

DÍA 11: INSTALACIÓN Y ADMINISTRACIÓN DE SERVIDORES DE BASES DE DATOS

Por Daniel Ariza

27/06/2025

Fase 1: Instalación y configuración del servidor de bases de datos

- **Elegir entre MySQL/MariaDB o PostgreSQL según los requerimientos de Codearts Solutions.**

Hemos elegido PostgreSQL principalmente por estos motivos:

1. **Seguridad robusta por defecto:** tiene un enfoque serio en la seguridad y es compatible con autenticación por contraseña, certificados SSH y mecanismos como peer o md5.
2. **Ideal para datos relacionales complejos:** Si los proyectos internos tienen relaciones entre entidades (usuarios → proyectos → tareas → archivos), PostgreSQL maneja esto mejor que otros motores. Soporta llaves foráneas, **constraints**, **triggers**, y **transacciones ACID completas**.
3. **Madurez y estabilidad:** Lleva más de 25 años de desarrollo activo. Es usado por empresas como Apple, Red Hat, y Spotify: probado en entornos grandes y críticos.

4. **Buen rendimiento en Ubuntu Server:** La documentación oficial es clara y extensa. Hay gran cantidad de tutoriales, foros y soluciones a errores comunes.
- **Instalar el motor de bases de datos en un servidor Linux (apt install mysql-server / apt install postgresql).**
`sudo apt install postgresql postgresql-contrib -y` Esto instalará Postgresql y algunas herramientas adicionales que pueden ser útiles como createdb, createuser, psql, etc.
 - **Verificar que el servicio está activo (systemctl status mysql / systemctl status postgresql).**
`sudo systemctl status postgresql` con esto verificamos que el servicio está corriendo.

```
daniel@ubuntucodearts:~$ sudo systemctl status postgresql
* postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
   Active: active (exited) since vie 2025-06-27 10:33:35 CEST; 5min ago
     Main PID: 3751 (code=exited, status=0/SUCCESS)
    CGroup: /system.slice/postgresql.service

jun 27 10:33:35 ubuntucodearts systemd[1]: Starting PostgreSQL RDBMS...
jun 27 10:33:35 ubuntucodearts systemd[1]: Started PostgreSQL RDBMS.
jun 27 10:34:33 ubuntucodearts systemd[1]: Started PostgreSQL RDBMS.
daniel@ubuntucodearts:~$ _
```

La palabra active lo confirma.

- **Configurar el servicio para iniciarse automáticamente al encender el servidor.**

`sudo systemctl enable postgresql` con esto ejecutamos la orden de iniciarse automáticamente al encender el servidor.

```
daniel@ubuntucodearts:~$ sudo systemctl enable postgresql
Synchronizing state of postgresql.service with SysV init with /lib/systemd/systemd-sysv-install...
Executing /lib/systemd/systemd-sysv-install enable postgresql
daniel@ubuntucodearts:~$ _

daniel@ubuntucodearts:~$ sudo systemctl is-enabled postgresql
enabled
daniel@ubuntucodearts:~$
```

La palabra enabled nos lo confirma.

Fase 2: Creación y gestión de bases de datos y usuarios

- **Crear una base de datos llamada codearts_db .**

`sudo -u postgres createdb codearts_db` con este comando crearemos la base de datos y con este otro `psql -U codearts_user -d codearts_db -h localhost` nos conectaremos a ella para confirmar que se ha creado correctamente.

```
codearts_db=> \q
daniel@ubuntucodearts:~$ psql -U codearts_daniel -d codearts_db -h localhost
Password for user codearts_daniel:
psql (9.5.25)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.

codearts_db=>
```

Aquí vemos que ya nos hemos conectado a ella con el usuario codearts_daniel que hemos creado.

