# Use tini for proper signal handling
ENTRYPOINT ["/sbin/tini", "--"]
CMD ["node", "server.js"]
```

✅ **.dockerignore Mejorado:**

```
# .dockerignore
.git
.gitignore
.next
.build
node_modules

# Testing
__tests__
*.test.ts
*.test.tsx
*.spec.ts
*.spec.tsx
coverage

# Documentation
*.md
!README.md
docs

# Build artifacts
dist
build
out

# Environment
.env.local
.env.*.local
.env.development
.env.test

# IDE
.vscode
.idea
*.swp
*.swo
*~

# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# OS
.DS_Store
Thumbs.db

# Temporary
tmp
temp
*.tmp
```

1.3 Scripts de Build Optimizados

package.json - Scripts mejorados:

```json
{
  "scripts": {
    "dev": "next dev",
    "build": "yarn prisma:generate && next build",
    "build:analyze": "ANALYZE=true yarn build",
    "build:production": "NODE_ENV=production NODE_OPTIONS='--max-old-space-size=4096'
yarn build",
    "start": "next start",
    "start:production": "NODE_ENV=production node .build/standalone/server.js",
    "lint": "next lint",
    "lint:fix": "next lint --fix",
    "type-check": "tsc --noEmit",
    "format": "prettier --write \"**/*.{js,jsx,ts,tsx,json,md}\"",
    "format:check": "prettier --check \"**/*.{js,jsx,ts,tsx,json,md}\"",

    "prisma:generate": "prisma generate",
    "prisma:migrate": "prisma migrate dev",
    "prisma:migrate:prod": "prisma migrate deploy",
    "prisma:seed": "tsx --require dotenv/config scripts/seed.ts",
    "prisma:studio": "prisma studio",
    "prisma:reset": "prisma migrate reset --force",

    "test": "vitest",
    "test:ci": "vitest run --coverage",
    "test:e2e": "playwright test",
    "test:e2e:ui": "playwright test --ui",

    "docker:build": "docker build -t inmova:latest .",
    "docker:run": "docker run -p 3000:3000 --env-file .env inmova:latest",
    "docker:compose": "docker-compose up -d",
    "docker:compose:down": "docker-compose down",

    "deploy:staging": "./scripts/deploy-staging.sh",
    "deploy:production": "./scripts/deploy-production.sh",
    "deploy:rollback": "./scripts/rollback.sh",

    "db:backup": "./scripts/backup-database.sh",
    "db:restore": "./scripts/restore-database.sh",

    "pre-commit": "yarn lint && yarn type-check && yarn test:ci",
    "pre-push": "yarn build"
  }
}
```

🔴 2. CI/CD PIPELINE

2.1 GitHub Actions Workflow

Crear: `.github/workflows/ci-cd.yml`

```yaml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main, develop]

env:
  NODE_VERSION: '20.x'
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  # ====================
  # JOB 1: Code Quality
  # ====================
  quality:
    name: Code Quality Checks
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'yarn'

      - name: Install dependencies
        run: yarn install --frozen-lockfile

      - name: Lint
        run: yarn lint

      - name: Type check
        run: yarn type-check

      - name: Format check
        run: yarn format:check

      - name: Security audit
        run: yarn audit --level high
        continue-on-error: true

  # ====================
  # JOB 2: Tests
  # ====================
  test:
    name: Run Tests
    runs-on: ubuntu-latest
    needs: quality
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'yarn'
      - name: Install dependencies
        run: yarn install --frozen-lockfile

      - name: Run unit tests
        run: yarn test:ci

      - name: Upload coverage
        uses: codecov/codecov-action@v3
        with:
          files: ./coverage/coverage-final.json
          fail_ci_if_error: false

      - name: Install Playwright
        run: npx playwright install --with-deps

      - name: Run E2E tests
        run: yarn test:e2e
        env:
          DATABASE_URL: ${{ secrets.TEST_DATABASE_URL }}

  # ====================
  # JOB 3: Build
  # ====================
  build:
    name: Build Application
    runs-on: ubuntu-latest
    needs: [quality, test]
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: ${{ env.NODE_VERSION }}
          cache: 'yarn'

