MANUAL DE CONFIGURACIÓN DE LA BASE DE DATOS

ZONA-44

(versión 1.0)

Fecha: 4 de noviembre de 2025

Versión: 1.0

ÍNDICE

1. Objetivo y alcance

Este manual explica cómo instalar, configurar y operar la base de datos para la aplicación ZONA-44 (backend Rails). Cubre Postgres y MySQL, muestra ejemplos de `database.yml`, variables de entorno, ejemplos para Docker, políticas de backup/restore, pruebas y tareas de mantenimiento.

2. Requisitos previos

- Acceso de administrador a la máquina (o permisos necesarios en Docker/Kubernetes).
- Ruby on Rails (versión usada en el proyecto). Ver `Gemfile`.
- Gestor de base de datos: PostgreSQL (recomendado) o MySQL.
- Herramientas clientes: `psql` / `pg_dump` / `pg_restore` para Postgres; `mysql` / `mysqldump` para MySQL.
- Espacio en disco suficiente para dumps y WALs.

3. Elección de SGBD

- Recomendación: PostgreSQL por características avanzadas (concurrencia, integridad, extensiones, JSONB). MySQL es viable si ya se usa en infraestructura existente.

4. Instalación (resumen)

4.1 PostgreSQL (Linux - apt)

- Actualizar paquetes
- sudo apt update; sudo apt install -y postgresql postgresql-contrib
- Iniciar servicio
  sudo systemctl enable --now postgresql

4.2 MySQL (Linux - apt)

- sudo apt update; sudo apt install -y mysql-server
- sudo systemctl enable --now mysql
- sudo mysql_secure_installation

5. Configuración para Rails (`config/database.yml`)

A continuación ejemplos para `config/database.yml`. Ajustar `username`, `password`, `host`, `port` y `database` según entorno.

5.1 PostgreSQL (ejemplo YAML)

development:

- adapter: postgresql
- encoding: unicode
- database: zona44_development
- pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
- username: <%= ENV.fetch('DB_USERNAME') { 'zona44' } %>
- password: <%= ENV.fetch('DB_PASSWORD') { 'password' } %>
- host: <%= ENV.fetch('DB_HOST') { 'localhost' } %>
- port: <%= ENV.fetch('DB_PORT') { 5432 } %>

test:

- adapter: postgresql
- encoding: unicode
- database: zona44_test

pool: 5

- username: <%= ENV.fetch('DB_USERNAME') { 'zona44' } %>
- password: <%= ENV.fetch('DB_PASSWORD') { 'password' } %>
- host: <%= ENV.fetch('DB_HOST') { 'localhost' } %>
- port: <%= ENV.fetch('DB_PORT') { 5432 } %>

production:

- adapter: postgresql
- encoding: unicode
- database: <%= ENV['DB_NAME'] %>
- pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
- username: <%= ENV['DB_USERNAME'] %>
- password: <%= ENV['DB_PASSWORD'] %>
- host: <%= ENV['DB_HOST'] %>
- port: <%= ENV['DB_PORT'] %>

Si usas SSL:

sslmode: require

5.2 MySQL (ejemplo YAML)

production:

- adapter: mysql2
- encoding: utf8mb4
- collation: utf8mb4_unicode_ci
- database: <%= ENV['DB_NAME'] %>
- username: <%= ENV['DB_USERNAME'] %>
- password: <%= ENV['DB_PASSWORD'] %>
- host: <%= ENV['DB_HOST'] %>
- port: <%= ENV['DB_PORT'] || 3306 %>
- pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>

6. **Variables de entorno y secretos**

Variables comunes:

- DB_HOST, DB_PORT, DB_USERNAME, DB_PASSWORD, DB_NAME
- DATABASE_URL (formato completo)
- RAILS_ENV
- RAILS_MASTER_KEY (si usas credenciales encriptadas)

Ejemplo de export (Linux):

- export DB_HOST=127.0.0.1
- export DB_PORT=5432
- export DB_USERNAME=zona44
- export DB_PASSWORD=secret
- export DB_NAME=zona44_production

7. **Docker / docker-compose**

Ejemplo `docker-compose.yml` con Postgres:

- version: '3.8'
- services:
- db:
- image: postgres:14
- restart: always
- environment:
- POSTGRES_USER: zona44
- POSTGRES_PASSWORD: secret
- POSTGRES_DB: zona44_development

volumes:

- db_data:/var/lib/postgresql/data
- ports:
  "5432:5432"

web:

build: .