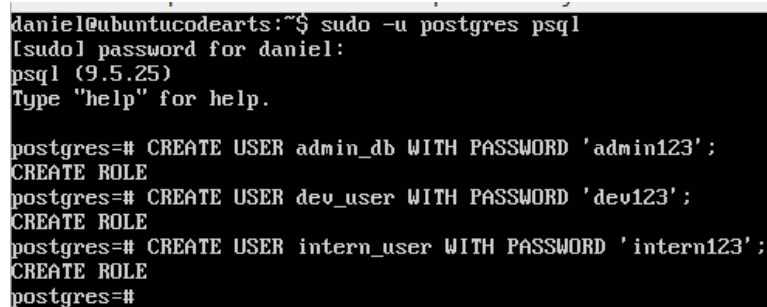
- **Crear tres usuarios con diferentes niveles de permisos:**
 - **admin_db** (Acceso total a la base de datos).
 - **dev_user** (Acceso solo lectura y escritura).
 - **intern_user** (Acceso solo de lectura).

Primero entramos a PostgreSQL como administrador con **sudo -u postgres psql** y creamos los tres usuarios escribiendo estas líneas dentro del prompt:

CREATE USER admin_db WITH PASSWORD 'admin123';

CREATE USER dev_user WITH PASSWORD 'dev123';

CREATE USER intern_user WITH PASSWORD 'intern123';



```
daniel@ubuntu:~$ sudo -u postgres psql
[sudo] password for daniel:
psql (9.5.25)
Type "help" for help.

postgres=# CREATE USER admin_db WITH PASSWORD 'admin123';
CREATE ROLE
postgres=# CREATE USER dev_user WITH PASSWORD 'dev123';
CREATE ROLE
postgres=# CREATE USER intern_user WITH PASSWORD 'intern123';
CREATE ROLE
postgres=#
```

Con este comando **GRANT ALL PRIVILEGES ON DATABASE codearts_db TO admin_db;** le daremos permisos de total acceso al usuario admin_db.

Con este otro **GRANT CONNECT ON DATABASE codearts_db TO dev_user;** otorgaremos conexión al usuario dev_user

```
postgres=# GRANT ALL PRIVILEGES ON DATABASE codearts_db TO admin_db;
GRANT
postgres=# GRANT CONNECT ON DATABASE codearts_db TO dev_user
postgres=# _
```

Ahora entramos a la base de datos para asignar permisos a nivel de tabla con **\c codearts_db** Y una vez dentro ejecutamos estas líneas: para dar permisos de lectura y escritura a dev_user:

GRANT USAGE ON SCHEMA public TO dev_user;

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO dev_user;

ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO dev_user;

```
postgres=# GRANT CONNECT ON DATABASE codearts_db TO dev_user
postgres=# \c codearts_db
You are now connected to database "codearts_db" as user "postgres".
codearts_db=# GRANT USAGE ON SCHEMA PUBLIC TO dev_user;
ERROR:  syntax error at or near "GRANT"
LINE 2: GRANT USAGE ON SCHEMA PUBLIC TO dev_user;
          ^
codearts_db=# GRANT USAGE ON SCHEMA public TO dev_user;
GRANT
codearts_db=# GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO dev_user;
GRANT
codearts_db=# ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TA
ES TO dev_user;
ALTER DEFAULT PRIVILEGES
codearts_db=# _
```

Y ahora otorgamos conexión al usuario intern_user con
`GRANT CONNECT ON DATABASE codearts_db TO
intern_user;`

Accedemos a la base de datos con `\c codearts_db`
y damos permisos de solo lectura al usuario intern_user
con:

`GRANT USAGE ON SCHEMA public TO intern_user;`
`GRANT SELECT ON ALL TABLES IN SCHEMA public TO
intern_user;`
`ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT
SELECT ON TABLES TO intern_user;`

Y con esto nos aseguramos que tenga también acceso de
lectura en tablas futuras: `ALTER DEFAULT PRIVILEGES IN
SCHEMA public GRANT SELECT ON TABLES TO
intern_user;`

```
daniel@ubuntu:~$ sudo -u postgres psql
[sudo] password for daniel:
psql (9.5.25)
Type "help" for help.

postgres=# GRANT CONNECT ON DATABASE codearts_db TO intern_user;
GRANT
postgres=# GRANT USAGE ON SCHEMA public TO intern_user;
GRANT
postgres=# GRANT SELECT ON ALL TABLES IN SCHEMA public TO intern_user;
GRANT
postgres=# ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON TABLES TO intern_user;
ALTER DEFAULT PRIVILEGES
postgres=# _
```

- **Asignar permisos específicos a cada usuario (GRAN PRIVILEGIOS).** Esto ya lo hemos realizado en el punto anterior.