      - name: Install dependencies
        run: yarn install --frozen-lockfile

      - name: Generate Prisma Client
        run: yarn prisma:generate

      - name: Build application
        run: yarn build:production
        env:
          SKIP_ENV_VALIDATION: true
          DATABASE_URL: ${{ secrets.DATABASE_URL }}

      - name: Upload build artifacts
        uses: actions/upload-artifact@v3
        with:
          name: build-artifacts
          path: |
            .build
            public
            package.json
            yarn.lock
          retention-days: 7

  # ====================
  # JOB 4: Docker Build
  # ====================
  docker:
    name: Build Docker Image
    runs-on: ubuntu-latest
    needs: build
    if: github.event_name == 'push' && (github.ref == 'refs/heads/main' || github.ref
== 'refs/heads/develop')
    permissions:
      contents: read
      packages: write
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Log in to Container Registry
        uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=ref,event=branch
            type=sha,prefix={{branch}}-
            type=semver,pattern={{version}}

      - name: Build and push Docker image
        uses: docker/build-push-action@v5
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

  # ====================
  # JOB 5: Deploy Staging
  # ====================
  deploy-staging:
    name: Deploy to Staging
    runs-on: ubuntu-latest
    needs: docker
    if: github.event_name == 'push' && github.ref == 'refs/heads/develop'
    environment:
      name: staging
      url: https://staging.inmova.app
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Deploy to Staging
        run: |
          echo "Deploying to staging..."
          # Tu comando de deploy aquí
          # Ejemplo: curl -X POST https://api.abacus.ai/deploy ...

      - name: Run smoke tests
        run: |
          echo "Running smoke tests..."
          curl -f https://staging.inmova.app/api/health || exit 1

  # ====================
  # JOB 6: Deploy Production
  # ====================
  deploy-production:
    name: Deploy to Production
    runs-on: ubuntu-latest
    needs: docker
    if: github.event_name == 'push' && github.ref == 'refs/heads/main'
    environment:
      name: production
      url: https://inmova.app
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Database Migration
        run: |
          echo "Running database migrations..."
          yarn prisma:migrate:prod
        env:
          DATABASE_URL: ${{ secrets.PROD_DATABASE_URL }}

      - name: Deploy to Production
        run: |
          echo "Deploying to production..."
          # Tu comando de deploy aquí

      - name: Health check
        run: |
          echo "Verifying deployment..."
          for i in {1..10}; do
            if curl -f https://inmova.app/api/health; then
              echo "✅ Deployment successful!"
              exit 0
            fi
            echo "⏳ Waiting for deployment... ($i/10)"
            sleep 30
          done
          echo "❌ Deployment failed!"
          exit 1

      - name: Notify deployment
        if: success()
        run: |
          echo "🚀 Deployment to production successful!"
          # Enviar notificación (Slack, Discord, email, etc.)

      - name: Rollback on failure
        if: failure()
        run: |
          echo "⚠️ Deployment failed, rolling back..."
          yarn deploy:rollback
```

2.2 Scripts de Deploy

Crear: `scripts/deploy-production.sh`

```bash
#!/bin/bash

# ============================================
# INMOVA - Production Deployment Script
# ============================================

set -e # Exit on error

# Colors
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

echo -e "${GREEN}🚀 INMOVA Production Deployment${NC}"
echo "============================================"

# 1. Pre-deployment checks
echo -e "\n${YELLOW}📋 Pre-deployment checks...${NC}"

# Check if we're on main branch
CURRENT_BRANCH=$(git rev-parse --abbrev-ref HEAD)
if [ "$CURRENT_BRANCH" != "main" ]; then
  echo -e "${RED}❌ Error: Not on main branch (current: $CURRENT_BRANCH)${NC}"
  exit 1
fi

# Check for uncommitted changes
if [ -n "$(git status --porcelain)" ]; then
  echo -e "${RED}❌ Error: Uncommitted changes detected${NC}"
  git status --short
  exit 1
fi

# Check if .env.production exists
if [ ! -f ".env.production" ]; then
  echo -e "${RED}❌ Error: .env.production not found${NC}"
  exit 1
fi

echo -e "${GREEN}✅ Pre-deployment checks passed${NC}"

# 2. Backup database
echo -e "\n${YELLOW}💾 Backing up database...${NC}"
./scripts/backup-database.sh production

if [ $? -ne 0 ]; then
  echo -e "${RED}❌ Database backup failed${NC}"
  exit 1
fi

echo -e "${GREEN}✅ Database backup completed${NC}"