command: bundle exec rails server -b 0.0.0.0

ports:

- "3000:3000"

environment:

- DB_HOST: db
- DB_USERNAME: zona44
- DB_PASSWORD: secret
- DB_NAME: zona44_development

depends_on:

- db

volumes:

db_data:

8. Migraciones, seeds y cargas iniciales

Comandos habituales:

Crear DB (dev/test):

- rails db:create
  - Ejecutar migraciones:
- rails db:migrate
  - Cargar seeds:
- rails db:seed
  - Resetear DB (usar con precaución):
- rails db:drop db:create db:migrate db:seed

9. Backups y restores

9.1 PostgreSQL

- Backup (dump completo):
- pg_dump -U <usuario> -h <host> -p <port> -F c -b -v -f zona44_backup.dump <db_name>
- # -F c crea un dump comprimido en formato custom

- Restore:

- pg_restore -U <usuario> -h <host> -p <port> -d <db_name> -v zona44_backup.dump

- Alternativa SQL:

- pg_dump -U user -h host -p 5432 -F p -f dump.sql dbname
- psql -U user -d dbname -f dump.sql

9.2 MySQL

- Backup:

mysqldump -u user -p -h host dbname > dump.sql

- Restore:

mysql -u user -p -h host dbname < dump.sql

Buenas prácticas:

1. Probar restauraciones periódicamente en entorno staging.
2. Mantener varios backups (rotación), conservar al menos 7-30 días según política.
3. Para Postgres, considerar `pg_basebackup` y WAL archiving para RPO mejor.

10. Pruebas de conexión y verificación

- Conexión con psql:

psql -h <host> -U <user> -d <db>

- Ver logs de Rails (log/development.log) para errores de conexión.
- Comando Rails para comprobar:

rails db:environment:set RAILS_ENV=development && rails db:create db:migrate

11. Seguridad y permisos

- No usar `postgres` o `root` para la app; crear usuario dedicado con mínimos permisos.
- Configurar TLS/SSL si la DB se accede por red pública.
- Configurar `pg_hba.conf` para limitar accesos por host y método.
  Cambiar contraseñas por defecto y usar gestores de secretos (Vault, AWS Secrets Manager, Azure KeyVault).
- Revocar permisos innecesarios en tablas sensibles.

12. Rendimiento y tuning básico

- Ajustar `pool` en `database.yml` acorde a `puma` y `RAILS_MAX_THREADS`.
- Parámetros Postgres a revisar: `shared_buffers`, `work_mem`, `maintenance_work_mem`, `effective_cache_size`.
- Indices: revisar y agregar índices para consultas frecuentes (WHERE, JOIN, ORDER BY).
- Considerar `pgbouncer` para pooling en entornos con muchas conexiones cortas.

13. Mantenimiento

- VACUUM ANALYZE regular (Postgres):
- VACUUM (VERBOSE, ANALYZE);
- Reindexar si índices fragmentados: `REINDEX TABLE <table>`
- Monitorizar crecimiento de tablas y logs WAL.

14. Automatización (CI/CD)

- En pipelines, usar variables seguras para credenciales.
- Ejemplo de pasos en CI:
- Crear DB (solo en test env)
- Ejecutar migraciones
- Ejecutar tests
- No ejecutar `db:seed` en CI salvo para datos necesarios de test.

15. Troubleshooting común

- Error de conexión: verificar host, puerto, credenciales, `pg_hba.conf` y que el servicio esté activo.
- Pool timeout: aumentar `pool` o reducir `RAILS_MAX_THREADS`/concurrencia de servidor.
- Problemas con migraciones: revisar `schema_migrations` y estado de migraciones pendientes.

16. Anexos: comandos útiles

- Ver procesos en Postgres: `sudo systemctl status postgresql`
- Logs Postgres (Ubuntu): `/var/log/postgresql/postgresql-<version>-main.log`
- Crear usuario Postgres:

sudo -u postgres createuser --pwprompt zona44

sudo -u postgres createdb -O zona44 zona44_development

17. Pasos recomendados para despliegue inicial

1. Instalar Postgres (o crear instancia en proveedor managed).
2. Crear DB y usuario con permisos mínimos.
3. Exportar variables de entorno en servidor (DB_HOST, DB_USERNAME, DB_PASSWORD, DB_NAME).
4. Ejecutar `bundle install`, `rails db:create`, `rails db:migrate`, `rails db:seed`.
5. Configurar backups automáticos y monitoreo.
6. Validar funcionamiento mediante pruebas de integración.