Fase 3: Conexión de una aplicación web a la base de datos

- **Instalar un servidor web con Apache/Nginx y PHP/Python.**

Instalaremos primero PHP y el módulo para apache con estos comandos:

```
sudo apt update
```

```
sudo apt install php libapache2-mod-php
```

libapache2-mod-php, conecta Apache con PHP, permitiendo que Apache procese archivos .php.

```
daniel@ubuntucodearts:~$ sudo apt update
[sudo] password for daniel:
Des:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [106 kB]
Obj:2 http://gb.archive.ubuntu.com/ubuntu xenial InRelease
Des:3 http://gb.archive.ubuntu.com/ubuntu xenial-updates InRelease [106 kB]
Des:4 http://gb.archive.ubuntu.com/ubuntu xenial-backports InRelease [106 kB]
Descargados 317 kB en 0s (358 kB/s)
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Todos los paquetes están actualizados.
daniel@ubuntucodearts:~$ sudo apt install php libapache2-mod-php
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  libapache2-mod-php7.0 php-common php7.0 php7.0-cli php7.0-common php7.0-json php7.0-opcache
  php7.0-readline
Paquetes sugeridos:
  php-pear
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-php libapache2-mod-php7.0 php php-common php7.0 php7.0-cli php7.0-common
  php7.0-json php7.0-opcache php7.0-readline
0 actualizados, 10 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 3.478 kB de archivos.
Se utilizarán 13,9 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

Ahora con este comando `echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php` creamos un archivo PHP que muestra la configuración actual del servidor PHP.

Y con este otro comando `sudo apt install python3 python3-pip libapache2-mod-wsgi-py3` habilitamos python en apache (mod_wsgi). Esto permite que Apache ejecute aplicaciones Python que siguen el estándar **WSGI** (como Flask o Django).

```
daniel@ubuntucodearts:~$ echo "<? phpinfo(); ?>" | sudo tee /var/www/html/info.php
[sudo] password for daniel:
<? phpinfo(); ?>
daniel@ubuntucodearts:~$ sudo apt install python3 python3-pip libapache2-mod-wsgi-py3
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
python3 ya está en su versión más reciente (3.5.1-3).
Se instalarán los siguientes paquetes adicionales:
  libexpat1-dev libpython3-dev libpython3.5 libpython3.5-dev python-pip-whl python3-dev
  python3-setuptools python3-wheel python3.5-dev
Paquetes sugeridos:
  python-setuptools-doc
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-wsgi-py3 libexpat1-dev libpython3-dev libpython3.5 libpython3.5-dev
  python-pip-whl python3-dev python3-pip python3-setuptools python3-wheel python3.5-dev
0 actualizados, 11 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 40,7 MB de archivos.
Se utilizarán 62,2 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] _
```

- **Crear un script de conexión que permita leer y escribir datos en la base de datos.**

Primero desde psql crearemos una tabla de prueba con estas líneas `CREATE TABLE empleados (`

`id SERIAL PRIMARY KEY,`
`nombre VARCHAR(100),`
`puesto VARCHAR(100));`


```
daniel@ubuntucodearts:~$ sudo -u postgres psql
psql (9.5.25)
Type "help" for help.

postgres=# CREATE TABLE empleados ( id SERIAL PRIMARY KEY, nombre VARCHAR(100), puesto VARCHAR(100)
;
CREATE TABLE
postgres=#
```

En segundo lugar instalaremos la extensión PHP para psql
con: **sudo apt update sudo apt install php-pgsql**