# 3. Run tests
echo -e "\n${YELLOW}🧪 Running tests...${NC}"
yarn test:ci

if [ $? -ne 0 ]; then
  echo -e "${RED}❌ Tests failed${NC}"
  exit 1
fi
```

```
echo -e "${GREEN}✅ Tests passed${NC}"

# 4. Build application
echo -e "\n${YELLOW}🏗️  Building application...${NC}"
cp .env.production .env
yarn build:production

if [ $? -ne 0 ]; then
  echo -e "${RED}❌ Build failed${NC}"
  exit 1
fi

echo -e "${GREEN}✅ Build completed${NC}"

# 5. Database migrations
echo -e "\n${YELLOW}🗄️  Running database migrations...${NC}"
yarn prisma:migrate:prod

if [ $? -ne 0 ]; then
  echo -e "${RED}❌ Migrations failed${NC}"
  echo -e "${YELLOW}⚠️  Rolling back database...${NC}"
  ./scripts/restore-database.sh production latest
  exit 1
fi

echo -e "${GREEN}✅ Migrations completed${NC}"

# 6. Deploy to Abacus.AI
echo -e "\n${YELLOW}🚀 Deploying to production...${NC}"

# Tag current deployment
DEPLOY_TAG="deploy-$(date +%Y%m%d-%H%M%S)"
git tag -a "$DEPLOY_TAG" -m "Production deployment $DEPLOY_TAG"
git push origin "$DEPLOY_TAG"

# Deploy command (ajustar según tu setup)
# Ejemplo con Abacus.AI CLI:
# abacus-cli deploy --project inmova --environment production

echo -e "${GREEN}✅ Deployment initiated${NC}"

# 7. Health checks
echo -e "\n${YELLOW}🏥 Running health checks...${NC}"
MAX_RETRIES=10
RETRY_DELAY=30
HEALTH_URL="https://inmova.app/api/health"

for i in $(seq 1 $MAX_RETRIES); do
  echo -e "Attempt $i/$MAX_RETRIES..."

  RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$HEALTH_URL" || echo "000")

  if [ "$RESPONSE" = "200" ]; then
    echo -e "${GREEN}✅ Health check passed!${NC}"
    break
  fi

  if [ $i -eq $MAX_RETRIES ]; then
    echo -e "${RED}❌ Health check failed after $MAX_RETRIES attempts${NC}"
    echo -e "${YELLOW}⚠️  Initiating rollback...${NC}"
    ./scripts/rollback.sh "$DEPLOY_TAG"
    exit 1
  fi
  echo -e "${YELLOW}⏳ Waiting ${RETRY_DELAY}s before retry...${NC}"
  sleep $RETRY_DELAY
done

# 8. Smoke tests
echo -e "\n${YELLOW}🔥 Running smoke tests...${NC}"

# Test critical endpoints
TEST_ENDPOINTS=(
  "$HEALTH_URL"
  "https://inmova.app/api/auth/signin"
  "https://inmova.app/dashboard"
)

for endpoint in "${TEST_ENDPOINTS[@]}"; do
  RESPONSE=$(curl -s -o /dev/null -w "%{http_code}" "$endpoint" || echo "000")
  if [ "$RESPONSE" != "200" ] && [ "$RESPONSE" != "401" ]; then
    echo -e "${RED}❌ Smoke test failed for $endpoint (HTTP $RESPONSE)${NC}"
    exit 1
  fi
  echo -e "${GREEN}✅ $endpoint${NC}"
done

echo -e "${GREEN}✅ Smoke tests passed${NC}"

# 9. Cleanup old backups (keep last 10)
echo -e "\n${YELLOW}🧹 Cleaning up old backups...${NC}"
ls -t backups/production-*.sql 2>/dev/null | tail -n +11 | xargs -r rm

