Guía de Deployment - Sistema de Notificaciones

Esta guía proporciona instrucciones paso a paso para desplegar el sistema completo de notificaciones de CitaPlanner en producción.

Tabla de Contenidos

Pre-requisitos

Antes de comenzar, asegúrate de tener:

- [ ] Acceso al servidor/plataforma de deployment (Vercel, Railway, etc.)
- [ ] Acceso a la base de datos PostgreSQL
- [ ] Cuenta de Evolution API configurada (para WhatsApp)
- [ ] Node.js 18+ instalado localmente
- [ ] Git configurado con acceso al repositorio

Configuración de Variables de Entorno

1. Variables Existentes

Verifica que tengas configuradas las variables básicas:

```
DATABASE_URL=postgresql://user:password@host:5432/database
NEXTAUTH_URL=https://tu-dominio.com
NEXTAUTH_SECRET=tu_secret_aqui
```

2. Nuevas Variables para Notificaciones

Agrega las siguientes variables a tu archivo `.env` o en la configuración de tu plataforma:

```
# Sistema de Notificaciones
NOTIFICATION_AUTOMATION_ENABLED=true
CRON_SECRET=genera_un_token_secreto_muy_seguro_aqui
NOTIFICATION_LOG_RETENTION_DAYS=90

# VAPID Keys para Push Notifications (generar en siguiente paso)
NEXT_PUBLIC_VAPID_PUBLIC_KEY=tu_vapid_public_key
VAPID_PRIVATE_KEY=tu_vapid_private_key

# Evolution API (WhatsApp)
EVOLUTION_API_URL=https://tu-evolution-api.com
EVOLUTION_API_KEY=tu_api_key_aqui
WHATSAPP_INSTANCE_NAME=citaplanner
```

3. Generar CRON_SECRET

```
# Opción 1: Usando OpenSSL
openssl rand -base64 32

# Opción 2: Usando Node.js
node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"

# Opción 3: Online
# Visita: https://generate-secret.vercel.app/32
```

Generación de VAPID Keys

Las VAPID keys son necesarias para las notificaciones push web.

Método 1: Usando web-push CLI

```
# Instalar web-push globalmente
npm install -g web-push

# Generar keys
web-push generate-vapid-keys

# Output:
# Public Key: BN...
# Private Key: ...
```

Método 2: Usando script Node.js

Crea un archivo `generate-vapid.js`:

```
const webpush = require('web-push');

const vapidKeys = webpush.generateVAPIDKeys();

console.log('VAPID Keys generadas:');
console.log('');
console.log('Public Key:');
console.log(vapidKeys.publicKey);
console.log('');
console.log('Private Key:');
console.log(vapidKeys.privateKey);
console.log('');
console.log('Agregar a .env:');
console.log(`NEXT_PUBLIC_VAPID_PUBLIC_KEY=${vapidKeys.publicKey}`);
console.log(`VAPID_PRIVATE_KEY=${vapidKeys.privateKey}`);
```

Ejecutar:

```
npm install web-push
node generate-vapid.js
```

Configurar en tu Plataforma

Vercel:

1. Ve a tu proyecto → Settings → Environment Variables
2. Agrega `NEXT_PUBLIC_VAPID_PUBLIC_KEY` (Production, Preview, Development)
3. Agrega `VAPID_PRIVATE_KEY` (Production, Preview, Development)

Railway:

1. Ve a tu proyecto → Variables
2. Agrega las variables con sus valores

Otras plataformas:

Consulta la documentación específica de tu plataforma.

Configuración de Evolution API

Evolution API es el servicio que permite enviar mensajes de WhatsApp.

1. Obtener Acceso a Evolution API

Opciones:

A. Usar servicio hospedado:
- Contacta a un proveedor de Evolution API
- Obtén tu URL y API Key

B. Auto-hospedar:

```
# Clonar repositorio
git clone https://github.com/EvolutionAPI/evolution-api.git
cd evolution-api

# Configurar
cp .env.example .env
# Editar .env con tus configuraciones

# Iniciar con Docker
docker-compose up -d
```

2. Crear Instancia de WhatsApp

```
# Crear instancia
curl -X POST https://tu-evolution-api.com/instance/create \
  -H "apikey: tu_api_key" \
  -H "Content-Type: application/json" \
  -d '{
    "instanceName": "citaplanner",
    "qrcode": true
  }'

# Obtener QR Code
curl -X GET https://tu-evolution-api.com/instance/connect/citaplanner \
  -H "apikey: tu_api_key"

# Escanear QR con WhatsApp
```