```
daniel@ubuntucodearts:~$ sudo apt update
Obj:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
Obj:2 http://gb.archive.ubuntu.com/ubuntu xenial InRelease
Obj:3 http://gb.archive.ubuntu.com/ubuntu xenial-updates InRelease
Obj:4 http://gb.archive.ubuntu.com/ubuntu xenial-backports InRelease
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Todos los paquetes están actualizados.
daniel@ubuntucodearts:~$ sudo apt install php-pgsql
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
  php7.0-pgsql
Se instalarán los siguientes paquetes NUEVOS:
  php-pgsql php7.0-pgsql
0 actualizados, 2 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 57,9 kB de archivos.
Se utilizarán 227 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

PHP Version 7.0.33-0ubuntu0.16.04.16

Inglés

Google Translate

System	Linux ubuntucodearts 4.4.0-210-generic #242-Ubuntu SMP Fri Apr 16 09:57:56 UTC 2021 x86_64
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-exif.ini, /etc/php/7.0/apache2/conf.d/20-fileinfo.ini, /etc/php/7.0/apache2/conf.d/20-ftp.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-pdo_pgsql.ini, /etc/php/7.0/apache2/conf.d/20-pgsql.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, phar
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, convert.iconv.*

La captura confirma que PHP está funcionando

perfectamente desde Apache en mi servidor. Metiendo esta dirección <http://192.168.1.40/info.php> desde el navegador del equipo anfitrión.

Y ahora crearemos el script creando el archivo con **sudo nano /var/www/html/db_php.php**

```
if (!$conn) {
die("Error al conectar: " . pg_last_error());
}
echo "Conexion exitosa.<br>";

// 2. Insertar un registro
$query_insert = "INSERT INTO empleados (nombre, puesto) VALUES ('Daniel', 'Desarrollador')";
$result_insert = pg_query($conn, $query_insert);

if ($result_insert) {
echo "Registro insertado correctamente.<br>";
} else {
echo "Error al insertar: " . pg_last_error() . "<br>";
}

// 3. Leer los registros
$query_select = "SELECT * FROM empleados";
$result_select = pg_query($conn, $query_select);

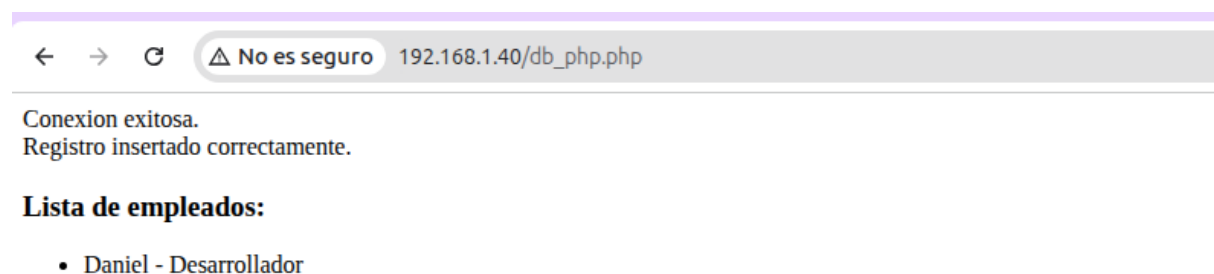
if (!$result_select) {
die("Error al leer datos: " . pg_last_error());
}

echo "<h3> Lista de empleados:</h3><ul>";
while ($row = pg_fetch_assoc($result_select)) {
echo "<li>" . htmlspecialchars($row['nombre']) . " - " . htmlspecialchars($row['puesto']) . "</li>";
}
echo "</ul>";

// 4. Cerrar Conexion
pg_close($conn);
?>

daniel@ubuntucodearts:~$
```

Nos vamos al navegador de la máquina anfitrión e introducimos la dirección ip del servidor y el nombre del archivo que contiene el script.



