

Guía de Despliegue - Sistema ERP Completo





Guía completa para desplegar el Sistema ERP en diferentes entornos de producción.

Opciones de Despliegue

1. Despliegue en Vercel (Recomendado)
 2. Despliegue con Docker
 3. Despliegue en VPS/Servidor Dedicado
 4. Despliegue en AWS/Google Cloud/Azure
-

Preparación General

Prerequisitos

-  Node.js 18+
-  PostgreSQL 14+
-  Dominio configurado (opcional)
-  Certificado SSL (recomendado)

Variables de Entorno Requeridas

```
# Base de datos
DATABASE_URL="postgresql://usuario:password@host:5432/database_name"

# Autenticación
NEXTAUTH_URL="https://tu-dominio.com"
NEXTAUTH_SECRET="your-super-secret-key-here"

# Configuración adicional
NODE_ENV="production"
APP_URL="https://tu-dominio.com"

# Integraciones (opcional)
OPENPAY_API_KEY="your-openpay-key"
OPENPAY_MERCHANT_ID="your-merchant-id"
EMAIL_SERVER="smtp://username:password@smtp.gmail.com:587"
```

Opción 1: Despliegue en Vercel

Paso 1: Preparar el Repositorio

```
# Clonar el proyecto
git clone https://github.com/tu-usuario/sistema-erp-completo.git
cd sistema-erp-completo/app

# Instalar dependencias
npm install

# Crear build local para verificar
npm run build
```

Paso 2: Configurar Base de Datos

```
# Usar un servicio como Neon, Supabase, o PlanetScale
# Ejemplo con Neon:
# 1. Crear cuenta en https://neon.tech
# 2. Crear nueva base de datos PostgreSQL
# 3. Obtener CONNECTION_STRING

# Configurar esquema
npx prisma generate
npx prisma db push
```

Paso 3: Deploy en Vercel

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

# Hacer login
vercel login

# Deploy
vercel --prod

# Configurar variables de entorno en Vercel dashboard
```

Configuración en Vercel Dashboard

1. Ir a tu proyecto en vercel.com
 2. Settings → Environment Variables
 3. Agregar todas las variables necesarias
 4. Redeploy si es necesario
-

Opción 2: Despliegue con Docker

Dockerfile

```
FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies based on the preferred package manager
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

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Generate Prisma client
RUN npx prisma generate

# Build the application
RUN npm run build

# Production image, copy all the files and run next
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

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
ENV HOSTNAME "0.0.0.0"

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

docker-compose.yml

```

version: '3.8'
services:
  app:
    build: .
    ports:
      - "3000:3000"
    environment:
      - DATABASE_URL=${DATABASE_URL}
      - NEXTAUTH_URL=${NEXTAUTH_URL}
      - NEXTAUTH_SECRET=${NEXTAUTH_SECRET}
    depends_on:
      - postgres
    restart: unless-stopped

  postgres:
    image: postgres:16
    environment:
      POSTGRES_DB: erp_db
      POSTGRES_USER: erp_user
      POSTGRES_PASSWORD: your_secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    ports:
      - "5432:5432"
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - app
    restart: unless-stopped

volumes:
  postgres_data:

```

Comandos de Deploy

```

# Construir y ejecutar
docker-compose up -d --build

# Ver logs
docker-compose logs -f app

# Ejecutar migraciones
docker-compose exec app npx prisma db push

# Backup de base de datos
docker-compose exec postgres pg_dump -U erp_user erp_db > backup.sql

```

Opción 3: VPS/Servidor Dedicado

Instalación en Ubuntu 22.04

Paso 1: Preparar el Servidor

```
# Actualizar sistema
sudo apt update && sudo apt upgrade -y

# Instalar Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Instalar PostgreSQL
sudo apt install postgresql postgresql-contrib -y

# Instalar PM2 para gestión de procesos
sudo npm install -g pm2

# Instalar Nginx como reverse proxy
sudo apt install nginx -y
```

Paso 2: Configurar PostgreSQL

```
# Configurar usuario y base de datos
sudo -u postgres psql

CREATE USER erp_user WITH PASSWORD 'your_secure_password';
CREATE DATABASE erp_db OWNER erp_user;
GRANT ALL PRIVILEGES ON DATABASE erp_db TO erp_user;
\q
```

Paso 3: Configurar la Aplicación

```
# Clonar el proyecto
git clone https://github.com/tu-usuario/sistema-erp-completo.git
cd sistema-erp-completo/app

# Instalar dependencias
npm install

# Configurar variables de entorno
cp .env.example .env
nano .env

# Generar cliente Prisma y aplicar esquema
npx prisma generate
npx prisma db push

# Construir la aplicación
npm run build
```

Paso 4: Configurar PM2

```
# Crear archivo de configuración PM2
cat > ecosystem.config.js << EOF
module.exports = {
  apps: [{
    name: 'erp-sistema',
    script: 'node_modules/.bin/next',
    args: 'start',
    instances: 'max',
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3000
    }
  }]
}
EOF

# Iniciar la aplicación
pm2 start ecosystem.config.js
pm2 save
pm2 startup
```

Paso 5: Configurar Nginx

```
# /etc/nginx/sites-available/erp-sistema
server {
    listen 80;
    server_name tu-dominio.com www.tu-dominio.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
        proxy_read_timeout 300s;
        proxy_connect_timeout 75s;
    }

    # Seguridad adicional
    add_header X-Frame-Options DENY;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
}
```

```
# Habilitar sitio
sudo ln -s /etc/nginx/sites-available/erp-sistema /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Paso 6: Configurar SSL con Let's Encrypt