3. Verificar Conexión

```
curl -X GET https://tu-evolution-api.com/instance/connectionState/citaplanner \
  -H "apikey: tu_api_key"

# Response esperado:
# {
#   "instance": "citaplanner",
#   "state": "open"
# }
```

4. Configurar en CitaPlanner

Actualiza las variables de entorno:

```
EVOLUTION_API_URL=https://tu-evolution-api.com
EVOLUTION_API_KEY=tu_api_key
WHATSAPP_INSTANCE_NAME=citaplanner
```

Configuración de Cron Jobs

Opción 1: Vercel Cron (Recomendado para Vercel)

1. Crear `vercel.json` en la raíz del proyecto:

```
{
  "crons": [
    {
      "path": "/api/cron/send-reminders",
      "schedule": "0 * * * *"
    }
  ]
}
```

2. Commit y push:

```
git add vercel.json
git commit -m "feat: add vercel cron configuration"
git push
```

3. Desplegar:

```
vercel --prod
```

4. Verificar en Vercel Dashboard:

- Ve a tu proyecto → Settings → Cron Jobs
- Deberías ver el job configurado

Opción 2: Servicio Externo (Para otras plataformas)

Usando cron-job.org:

1. Crea cuenta en https://cron-job.org
2. Crea nuevo cron job:
   - **Title**: CitaPlanner - Send Reminders
   - **URL**: `https://tu-dominio.com/api/cron/send-reminders`
   - **Schedule**: `0 * * * *` (cada hora)
   - **Request Method**: GET
   - **Headers**:
     - Name: `Authorization`
     - Value: `Bearer tu_cron_secret`

Usando EasyCron:

1. Crea cuenta en https://www.easycron.com
2. Crea nuevo cron job:
   - **URL**: `https://tu-dominio.com/api/cron/send-reminders`
   - **Cron Expression**: `0 * * * *`
   - **HTTP Headers**: `Authorization: Bearer tu_cron_secret`

Usando GitHub Actions:

Crea `.github/workflows/send-reminders.yml`:

```yaml
name: Send Appointment Reminders

on:
  schedule:
    - cron: '0 * * * *'  # Cada hora
  workflow_dispatch:  # Permite ejecución manual

jobs:
  send-reminders:
    runs-on: ubuntu-latest
    steps:
      - name: Call Cron Endpoint
        run: |
          response=$(curl -s -w "\n%{http_code}" -X GET \
            -H "Authorization: Bearer ${{ secrets.CRON_SECRET }}" \
            https://tu-dominio.com/api/cron/send-reminders)

          http_code=$(echo "$response" | tail -n1)
          body=$(echo "$response" | head -n-1)

          echo "HTTP Status: $http_code"
          echo "Response: $body"

          if [ "$http_code" != "200" ]; then
            echo "Error: Cron job failed"
            exit 1
          fi
```

Configurar secret en GitHub:
1. Ve a tu repositorio → Settings → Secrets and variables → Actions
2. Agrega `CRON_SECRET` con tu token

Migraciones de Base de Datos

Las migraciones ya están incluidas en el schema de Prisma.

1. Verificar Schema

```bash
cd app
cat prisma/schema.prisma | grep -A 20 "model NotificationSettings"
```

2. Generar Cliente de Prisma

```bash
cd app
npx prisma generate
```

3. Aplicar Migraciones

```bash
# Desarrollo
npx prisma migrate dev

# Producción
npx prisma migrate deploy
```

4. Verificar Tablas

```
-- Conectar a la base de datos
psql $DATABASE_URL

-- Verificar tablas
\dt notification*

-- Deberías ver:
-- notification_settings
-- notification_templates
-- notification_logs
-- push_subscriptions
```

Seed de Datos Iniciales

1. Crear Configuración de Notificaciones

```
-- Insertar configuración por defecto para cada tenant
INSERT INTO notification_settings (
  id,
  tenant_id,
  whatsapp_enabled,
  push_enabled,
  email_enabled,
  sms_enabled,
  appointment_reminder_enabled,
  appointment_reminder_times,
  auto_confirm_enabled,
  evolution_api_url,
  evolution_api_key,
  whatsapp_instance_name,
  created_at,
  updated_at
) VALUES (
  gen_random_uuid(),
  'tu_tenant_id',
  true,
  true,
  true,
  false,
  true,
  '[1440, 60]',
  false,
  'https://tu-evolution-api.com',
  'tu_api_key',
  'citaplanner',
  NOW(),
  NOW()
);
```

