PostgreSQL Backup & Restore with Google Drive (rclone)

This document describes the **complete, production-safe process** to back up a PostgreSQL database from a **remote / free-tier DB instance** to **Google Drive** using **rclone**, and how to **restore it correctly** on a new server or Docker container.

It also documents **common mistakes**, **why they happened**, and the **correct commands**.

---

1. Architecture Overview

- **Source**: Remote PostgreSQL DB-only instance
- **Backup type**: Logical backup (`pg_dump -Fc`)
- **Storage**: Google Drive (via rclone)
- **Automation**: cron (daily)
- **Restore targets**:
- New VM
- Docker PostgreSQL container

Why logical backups? - Portable across OS, VM, Docker - Small size - Easy to restore partially or fully

---

2. Prerequisites

On DB Server

- PostgreSQL installed and running
- `pg_dump` available
- Internet access

On Local Machine (one-time)

- Google account
- Google Cloud Console access

---

3. Google Drive Setup (Headless / Remote Server)

3.1 Create Google API Credentials

1. Open Google Cloud Console
2. Create a new project (or reuse one)
3. Enable **Google Drive API**
4. Create **OAuth Client ID**
5. Type: **Desktop App**
6. Save:

7. `CLIENT_ID`
8. `CLIENT_SECRET`

---

4. Install & Configure rclone

4.1 Install rclone

```
sudo apt update
sudo apt install -y rclone
```

4.2 Configure rclone (IMPORTANT: Headless Mode)

```
rclone config
```

Selections:

```
n) New remote
name> gdrive
Storage> drive
client_id> YOUR_CLIENT_ID
client_secret> YOUR_CLIENT_SECRET
scope> 1
root_folder_id> (Enter)
service_account_file> (Enter)
Edit advanced config?> n
Use auto config?> n   # VERY IMPORTANT
```

At this point **rclone may behave in one of two valid ways** depending on version:

Flow A (URL shown directly on server)

- rclone prints a long Google OAuth URL
- You open that URL on your local browser
- Approve access
- Google gives a verification code
- Paste the code back into the server terminal

Flow B (Local-machine command — what we actually used)

- rclone prints **a command to run on your local machine**, for example:

```
rclone authorize "drive" "CLIENT_ID" "CLIENT_SECRET"
```

Steps: 1. Run that command **on your local laptop/desktop** 2. A browser window opens automatically 3. Login to Google and approve access 4. The local CLI outputs a **JSON object containing tokens** 5. Copy that full JSON output 6. Paste it back into the **server rclone prompt** when asked

This method is **official, secure, and recommended** for fully headless servers.

After either flow completes, verify:

```
rclone lsd gdrive:
```

4.3 Create Backup Folder on Drive

```
rclone mkdir gdrive:postgres-backups
```

---

5. Prepare Local Backup Directory (DB Server)

```
sudo mkdir -p /var/backups/postgres
sudo chown postgres:postgres /var/backups/postgres
```

---

6. Backup Script (Correct & Final)

Create script:

```
sudo -u postgres nano /var/backups/postgres/daily_backup.sh
```

```
#!/bin/bash
set -euo pipefail
cd /

DB_NAME="app_prod"
BACKUP_DIR="/var/backups/postgres"
DATE=$(date +%F)
FILE="${BACKUP_DIR}/${DB_NAME}_${DATE}.dump"

# Create logical backup
pg_dump -Fc "$DB_NAME" > "$FILE"
```

```
# Upload to Google Drive
rclone copy "$FILE" gdrive:postgres-backups --quiet

# Keep only last 14 days locally
find "$BACKUP_DIR" -type f -name "*.dump" -mtime +14 -delete
```

```
sudo chmod +x /var/backups/postgres/daily_backup.sh
```

---

7. rclone Access for postgres User (CRITICAL)

```
sudo mkdir -p /var/lib/postgresql/.config/rclone
sudo cp ~/.config/rclone/rclone.conf /var/lib/postgresql/.config/rclone/
sudo chown -R postgres:postgres /var/lib/postgresql/.config
```

Test:

```
sudo -u postgres rclone lsd gdrive:
```

---

8. Manual Test (MUST DO)

```
sudo -u postgres /var/backups/postgres/daily_backup.sh
```

Verify:

```
ls -lh /var/backups/postgres
rclone ls gdrive:postgres-backups
```

---

9. Automate with cron

```
sudo -u postgres crontab -e
```

```
0 2 * * * /var/backups/postgres/daily_backup.sh >> /var/backups/postgres/
backup.log 2>&1
```

---

10. Restore Process (Docker PostgreSQL)

10.1 Copy Dump into Container

```
docker cp app_prod_2026-02-07.dump postgres:/tmp/app_prod.dump
```

10.2 Use the CORRECT User

Docker PostgreSQL **does NOT always have** `postgres` **role**. Use the user defined by `POSTGRES_USER`.

Example:

```
docker exec -it postgres psql -U gasops
```

10.3 Drop & Recreate Target DB (Recommended)

```
docker exec -it postgres psql -U gasops -d postgres -c "DROP DATABASE
app_prod_restore;"
docker exec -it postgres psql -U gasops -d postgres -c "CREATE DATABASE
app_prod_restore;"
```

10.4 Restore Command (Correct)

```
docker exec -i postgres pg_restore
  -U gasops
  -d app_prod_restore
  --no-owner
  --no-privileges
  < app_prod_2026-02-07.dump
```

---

11. Verification After Restore

```
\dn
\dt auth.*
\dt public.*
SELECT COUNT(*) FROM public.songs;
SELECT COUNT(*) FROM auth.users;
```

12. Mistakes Encountered & Corrections

❌ Mistake 1: Using auto-config on remote server

- **Problem**: No browser available
- **Fix**: Use manual OAuth (`Use auto config? n`)

❌ Mistake 2: Missing backup directory

- **Problem**: nano could not save file
- **Fix**:

```
mkdir -p /var/backups/postgres
```

❌ Mistake 3: `find` permission warning

- **Cause**: postgres user cannot access `/home/ubuntu`
- **Fix**:

```
cd /
```

❌ Mistake 4: Assuming `postgres` role exists in Docker

- **Reality**: Docker creates ONLY `POSTGRES_USER`
- **Fix**: Always restore using that user

❌ Mistake 5: Schema already exists during restore

- **Cause**: Restoring into non-empty DB
- **Fix**: Drop & recreate DB before restore

13. Best Practices (Final)

- Always test restore
- Always store backups **outside the VM**
- Always use `--no-owner --no-privileges` across environments
- Prefer logical backups for small/medium DBs
- Keep backups boring and predictable

14. Outcome

This setup provides: - Daily automated backups - Off-server safety - Zero cost - Cross-environment restore - Verified disaster recovery

This is **production-grade**, even on free tier.