```
# Instalar Certbot
sudo apt install certbot python3-certbot-nginx -y

# Obtener certificado SSL
sudo certbot --nginx -d tu-dominio.com -d www.tu-dominio.com

# Configurar renovación automática
sudo crontab -e
# Agregar línea:
0 12 * * * /usr/bin/certbot renew --quiet
```



Opción 4: Cloud Providers

AWS Deployment

Usando AWS Amplify

```
# Instalar AWS CLI
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install

# Configurar credenciales
aws configure

# Deploy con Amplify
npm install -g @aws-amplify/cli
amplify init
amplify add hosting
amplify publish
```

Usando EC2 + RDS

1. **Crear instancia EC2** con Ubuntu 22.04
2. **Crear instancia RDS** PostgreSQL
3. **Configurar Security Groups** para permitir tráfico
4. **Seguir pasos de VPS** pero usando RDS como base de datos

Google Cloud Deployment

Usando Cloud Run

```
# Dockerfile optimizado para Cloud Run
FROM node:18-alpine AS base
# ... (mismo Dockerfile de arriba)
```

```
# Build y deploy
gcloud auth login
gcloud config set project tu-proyecto-id
gcloud builds submit --tag gcr.io/tu-proyecto-id/erp-sistema
gcloud run deploy --image gcr.io/tu-proyecto-id/erp-sistema --platform managed
```

Azure Deployment

Usando Azure App Service

```
# Instalar Azure CLI
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

# Login y deploy
az login
az group create --name erp-rg --location "East US"
az appservice plan create --name erp-plan --resource-group erp-rg --sku B1 --is-linux
az webapp create --resource-group erp-rg --plan erp-plan --name tu-erp-app --runtime "
NODE|18-lts"
```

Configuración de Seguridad

SSL/TLS

```
# Configurar headers de seguridad en Nginx
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains; preload";
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
add_header X-XSS-Protection "1; mode=block";
add_header Referrer-Policy "strict-origin-when-cross-origin";
```

Firewall

```
# Configurar UFW en Ubuntu
sudo ufw enable
sudo ufw allow ssh
sudo ufw allow 'Nginx Full'
sudo ufw allow 5432 # Solo si necesitas acceso externo a PostgreSQL
```

Backup Automático

```
# Script de backup diario
cat > /home/ubuntu/backup.sh << EOF
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
pg_dump -h localhost -U erp_user erp_db > /backups/erp_backup_$(DATE).sql
find /backups -name "erp_backup_*.sql" -mtime +7 -delete
EOF

# Programar en crontab
echo "0 2 * * * /home/ubuntu/backup.sh" | crontab -
```




Monitoreo y Logs

Configurar Logging

```
# PM2 logs
pm2 logs erp-sistema

# Logs de Nginx
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log

# Logs de PostgreSQL
sudo tail -f /var/log/postgresql/postgresql-14-main.log
```

Monitoreo con PM2

```
# Instalar PM2 monitoring
pm2 install pm2-server-monit

# Ver estadísticas
pm2 monit
```



Solución de Problemas Comunes

Error de Conexión a Base de Datos

```
# Verificar conexión PostgreSQL
psql -h localhost -U erp_user -d erp_db

# Verificar configuración de red PostgreSQL
sudo nano /etc/postgresql/14/main/postgresql.conf
# listen_addresses = 'localhost'

sudo nano /etc/postgresql/14/main/pg_hba.conf
# local    all             erp_user                                md5
```

Error de Permisos

```
# Permisos de archivos
sudo chown -R ubuntu:ubuntu /home/ubuntu/sistema-erp-completo
chmod +x /home/ubuntu/sistema-erp-completo/app/node_modules/.bin/next
```

Problemas de Memoria







```
# Aumentar memoria swap
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Problemas de Build







```
# Limpiar caché y reinstalar
rm -rf .next node_modules package-lock.json
npm install
npm run build
```

Checklist Post-Despliegue






Verificaciones Básicas

- ☐  Aplicación accesible desde navegador
- ☐  Login funciona correctamente
- ☐  Base de datos conectada
- ☐  SSL configurado (HTTPS)
- ☐  Todas las rutas funcionan
- ☐  APIs responden correctamente

Verificaciones de Seguridad

- ☐  Variables de entorno seguras
- ☐  Headers de seguridad configurados
- ☐  Firewall habilitado
- ☐  Backup automático funcionando
- ☐  Logs configurados
- ☐  Monitoreo habilitado

Verificaciones de Rendimiento

- ☐  Tiempos de respuesta < 2s
- ☐  Base de datos optimizada
- ☐  Caché configurado
- ☐  Compresión gzip habilitada
- ☐  CDN configurado (opcional)

Soporte Post-Despliegue

Mantenimiento Recomendado

- **Diario:** Verificar logs de errores
- **Semanal:** Verificar backups
- **Mensual:** Actualizar dependencias de seguridad
- **Trimestral:** Análisis completo de seguridad

Actualizaciones

```
# Actualizar aplicación
git pull origin main
npm install
npm run build
pm2 restart erp-sistema

# Actualizar base de datos
npx prisma db push
```



¡Sistema ERP desplegado y listo para producción! Sigue las mejores prácticas de seguridad y mantén el sistema actualizado.