2. Crear Plantillas por Defecto

```
-- Plantilla de Confirmación (WhatsApp)
INSERT INTO notification_templates (
  id,
  tenant_id,
  name,
  type,
  channel,
  subject,
  message,
  is_active,
  is_default,
  created_at,
  updated_at
) VALUES (
  gen_random_uuid(),
  'tu_tenant_id',
  'Confirmación de Cita - WhatsApp',
  'APPOINTMENT_CONFIRMATION',
  'WHATSAPP',
  NULL,
  '¡Hola {{clientName}}! 👋

Tu cita ha sido confirmada:

📅 Fecha: {{appointmentDate}}
🕐 Hora: {{appointmentTime}}
💼 Servicio: {{serviceName}}
👤 Profesional: {{professionalName}}
📍 Sucursal: {{branchName}}

¡Te esperamos!',
  true,
  true,
  NOW(),
  NOW()
);

-- Plantilla de Recordatorio (WhatsApp)
INSERT INTO notification_templates (
  id,
  tenant_id,
  name,
  type,
  channel,
  subject,
  message,
  is_active,
  is_default,
  created_at,
  updated_at
) VALUES (
  gen_random_uuid(),
  'tu_tenant_id',
  'Recordatorio de Cita - WhatsApp',
  'APPOINTMENT_REMINDER',
  'WHATSAPP',
  NULL,
  '¡Hola {{clientName}}! 👋

Te recordamos tu cita en {{timeUntil}}:

📅 Fecha: {{appointmentDate}}
```

```
🕐 Hora: {{appointmentTime}}
💼 Servicio: {{serviceName}}
👤 Profesional: {{professionalName}}
📍 Sucursal: {{branchName}}

¡Te esperamos!',
  true,
  true,
  NOW(),
  NOW()
);

-- Plantilla de Cancelación (WhatsApp)
INSERT INTO notification_templates (
  id,
  tenant_id,
  name,
  type,
  channel,
  subject,
  message,
  is_active,
  is_default,
  created_at,
  updated_at
) VALUES (
  gen_random_uuid(),
  'tu_tenant_id',
  'Cancelación de Cita - WhatsApp',
  'APPOINTMENT_CANCELLATION',
  'WHATSAPP',
  NULL,
  'Hola {{clientName}},

Tu cita ha sido cancelada:

📅 Fecha: {{appointmentDate}}
🕐 Hora: {{appointmentTime}}
💼 Servicio: {{serviceName}}

Si deseas reagendar, contáctanos.',
  true,
  true,
  NOW(),
  NOW()
);
```

3. Script de Seed Automatizado

Crea `app/prisma/seed-notifications.ts`:

```
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

async function main() {
  console.log('Seeding notification system...');

  // Obtener todos los tenants
  const tenants = await prisma.tenant.findMany();

  for (const tenant of tenants) {
    // Crear configuración si no existe
    const existingSettings = await prisma.notificationSettings.findUnique({
      where: { tenantId: tenant.id },
    });

    if (!existingSettings) {
      await prisma.notificationSettings.create({
        data: {
          tenantId: tenant.id,
          whatsappEnabled: true,
          pushEnabled: true,
          emailEnabled: true,
          smsEnabled: false,
          appointmentReminderEnabled: true,
          appointmentReminderTimes: JSON.stringify([1440, 60]),
          autoConfirmEnabled: false,
          evolutionApiUrl: process.env.EVOLUTION_API_URL,
          evolutionApiKey: process.env.EVOLUTION_API_KEY,
          whatsappInstanceName: process.env.WHATSAPP_INSTANCE_NAME,
        },
      });
      console.log(`✓ Settings created for tenant ${tenant.name}`);
    }

    // Crear plantillas por defecto
    // ... (código de plantillas)
  }

  console.log('✓ Notification system seeded successfully');
}

main()
  .catch((e) => {
    console.error(e);
    process.exit(1);
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

Ejecutar:

```
cd app
npx ts-node prisma/seed-notifications.ts
```

Verificación del Sistema

1. Verificar Configuración

```
# Probar endpoint de cron
curl -X GET https://tu-dominio.com/api/cron/send-reminders \
  -H "Authorization: Bearer tu_cron_secret"

