






Guía de Persistencia de Base de Datos - CitaPlanner

Tabla de Contenidos

1. [Introducción](#)
 2. [Configuración de Volúmenes en Easypanel](#)
 3. [Sistema de Backups Automáticos](#)
 4. [Seed Script Idempotente](#)
 5. [Proceso de Deployment](#)
 6. [Restauración de Backups](#)
 7. [Troubleshooting](#)
 8. [Mejores Prácticas](#)
-

Introducción

CitaPlanner implementa un sistema completo de persistencia de datos que garantiza:

-  **Persistencia total:** Los datos nunca se pierden entre deployments
 -  **Backups automáticos:** Sistema de respaldo diario, semanal y mensual
 -  **Seed idempotente:** Datos iniciales sin duplicación
 -  **Fácil restauración:** Proceso simple para recuperar datos
 -  **Verificación de integridad:** Checksums y validación automática
-

Configuración de Volúmenes en Easypanel

1. Volumen de Base de Datos PostgreSQL

Configuración actual (ya implementada):

```
Servicio: PostgreSQL
Volumen: citaplanner-postgres-data
Mount Path: /var/lib/postgresql/data
Tipo: Persistent Volume
```

Verificación:

1. Ve a tu servicio PostgreSQL en Easypanel
2. Navega a la pestaña "Volumes"
3. Confirma que existe el volumen `citaplanner-postgres-data`
4. Verifica que está montado en `/var/lib/postgresql/data`

2. Volumen de Backups

Configuración requerida:

```
Servicio: CitaPlanner App
Volumen: citaplanner-backups
Mount Path: /backup-citaplanner
Tipo: Persistent Volume
```

Pasos para configurar:

1. En Easypanel, ve a tu servicio de CitaPlanner
2. Click en "Volumes" → "Add Volume"
3. Configura:
 - **Name:** citaplanner-backups
 - **Mount Path:** /backup-citaplanner
 - **Size:** Mínimo 10GB (recomendado 20GB)
4. Guarda y redeploya el servicio

3. Configuración en docker-compose.yml

El archivo `docker-compose.yml` ya está configurado correctamente:

```
services:
  postgres:
    volumes:
      - postgres_data:/var/lib/postgresql/data # ✓ Persistencia de datos
      - postgres_backups:/backup-citaplanner # ✓ Backups automáticos

volumes:
  postgres_data: # Volumen nombrado para datos
  postgres_backups: # Volumen nombrado para backups
```

Sistema de Backups Automáticos

Características del Sistema

- **Backups diarios:** Cada día a las 2:00 AM (configurable)
- **Backups semanales:** Cada domingo
- **Backups mensuales:** Primer día de cada mes
- **Rotación automática:**
 - Diarios: últimos 7 días
 - Semanales: últimas 4 semanas
 - Mensuales: últimos 6 meses
- **Compresión:** Todos los backups se comprimen con gzip
- **Verificación:** Checksum SHA256 para cada backup
- **Logs detallados:** Registro completo de cada operación

Configuración de Backups Automáticos

Opción 1: Cron Job en Easypanel (Recomendado)

1. En Easypanel, ve a tu servicio PostgreSQL
2. Agrega un nuevo "Cron Job":

```
# Backup diario a las 2:00 AM
0 2 * * * /app/scripts/pg_backup.sh
```

Opción 2: Cron Job Manual en el Contenedor

Si prefieres configurarlo manualmente:

```
# Conectarse al contenedor
docker exec -it citaplanner-app bash

# Editar crontab
crontab -e

# Agregar línea:
0 2 * * * /app/scripts/pg_backup.sh >> /backup-citaplanner/logs/cron.log 2>&1
```

Uso del Script de Backup

Backup Manual

```
# Ejecutar backup manual
./scripts/pg_backup.sh

# El script automáticamente:
# - Detecta el día (diario/semanal/mensual)
# - Crea el backup correspondiente
# - Verifica integridad
# - Genera checksum
# - Rota backups antiguos
```

Verificar Backups Existentes

```
# Verificar integridad de todos los backups
./scripts/pg_backup.sh --verify-only
```

Estructura de Directorios de Backup

```

/backup-citaplanner/
├── daily/
│   ├── citaplanner_daily_20250109_020000.sql.gz
│   ├── citaplanner_daily_20250109_020000.sql.gz.sha256
│   └── ... (últimos 7 días)
├── weekly/
│   ├── citaplanner_weekly_20250105_020000.sql.gz
│   ├── citaplanner_weekly_20250105_020000.sql.gz.sha256
│   └── ... (últimas 4 semanas)
├── monthly/
│   ├── citaplanner_monthly_20250101_020000.sql.gz
│   ├── citaplanner_monthly_20250101_020000.sql.gz.sha256
│   └── ... (últimos 6 meses)
└── logs/
    ├── backup_20250109.log
    └── ...
  
