🚀 Guía de Despliegue en Render - HumanS

Resumen de tu Aplicación

Tu aplicación `server_postgres_cloud.py` es un servidor Flask con:

- **PostgreSQL** para persistencia de datos vitales
- **Email SMTP** para alertas críticas
- **WebSocket** (Flask-SocketIO) para datos en tiempo real
- **OpenAI API** para generación de reportes PDF

📋 Arquitectura en Render

Necesitarás crear **2 servicios**:

Servicio	Tipo	Descripción
HumanS Server	Web Service	Tu aplicación Python
PostgreSQL	Database	Base de datos persistente

🗄️ Paso 1: Crear Base de Datos PostgreSQL

1. Ve a **Render Dashboard → New → PostgreSQL**
2. Configura:
   - **Name**: `humans-db`
   - **Database**: `humans`
   - **User**: `humans_user`
   - **Region**: Elige la más cercana a tus usuarios
   - **Plan**:
     - **Free** (90 días, para pruebas)
     - **Starter $7/mes** (persistente, recomendado para producción)
3. **Copia el Internal Database URL** - lo usarás en el siguiente paso

> ⚠️ **Importante**: El plan Free se borra después de 90 días. Para producción, usa un plan de pago.

🌐 Paso 2: Crear Web Service

1. Ve a **New → Web Service**
2. Conecta tu repositorio Git o sube los archivos
3. Configura:

Configuración Básica

Campo	Valor
Name	`humans-server`
Region	Misma que la base de datos
Branch	`main`
Runtime	`Python 3`
Build Command	`pip install -r requirements.txt`
Start Command	`gunicorn --worker-class eventlet -w 1 server_postgres_cloud:app`

⚙️ Variables de Entorno (Environment Variables)

Añade estas variables en **Environment → Add Environment Variable**:

```
DATABASE_URL        = [Pega el Internal Database URL del paso 1]
OPENAI_API_KEY      = sk-tu-clave-openai
EMAIL_FROM          = tu-email@gmail.com
EMAIL_PASS          = tu-contraseña-de-aplicacion
SMTP_SERVER         = smtp.gmail.com
SMTP_PORT           = 465
EMAIL_TO            = email-destinatario@ejemplo.com (opcional)
```

> 💡 **Tip**: Si usas Gmail, necesitas una "Contraseña de aplicación", no tu contraseña normal.

📦 Paso 3: Archivo requirements.txt

Crea este archivo en la raíz de tu proyecto:

txt

```
flask>=2.3.0
flask-socketio>=5.3.0
eventlet>=0.33.0
gunicorn>=21.0.0
psycopg2-binary>=2.9.0
python-dotenv>=1.0.0
openai>=1.0.0
weasyprint>=60.0
numpy>=1.24.0
```

Paso 4: Archivos Adicionales Necesarios

render.yaml (Opcional - Infrastructure as Code)

yaml

```yaml
services:
  - type: web
    name: humans-server
    runtime: python
    buildCommand: pip install -r requirements.txt
    startCommand: gunicorn --worker-class eventlet -w 1 server_postgres_cloud:app
    envVars:
      - key: DATABASE_URL
        fromDatabase:
          name: humans-db
          property: connectionString
      - key: OPENAI_API_KEY
        sync: false
      - key: EMAIL_FROM
        sync: false
      - key: EMAIL_PASS
        sync: false
      - key: SMTP_SERVER
        value: smtp.gmail.com
      - key: SMTP_PORT
        value: "465"

databases:
  - name: humans-db
    plan: starter
```

🔐 Configuración de Email (Gmail)

Para que Gmail funcione, necesitas una **Contraseña de Aplicación**:

1. Ve a myaccount.google.com
2. **Seguridad → Verificación en 2 pasos** (actívala si no está)
3. **Seguridad → Contraseñas de aplicaciones**
4. Crea una nueva para "Correo" + "Otro (HumanS)"
5. Usa esa contraseña de 16 caracteres en `EMAIL_PASS`

🔧 Consideraciones Importantes

1. WebSocket con Render

Tu aplicación usa Flask-SocketIO. El comando de inicio ya está configurado correctamente:

```bash
gunicorn --worker-class eventlet -w 1 server_postgres_cloud:app
```

- **`--worker-class eventlet`**: Necesario para WebSocket
- **`-w 1`**: Un solo worker (SocketIO requiere esto)

2. Puerto

Render asigna el puerto automáticamente via `PORT`. Tu código ya lo maneja:

```python
port = int(os.environ.get("PORT", 5050))
```

3. Base de Datos - Connection String

Render usa `postgres://` pero psycopg2 necesita `postgresql://`. Tu código ya lo convierte:

```python
if connection_string.startswith("postgres://"):
    connection_string = connection_string.replace("postgres://", "postgresql://", 1)
```

4. Dependencias del Sistema para WeasyPrint

Si WeasyPrint falla, crea un archivo `render-build.sh`:

```bash
#!/bin/bash
apt-get update && apt-get install -y \
    libpango-1.0-0 \
    libpangocairo-1.0-0 \
    libgdk-pixbuf2.0-0 \
    libffi-dev \
    shared-mime-info
pip install -r requirements.txt
```

Y cambia el **Build Command** a: `./render-build.sh`

📊 Verificación del Despliegue

Después de desplegar, verifica:

1. **Logs**: Dashboard → Tu servicio → Logs
2. **Health Check**: Visita `https://tu-app.onrender.com/api/diagnostics`
3. **Base de datos**: Debería mostrar `"status": "connected"`

Respuesta esperada de `/api/diagnostics`:

```json
{
  "database": {
    "type": "PostgreSQL",
    "status": "connected"
  },
  "email_config": {
    "enabled": true,
    "recipient": "tu-email@ejemplo.com"
  }
}
```

💰 Costos Estimados

Componente	Plan	Costo/mes
Web Service	Free	$0 (se duerme tras 15min inactividad)
Web Service	Starter	$7
PostgreSQL	Free	$0 (90 días)
PostgreSQL	Starter	$7
Total Producción		**~$14/mes**

🪛 Solución de Problemas

Error: "No module named psycopg2"

Asegúrate de tener `psycopg2-binary` en requirements.txt

Error: "Connection refused" a PostgreSQL

Verifica que `DATABASE_URL` esté configurado correctamente

WebSocket no conecta

- Verifica que usas `eventlet` como worker
- Solo 1 worker (`-w 1`)
- El cliente debe conectar a `wss://` (no `ws://`)

Emails no se envían

- Verifica contraseña de aplicación de Gmail
- Revisa los logs para ver errores SMTP

🚀 Comandos de Despliegue Rápido

Si usas Render CLI:

```bash
```

```
# Login
render login

# Deploy
render deploy
```

✅ Checklist Final

- [ ] PostgreSQL creado en Render
- [ ] Web Service configurado
- [ ] `requirements.txt` actualizado
- [ ] Variables de entorno configuradas
- [ ] Contraseña de aplicación de Gmail generada
- [ ] OpenAI API key configurada
- [ ] Primer despliegue exitoso
- [ ] `/api/diagnostics` muestra base de datos conectada
- [ ] Email de prueba enviado

¿Necesitas ayuda? Revisa los logs en tiempo real en el Dashboard de Render.