Esto confirma que el script funciona correctamente.

- **Realizar pruebas con consultas SELECCIONAR, INSERTAR, ACTUALIZAR y BORRAR desde la aplicación web.**

Añadimos este bloque de script en el archivo del script anterior: `if ($op === 'insert') {`

```
    echo "<p> Insertando nuevo empleado...</p>";

    $query_insert = "INSERT INTO empleados (nombre, puesto) VALUES
('Ana', 'Diseñadora)";

    $result_insert = pg_query($conn, $query_insert);

    if ($result_insert) {

        echo " Empleado insertado correctamente.<br>";

    } else {

        echo " Error al insertar: " . pg_last_error($conn) . "<br>";

    }

}

if ($op === 'update') {

    echo "<p> Actualizando puesto de Daniel...</p>";

    $query_update = "UPDATE empleados SET puesto = 'Jefe de proyecto'
WHERE nombre = 'Daniel'";

    $result_update = pg_query($conn, $query_update);

    if ($result_update) {

        echo " Puesto actualizado correctamente.<br>";

    } else {

        echo " Error al actualizar: " . pg_last_error($conn) . "<br>";

    }

}
```

```

}

if ($op === 'delete') {
    echo "<p> Eliminando a Ana...</p>";
    $query_delete = "DELETE FROM empleados WHERE nombre = 'Ana'";
    $result_delete = pg_query($conn, $query_delete);
    if ($result_delete) {
        echo " Empleado eliminado correctamente.<br>";
    } else {
        echo " Error al eliminar: " . pg_last_error($conn) . "<br>";
    }
}
}

```

E incluimos esta línea: `$op = $_GET['op'] ?? 'select';` justo después de donde tenemos esta otra: `echo "Conexion exitosa.
";` y para probarlo metemos esto en el navegador:

http://192.168.1.40/db_php.php?op=insert

`http://192.168.1.40/db_php.php?op=update`

`http://192.168.1.40/db_php.php?op=delete`

`http://192.168.1.40/db_php.php` (para solo ver)

Fase 4: Seguridad y respaldo de datos

- **Configurar accesos seguros restringiendo conexiones remotas (bind-address).**

Primero abrimos el archivo `sudo nano /etc/postgresql/*/main/postgresql.conf` y buscamos la línea `#listen_addresses = 'localhost'`

```
# - Connection Settings -  
listen_addresses = 'localhost'          # what IP address(es) to listen on;  
                                         # comma-separated list of addresses;
```

La descomentamos y solo permitiremos conexiones locales.

Ahora abrimos el archivo `sudo nano /etc/postgresql/*/main/pg_hba.conf` Este archivo determina **qué usuarios pueden conectarse desde qué IPs**.

Buscamos las líneas de la siguiente captura

```
# replication privilege.  
#local   replication    postgres                    peer  
#host    replication    postgres    192.168.1.47/24          md5  
#host    replication    postgres    ::1/128                 md5
```

Y yo en mi caso he permitido solo la conexión de mi máquina anfitriona, modificando solo la línea del medio. Si no queremos permitir más conexiones externas, no modificamos más líneas.

- **Implementar autenticación segura con políticas de contraseñas (mysql_secure_installation).**

Entramos a psql como superusuarios con `sudo -u postgres psql` y luego para cada usuario aplicamos con `ALTER USER codearts_user WITH PASSWORD 'UnaContraseñaMuySegura123!'`; Después aplicaremos estas líneas para crear una política de longitud mínima de contraseñas.

```
CREATE OR REPLACE FUNCTION  
enforce_password_policy(pwd TEXT) RETURNS BOOLEAN  
AS $$
```

```
BEGIN
```

```
    IF length(pwd) < 12 THEN
```

```
        RAISE EXCEPTION 'La contraseña debe tener al menos  
12 caracteres.';
```

```
    END IF;
```

```
    RETURN TRUE;
```