```

Seed Script Idempotente

¿Qué es un Seed Idempotente?

Un seed idempotente es un script que puede ejecutarse múltiples veces sin duplicar datos. Verifica si los datos ya existen antes de crearlos.

Características del Nuevo Seed

- ✓ **Verificación antes de crear:** Comprueba existencia de cada entidad
- ✓ **Sin eliminación de datos:** No borra información existente
- ✓ **Upsert inteligente:** Actualiza o crea según sea necesario
- ✓ **Seguro para producción:** Puede ejecutarse en cada deploy
- ✓ **Logs detallados:** Indica qué se creó y qué ya existía

Comportamiento del Seed

```

// Ejemplo de verificación antes de crear
let tenant = await prisma.tenant.findFirst({
  where: { email: 'contacto@bellavita.com' }
})

if (!tenant) {
  tenant = await prisma.tenant.create({ ... })
  console.log('✅ Tenant creado')
} else {
  console.log('❌ Tenant ya existe')
}
  
```

Datos que Crea el Seed (Solo si no existen)

1. **1 Empresa (Tenant):** Bella Vita Spa & Wellness
2. **1 Sucursal:** Sucursal Centro
3. **5 Usuarios:**
 - Admin: `admin@citaplanner.com` / `admin123`

- Manager: `manager@citaplanner.com` / `manager123`
- Profesional 1: `pro1@citaplanner.com` / `prof123`
- Profesional 2: `pro2@citaplanner.com` / `prof123`
- Recepcionista: `recepcionista@citaplanner.com` / `prof123`

4. **6 Servicios:** Faciales, masajes, manicure, corte, etc.

5. **6 Clientes:** Datos de ejemplo

6. **Horarios de trabajo:** Lunes a sábado

7. **6 Citas de ejemplo:** Algunas completadas, otras pendientes

8. **Pagos:** Para citas completadas

Ejecución del Seed

El seed se ejecuta automáticamente en el `docker-entrypoint.sh` **solo si la base de datos está vacía:**

```
# Verificar si la BD está vacía
if is_database_empty; then
    run_seed # Solo ejecuta si no hay usuarios
fi
```

Proceso de Deployment

Flujo Completo de Deployment

```
graph TD
    A[Deploy en Easypanel] --> B[Iniciar Contenedor]
    B --> C[docker-entrypoint.sh]
    C --> D[Validar DATABASE_URL]
    D --> E[Conectar a PostgreSQL]
    E --> F[Ejecutar Migraciones]
    F --> G{BD Vacía?}
    G -- Sí --> H[Ejecutar Seed]
    G -- No --> I[Omitir Seed]
    H --> J[Configurar Master Admin]
    I --> J
    J --> K[Iniciar Next.js]
```

Qué Sucede en Cada Deploy

1. Validación de Conexión

- Verifica `DATABASE_URL`
- Prueba conectividad con PostgreSQL
- Valida credenciales

2. Migraciones de Prisma

- Ejecuta `prisma migrate deploy`
- Aplica cambios de esquema pendientes
- Mantiene datos existentes intactos

3. Generación de Cliente Prisma

- Regenera el cliente Prisma
- Sincroniza con el esquema actual

4. Seed Condicional

- Verifica si hay usuarios en la BD
- Solo ejecuta si la BD está vacía
- Crea datos iniciales sin duplicar

5. Configuración Master Admin

- Verifica si existe configuración
- Solo crea si no existe
- Idempotente y seguro

Variables de Entorno Requeridas

```
# Base de datos
DATABASE_URL=postgresql://user:password@host:5432/database

# NextAuth
NEXTAUTH_URL=https://tu-dominio.com
NEXTAUTH_SECRET=tu-secret-aleatorio-seguro

# Master Admin (opcional)
MASTER_PASSWORD_HASH=$2b$10$... # Hash bcrypt del password
```

Restauración de Backups

Proceso de Restauración

1. Listar Backups Disponibles

```
# Ver todos los backups
ls -lh /backup-citaplanner/daily/
ls -lh /backup-citaplanner/weekly/
ls -lh /backup-citaplanner/monthly/

# Ver backups con fechas
find /backup-citaplanner -name "*.sql.gz" -type f -exec ls -lh {} \;
```

2. Verificar Integridad del Backup

```
# Verificar un backup específico
./scripts/pg_backup.sh --verify-only

# O verificar manualmente
gzip -t /backup-citaplanner/daily/citaplanner_daily_20250109_020000.sql.gz

# Verificar checksum
cd /backup-citaplanner/daily
sha256sum -c citaplanner_daily_20250109_020000.sql.gz.sha256
```