# 10. Send notification
echo -e "\n${YELLOW}📧 Sending notifications...${NC}"
# Ejemplo: curl -X POST https://hooks.slack.com/services/YOUR/WEBHOOK/URL \
#   -d '{"text": "✅ INMOVA deployed to production successfully!"}'

echo -e "\n${GREEN}============================================${NC}"
echo -e "${GREEN}🎉 Deployment completed successfully!${NC}"
echo -e "${GREEN}============================================${NC}"
echo -e "\nDeployment tag: ${YELLOW}$DEPLOY_TAG${NC}"
echo -e "URL: ${YELLOW}https://inmova.app${NC}"
echo -e "\nNext steps:"
echo -e "  1. Monitor logs: ${YELLOW}yarn logs:production${NC}"
echo -e "  2. Check metrics: ${YELLOW}https://inmova.app/admin/metrics${NC}"
echo -e "  3. Verify user feedback"

exit 0
```

Dar permisos de ejecución:

```
chmod +x scripts/deploy-production.sh
chmod +x scripts/deploy-staging.sh
chmod +x scripts/rollback.sh
chmod +x scripts/backup-database.sh
```

🔴 3. HEALTH CHECKS & MONITORING

3.1 Health Check Endpoint

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

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

export const dynamic = 'force-dynamic';

interface HealthStatus {
  status: 'healthy' | 'degraded' | 'unhealthy';
  timestamp: string;
  uptime: number;
  version: string;
  checks: {
    database: {
      status: 'up' | 'down';
      latency?: number;
      error?: string;
    };
    redis: {
      status: 'up' | 'down';
      latency?: number;
      error?: string;
    };
    storage: {
      status: 'up' | 'down';
      error?: string;
    };
  };
}

export async function GET() {
  const startTime = Date.now();
  const health: HealthStatus = {
    status: 'healthy',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    version: process.env.npm_package_version || '1.0.0',
    checks: {
      database: { status: 'down' },
      redis: { status: 'down' },
      storage: { status: 'down' },
    },
  };

  // Check database
  try {
    const dbStart = Date.now();
    await prisma.$queryRaw`SELECT 1`;
    health.checks.database = {
      status: 'up',
      latency: Date.now() - dbStart,
    };
  } catch (error: any) {
    health.status = 'unhealthy';
    health.checks.database = {
      status: 'down',
      error: error.message,
    };
  }

  // Check Redis
  try {
    const redis = await import('@/lib/rate-limit-enhanced').then(m => m.redis);
    if (redis) {
      const redisStart = Date.now();
      await redis.ping();
      health.checks.redis = {
        status: 'up',
        latency: Date.now() - redisStart,
      };
    } else {
      health.checks.redis = { status: 'down', error: 'Not configured' };
    }
  } catch (error: any) {
    health.status = 'degraded';
    health.checks.redis = {
      status: 'down',
      error: error.message,
    };
  }

  // Check S3 storage
  try {
    const { S3Client, HeadBucketCommand } = await import('@aws-sdk/client-s3');
    const { getBucketConfig } = await import('@/lib/aws-config');

    const config = getBucketConfig();
    const client = new S3Client({});

    await client.send(new HeadBucketCommand({ Bucket: config.bucketName }));
    health.checks.storage = { status: 'up' };
  } catch (error: any) {
    health.status = 'degraded';
    health.checks.storage = {
      status: 'down',
      error: error.message,
    };
  }

  // Determine HTTP status code
  const statusCode = health.status === 'healthy' ? 200 : health.status ===
'degraded' ? 200 : 503;

  return NextResponse.json(health, {
    status: statusCode,
    headers: {
      'Cache-Control': 'no-cache, no-store, must-revalidate',
    },
  });
}
```