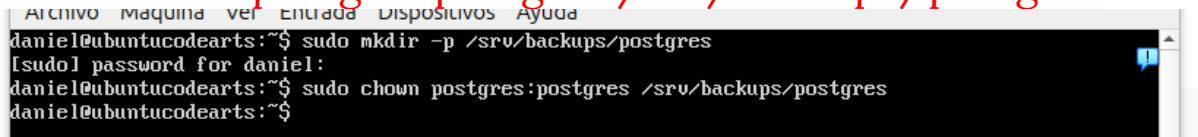
```
END;
```

```
$$ LANGUAGE plpgsql;
```

- **Crear un sistema de copias de seguridad automáticas (mysqldump o pg_dump).**

En primer lugar crearemos un usuario y carpeta de backups con `sudo mkdir -p /srv/backups/postgres`

`sudo chown postgres:postgres /srv/backups/postgres`

A terminal window with a dark background and light text. The prompt is 'daniel@ubuntucodearts:~\$'. The first command is 'sudo mkdir -p /srv/backups/postgres', followed by a password prompt '[sudo] password for daniel:'. The second command is 'sudo chown postgres:postgres /srv/backups/postgres', followed by another password prompt. The prompt returns to 'daniel@ubuntucodearts:~\$'.

```
daniel@ubuntucodearts:~$ sudo mkdir -p /srv/backups/postgres
[sudo] password for daniel:
daniel@ubuntucodearts:~$ sudo chown postgres:postgres /srv/backups/postgres
daniel@ubuntucodearts:~$
```

En segundo lugar crearemos el script de copia con

`sudo -u postgres nano /usr/local/bin/pg_backup.sh`

E incluimos estas líneas:

`#!/bin/bash`

`# Nombre del archivo con fecha`

`FECHA=$(date +%F_%H-%M)`

`DESTINO="/srv/backups/postgres/backup_$(FECHA).sql"`

`# Dump completo de todas las bases de datos`

`pg_dumpall -U postgres > "$DESTINO"`

`# Opcional: eliminar copias de más de 7 días`

`find /srv/backups/postgres -type f -name "*.sql" -mtime +7
-exec rm {} \;`


```
Archivo  Maquina  Ver  Entrada  Dispositivos  Ayuda
GNU nano 2.5.3      Archivo: /usr/local/bin/pg_backup.sh  Modificado
#!/bin/bash

# Nombre del archivo con fecha
FECHA=$(date +%F_%H-%M)
DESTINO="/srv/backups/postgres/backup_$FECHA.sql"

#Dump completo de todas las bases de datos
pg_dumpall -U postgres > "$DESTINO"

#Opcional: eliminar copias de mas de 7 dias
find /srv/backups/postgres -type f -name "*.sql" -mtime +7 -exec rm {} \;
```

Le otorgamos permisos de ejecución con:

`sudo chmod +x /usr/local/bin/pg_backup.sh`

- Registrar intentos de acceso fallidos y auditoría de consultas con logs del sistema (/var/log/mysql.log, /var/log/postgresql.log).

Paso 1: Verificar la ruta de los logs de PostgreSQL

En Ubuntu, PostgreSQL suele usar rsyslog o su propio log rotado en:

`/var/log/postgresql/postgresql-XX-main.log`

Paso 2: Habilitar el logueo de conexiones y consultas

Edita el archivo principal:

`sudo nano /etc/postgresql/*/main/postgresql.conf`

Busca y asegúrate de tener (descomenta si hace falta):

```
logging_collector = on
```

```
log_directory = 'log'
```

```
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
```

```
log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d,app= %a,client=%h '
```

```
log_connections = on
```

```
log_disconnections = on
```

```
log_statement = 'none'      # cambia esto si quieres registrar  
consultas (ver más abajo)
```

```
log_duration = off
```

```
log_timezone = 'Europe/Madrid'
```

Para registrar accesos fallidos y exitosos:

Asegúrate de tener:

```
log_connections = on
```

```
log_disconnections = on
```

Esto registrará los accesos exitosos y los intentos con usuario incorrecto.