3. Restaurar desde Backup

```
# Restaurar desde un backup específico
./scripts/pg_backup.sh --restore /backup-citaplanner/daily/citaplanner_daily_20250109_020000.sql.gz

# El script te pedirá confirmación:
# ⚠️ ADVERTENCIA: Esta operación SOBREScribirá todos los datos actuales
# ¿Estás seguro de continuar? (escribe 'SI' para confirmar):
```

4. Verificar Restauración

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

# Verificar datos
SELECT COUNT(*) FROM users;
SELECT COUNT(*) FROM appointments;
SELECT COUNT(*) FROM clients;
```

Restauración Manual (Sin Script)

Si prefieres restaurar manualmente:

```
# 1. Descomprimir backup
gunzip -c /backup-citaplanner/daily/backup.sql.gz > /tmp/restore.sql

# 2. Conectar y restaurar
psql $DATABASE_URL < /tmp/restore.sql

# 3. Limpiar archivo temporal
rm /tmp/restore.sql
```

Troubleshooting

Problema: Volumen no persiste datos

Síntomas:

- Datos se pierden después de redeploy
- Base de datos vacía en cada inicio



Solución:

1. Verificar configuración del volumen en Easypanel:

```
bash
# En el contenedor PostgreSQL
docker exec -it postgres-container df -h
# Debe mostrar el volumen montado en /var/lib/postgresql/data
```

2. Verificar que el volumen es persistente (no temporal):

```
yaml
# En docker-compose.yml
volumes:
```

```
postgres_data: #  Correcto: volumen nombrado
# NO usar: /tmp/data #  Incorrecto: directorio temporal
```

3. Verificar permisos:

```
bash
docker exec -it postgres-container ls -la /var/lib/postgresql/data
# Debe ser propiedad del usuario postgres
```

Problema: Backups no se crean

Síntomas:

- Directorio de backups vacío
- Cron job no ejecuta

Solución:

1. Verificar que el volumen de backups está montado:

```
bash
docker exec -it app-container ls -la /backup-citaplanner
```

2. Verificar permisos del script:

```
bash
docker exec -it app-container ls -la /app/scripts/pg_backup.sh
# Debe tener permisos de ejecución (x)
chmod +x /app/scripts/pg_backup.sh
```

3. Ejecutar backup manual para ver errores:

```
bash
docker exec -it app-container /app/scripts/pg_backup.sh
```

4. Verificar cron job:

```
bash
docker exec -it app-container crontab -l
```

Problema: Seed duplica datos

Síntomas:

- Usuarios duplicados
- Servicios repetidos

Solución:

El nuevo seed es idempotente, pero si tienes el seed antiguo:

1. Actualizar a la nueva versión del seed:

```
bash
git pull origin feature/db-persistence
```

2. Si ya tienes datos duplicados, limpiar manualmente:

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

- Ver usuarios duplicados

```
SELECT email, COUNT() FROM users GROUP BY email HAVING COUNT() > 1;
```


- Eliminar duplicados (mantener el más antiguo)

```
DELETE FROM users a USING users b
```

```
WHERE a.id > b.id AND a.email = b.email;
```

```
```
```

## Problema: Error en migraciones

### Síntomas:

- Migraciones fallan en deploy
- Error "migration already applied"

### Solución:

1. Verificar estado de migraciones:

```
bash
```

```
docker exec -it app-container npx prisma migrate status
```

2. Resolver migraciones pendientes:

```
```bash
```

```
# Si hay migraciones pendientes
```

```
docker exec -it app-container npx prisma migrate deploy
```

```
# Si hay conflictos
```

```
docker exec -it app-container npx prisma migrate resolve --applied "migration_name"
```

```
```
```

1. En caso extremo, resetear migraciones (⚠ CUIDADO: borra datos):

```
bash
```

```
Solo en desarrollo
```

```
docker exec -it app-container npx prisma migrate reset
```

## Problema: No se puede conectar a PostgreSQL

### Síntomas:

- Error "connection refused"
- Timeout al conectar

### Solución:

1. Verificar que PostgreSQL está corriendo:

```
bash
```

```
docker ps | grep postgres
```

2. Verificar DATABASE\_URL:

```
bash
```

```
echo $DATABASE_URL
```

```
Debe ser: postgresql://user:pass@host:5432/db
```

3. Verificar conectividad de red:

```
bash
```

```
docker exec -it app-container ping postgres-host
```

```
docker exec -it app-container nc -zv postgres-host 5432
```