# Response esperado:
# {
#   "success": true,
#   "message": "Recordatorios procesados exitosamente",
#   "stats": { "sent": 0, "failed": 0, "skipped": 0 }
# }
```

2. Crear Cita de Prueba

```
# Crear cita
curl -X POST https://tu-dominio.com/api/appointments \
  -H "Content-Type: application/json" \
  -H "Cookie: next-auth.session-token=..." \
  -d '{
    "clientId": "...",
    "serviceId": "...",
    "userId": "...",
    "branchId": "...",
    "startTime": "2025-10-10T10:00:00Z"
  }'
```

3. Verificar Notificación Enviada

```
-- Ver últimas notificaciones
SELECT * FROM notification_logs
ORDER BY created_at DESC
LIMIT 10;

-- Verificar estado
SELECT status, COUNT(*)
FROM notification_logs
GROUP BY status;
```

4. Probar Push Notifications

1. Abre la aplicación en el navegador
2. Ve a Configuración → Notificaciones
3. Habilita notificaciones push
4. Crea una cita de prueba
5. Deberías recibir una notificación push

5. Probar WhatsApp

1. Asegúrate de que Evolution API esté conectado
2. Crea una cita para un cliente con número de teléfono
3. Verifica que el cliente reciba el mensaje de WhatsApp

Monitoreo y Logs

1. Configurar Logging

En producción, considera usar un servicio de logging:

- **Vercel**: Logs integrados en el dashboard
- **Railway**: Logs en tiempo real en el dashboard
- **Sentry**: Para errores y excepciones
- **LogRocket**: Para sesiones de usuario

2. Queries Útiles para Monitoreo

```sql
-- Notificaciones en las últimas 24 horas
SELECT
  type,
  channel,
  status,
  COUNT(*) as count
FROM notification_logs
WHERE created_at > NOW() - INTERVAL '24 hours'
GROUP BY type, channel, status
ORDER BY count DESC;

-- Tasa de éxito por canal
SELECT
  channel,
  COUNT(*) FILTER (WHERE status = 'SENT') as sent,
  COUNT(*) FILTER (WHERE status = 'FAILED') as failed,
  ROUND(
    COUNT(*) FILTER (WHERE status = 'SENT')::numeric /
    COUNT(*)::numeric * 100,
    2
  ) as success_rate
FROM notification_logs
WHERE created_at > NOW() - INTERVAL '7 days'
GROUP BY channel;

-- Notificaciones fallidas recientes
SELECT
  type,
  channel,
  recipient_name,
  error_message,
  created_at
FROM notification_logs
WHERE status = 'FAILED'
  AND created_at > NOW() - INTERVAL '24 hours'
ORDER BY created_at DESC;
```

3. Alertas Recomendadas

Configura alertas para:

- Tasa de fallo > 10%
- Cron job no ejecutado en 2 horas
- Evolution API desconectado
- Errores críticos en logs

Rollback

Si necesitas revertir el deployment:

1. Rollback de Código

```
# Vercel
vercel rollback

# Railway
railway rollback

# Manual
git revert HEAD
git push
```

2. Deshabilitar Automatización

```
# Actualizar variable de entorno
NOTIFICATION_AUTOMATION_ENABLED=false
```

3. Detener Cron Jobs

- **Vercel**: Eliminar `vercel.json` y redesplegar
- **Servicios externos**: Pausar o eliminar el cron job

4. Rollback de Base de Datos (si es necesario)

```
# Revertir última migración
cd app
npx prisma migrate resolve --rolled-back <migration_name>
```

Checklist Final

Antes de considerar el deployment completo:

- [ ] Variables de entorno configuradas
- [ ] VAPID keys generadas y configuradas
- [ ] Evolution API conectado y funcionando
- [ ] Cron jobs configurados y ejecutándose
- [ ] Migraciones aplicadas
- [ ] Seed de datos completado
- [ ] Plantillas de notificaciones creadas
- [ ] Cita de prueba creada y notificación recibida
- [ ] Push notifications funcionando
- [ ] WhatsApp funcionando
- [ ] Logs monitoreados
- [ ] Alertas configuradas
- [ ] Documentación revisada

Soporte

Si encuentras problemas durante el deployment:

1. Revisa los logs de la aplicación
2. Verifica las variables de entorno
3. Consulta `NOTIFICATION_AUTOMATION.md` para troubleshooting
4. Verifica el estado de Evolution API
5. Revisa los logs de notificaciones en la base de datos

¡Deployment completado! 🎉