3.2 Monitoring Dashboard

Crear: `app/admin/monitoring/page.tsx`

```
'use client';

import { useEffect, useState } from 'react';
import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';
import { Badge } from '@/components/ui/badge';

interface HealthData {
  status: 'healthy' | 'degraded' | 'unhealthy';
  timestamp: string;
  uptime: number;
  version: string;
  checks: {
    database: { status: string; latency?: number; error?: string };
    redis: { status: string; latency?: number; error?: string };
    storage: { status: string; error?: string };
  };
}

export default function MonitoringPage() {
  const [health, setHealth] = useState<HealthData | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchHealth = async () => {
      try {
        const res = await fetch('/api/health');
        const data = await res.json();
        setHealth(data);
      } catch (error) {
        console.error('Failed to fetch health:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchHealth();
    const interval = setInterval(fetchHealth, 10000); // Poll every 10s

    return () => clearInterval(interval);
  }, []);

  if (loading) {
    return <div>Loading...</div>;
  }

  const getStatusColor = (status: string) => {
    switch (status) {
      case 'healthy':
      case 'up':
        return 'bg-green-500';
      case 'degraded':
        return 'bg-yellow-500';
      case 'unhealthy':
      case 'down':
        return 'bg-red-500';
      default:
        return 'bg-gray-500';
    }
  };

  return (
    <div className="p-6">
```

```
      <h1 className="text-3xl font-bold mb-6">System Monitoring</h1>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-4">
        <Card>
          <CardHeader>
            <CardTitle>Status</CardTitle>
          </CardHeader>
          <CardContent>
            <Badge className={getStatusColor(health?.status || '')}>
              {health?.status?.toUpperCase()}
            </Badge>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle>Uptime</CardTitle>
          </CardHeader>
          <CardContent>
            <p className="text-2xl font-bold">
              {health?.uptime ? Math.floor(health.uptime / 3600) : 0}h
            </p>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle>Version</CardTitle>
          </CardHeader>
          <CardContent>
            <p className="text-2xl font-bold">{health?.version}</p>
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle>Last Check</CardTitle>
          </CardHeader>
          <CardContent>
            <p className="text-sm">
              {health?.timestamp
                ? new Date(health.timestamp).toLocaleTimeString()
                : 'N/A'}
            </p>
          </CardContent>
        </Card>
      </div>

      <div className="grid grid-cols-1 md:grid-cols-3 gap-4 mt-6">
        <Card>
          <CardHeader>
            <CardTitle>Database</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center justify-between">
              <Badge className={getStatusColor(health?.checks.database.status || '')}>
                {health?.checks.database.status?.toUpperCase()}
              </Badge>
              {health?.checks.database.latency && (
                <span className="text-sm text-muted-foreground">
                  {health.checks.database.latency}ms
                </span>
              )}
            </div>
            {health?.checks.database.error && (
              <p className="text-sm text-red-500 mt-2">
                {health.checks.database.error}
              </p>
            )}
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle>Redis</CardTitle>
          </CardHeader>
          <CardContent>
            <div className="flex items-center justify-between">
              <Badge className={getStatusColor(health?.checks.redis.status || '')}>
                {health?.checks.redis.status?.toUpperCase()}
              </Badge>
              {health?.checks.redis.latency && (
                <span className="text-sm text-muted-foreground">
                  {health.checks.redis.latency}ms
                </span>
              )}
            </div>
            {health?.checks.redis.error && (
              <p className="text-sm text-red-500 mt-2">
                {health.checks.redis.error}
              </p>
            )}
          </CardContent>
        </Card>

        <Card>
          <CardHeader>
            <CardTitle>Storage (S3)</CardTitle>
          </CardHeader>
          <CardContent>
            <Badge className={getStatusColor(health?.checks.storage.status || '')}>
              {health?.checks.storage.status?.toUpperCase()}
            </Badge>
            {health?.checks.storage.error && (
              <p className="text-sm text-red-500 mt-2">
                {health?.checks.storage.error}
              </p>
            )}
          </CardContent>
        </Card>
      </div>
    </div>
  );
}
```

✅ CHECKLIST DE IMPLEMENTACIÓN

Semana 1: Fundamentos

- [ ] Crear `next.config.optimized.js`
- [ ] Optimizar `Dockerfile` y `.dockerignore`

- [ ] Actualizar scripts en `package.json`
- [ ] Crear health check endpoint `/api/health`
- [ ] Rotar credenciales expuestas en `.env`

Semana 2: CI/CD

- [ ] Configurar GitHub Actions (`.github/workflows/ci-cd.yml`)
- [ ] Crear scripts de deploy (`scripts/deploy-*.sh`)
- [ ] Configurar ambientes en GitHub (staging, production)
- [ ] Implementar database backup script
- [ ] Configurar rollback mechanism

Semana 3: Monitoring

- [ ] Implementar dashboard de monitoring
- [ ] Configurar alertas (Sentry, PagerDuty)
- [ ] Implementar logging estructurado (Winston)
- [ ] Configurar métricas de performance
- [ ] Documentar proceso de deploy

📞 CONTACTO

Auditor: Arquitecto de Software Senior
Fecha: 7 de Diciembre, 2025

Firma Digital: ✅ Auditoria completada satisfactoriamente