#### 4. Verificar logs de PostgreSQL:

```
bash
```

```
docker logs postgres-container
```

---

## Mejores Prácticas

---

### 1. Backups

#### ✓ Hacer:

- Configurar backups automáticos desde el día 1
- Verificar backups regularmente (al menos semanalmente)
- Mantener backups en múltiples ubicaciones (local + cloud)
- Probar restauración periódicamente
- Documentar el proceso de restauración

#### ✗ Evitar:

- Confiar solo en el volumen persistente
- No verificar integridad de backups
- Mantener backups indefinidamente sin rotación
- Almacenar backups solo en el mismo servidor

### 2. Migraciones

#### ✓ Hacer:

- Probar migraciones en desarrollo primero
- Hacer backup antes de aplicar migraciones en producción
- Usar `prisma migrate deploy` en producción
- Documentar cambios de esquema

#### ✗ Evitar:

- Usar `prisma migrate dev` en producción
- Aplicar migraciones sin backup previo
- Modificar migraciones ya aplicadas
- Saltarse migraciones

### 3. Seed de Datos

#### ✓ Hacer:

- Usar seed idempotente (como el nuevo)
- Verificar existencia antes de crear
- Mantener datos de ejemplo mínimos
- Documentar credenciales de prueba

#### ✗ Evitar:

- Eliminar datos existentes en seed
- Crear datos duplicados
- Usar seed destructivo en producción
- Hardcodear datos sensibles

### 4. Monitoreo

#### ✓ Hacer:

- Monitorear espacio en disco del volumen

- Revisar logs de backup regularmente
- Configurar alertas para fallos de backup
- Monitorear tamaño de base de datos

#### ✗ Evitar:

- Ignorar alertas de espacio en disco
- No revisar logs de errores
- Dejar que los backups llenen el disco

## 5. Seguridad

#### ✓ Hacer:

- Usar contraseñas fuertes para PostgreSQL
- Rotar credenciales periódicamente
- Encriptar backups sensibles
- Limitar acceso a volúmenes de datos

#### ✗ Evitar:

- Usar contraseñas por defecto
- Exponer DATABASE\_URL en logs
- Compartir credenciales sin encriptar
- Dar acceso root innecesario

## Checklist de Implementación

### Configuración Inicial

- ☐ Volumen PostgreSQL configurado en Easypanel
- ☐ Volumen de backups configurado en Easypanel
- ☐ DATABASE\_URL configurada correctamente
- ☐ Script de backup tiene permisos de ejecución
- ☐ Cron job configurado para backups automáticos

### Verificación Post-Deploy

- ☐ Aplicación inicia correctamente
- ☐ Migraciones se aplican sin errores
- ☐ Seed crea datos iniciales (solo primera vez)
- ☐ Datos persisten después de redeploy
- ☐ Backups se crean automáticamente
- ☐ Logs no muestran errores

### Mantenimiento Regular

- ☐ Verificar backups semanalmente
- ☐ Probar restauración mensualmente
- ☐ Revisar espacio en disco
- ☐ Actualizar documentación de cambios
- ☐ Rotar credenciales trimestralmente

## Recursos Adicionales

---

### Comandos Útiles

```
Ver tamaño de base de datos
docker exec -it postgres-container psql $DATABASE_URL -c "\l+"

Ver tablas y tamaños
docker exec -it postgres-container psql $DATABASE_URL -c "\dt+"

Contar registros en todas las tablas
docker exec -it postgres-container psql $DATABASE_URL -c "
SELECT schemaname,relname,n_live_tup
FROM pg_stat_user_tables
ORDER BY n_live_tup DESC;"

Ver conexiones activas
docker exec -it postgres-container psql $DATABASE_URL -c "
SELECT * FROM pg_stat_activity
WHERE datname = current_database();"

Espacio usado por volúmenes
docker system df -v
```

### Enlaces Útiles

- [Documentación de Prisma Migrate](https://www.prisma.io/docs/concepts/components/prisma-migrate) (https://www.prisma.io/docs/concepts/components/prisma-migrate)
- [PostgreSQL Backup Best Practices](https://www.postgresql.org/docs/current/backup.html) (https://www.postgresql.org/docs/current/backup.html)
- [Docker Volumes](https://docs.docker.com/storage/volumes/) (https://docs.docker.com/storage/volumes/)
- [Easypanel Documentation](https://easypanel.io/docs) (https://easypanel.io/docs)

---

## Soporte

Si encuentras problemas no cubiertos en esta guía:

1. Revisa los logs detallados en `/backup-citaplanner/logs/`
2. Verifica el estado de los servicios en Easypanel
3. Consulta la documentación de Prisma y PostgreSQL
4. Contacta al equipo de desarrollo con logs específicos

---

**Última actualización:** 9 de Octubre, 2025

**Versión:** 1.0.0

**Autor:** CitaPlanner Development Team