Database Backup & Restore Scripts

A small, cross-platform toolkit for **full logical backups** of a PostgreSQL database using the native tools (`pg_dump` / `psql`).
Works on **macOS/Linux** (Bash) and **Windows** (Batch + PowerShell). Supports **Docker auto-detection**, **optional gzip**, and **simple rotation**.

## At a glance

- Scripts read **`.env` in your current project directory** to discover connection info.
- They **prefer Docker** if your PostgreSQL is running in a container; otherwise they use local client tools.
- If the selected DB container **doesn’t include client tools**, the scripts run an **ephemeral `postgres:16` client** for you.
- Backups are **plain SQL** (`.sql`) and optionally **gzipped** (`.sql.gz`).
- File naming: `<database>_<suffix>.sql[.gz]` in a target **backup directory** (default `./backups`).

## Repo layout (suggested)

```
project/
■■ .env
■■ backups/ # default output directory (created if missing)
■■ scripts/
■■ db/
■■ managedb.sh # macOS/Linux engine (auto-detects Docker/local/ephemeral)
■■ backupdb.sh # minimal backup wrapper (.sql)
■■ restoredb.sh # minimal restore wrapper (.sql)
■■ backupdbzip.sh # backup then gzip (.sql.gz)
■■ restoredbzip.sh # gunzip then restore
■■ backup_rotate.sh # timestamped .sql + prune
■■ backupdbzip_rotate.sh # timestamped .sql.gz + prune
■■ managedb.bat # Windows wrapper (calls managedb.ps1)
■■ managedb.ps1 # Windows engine
■■ backupdb.bat # minimal backup wrapper (.sql)
■■ restoredb.bat # minimal restore wrapper (.sql)
■■ backupdbzip.bat # backup then gzip (.sql.gz)
■■ restoredbzip.bat # gunzip then restore
■■ backup_rotate.bat # timestamped .sql + prune
■■ backupdbzip_rotate.bat # timestamped .sql.gz + prune
```

> The Windows pieces (`*.bat` wrappers + `managedb.ps1`) mirror the functionality of the shell scripts.

## `.env` configuration

The scripts expect **one** of the following:

### 1) A single `DATABASE_URL` (recommended)

```env
DATABASE_URL="postgresql://PGUSER:PGPASSWORD@PGHOST:5432/PGDATABASE?schema=public"
```

The engine parses this URL and exports `PGHOST`, `PGPORT`, `PGUSER`, `PGPASSWORD`, `PGDATABASE`.

### 2) Or explicit `PG*` variables

```env
PGHOST=localhost
PGPORT=5432
PGUSER=postgres
PGPASSWORD=postgres
PGDATABASE=myapp
```

> The `.env` file is read **from your current working directory** (i.e., run commands from your `project/` folder so the scripts pick up the correct `.env`).

## How the engine chooses where to run `pg_dump`/`psql`

1. **Running Postgres container?**
The engine scans `docker ps` for a container whose **name or image** includes “postgres” (or common variants), **preferring** the one that **publishes your `PGPORT`**.
2. **If that container has client tools** (`pg_dump`, `psql`) → use `docker exec`.
3. **If it does not** → run an **ephemeral `postgres:16` client** attached to that container’s network (`--network container:<id>`) and connect to `127.0.0.1`.
4. **If no container fits**:
- If local `pg_dump` exists → use **local** client tools.
- Else → use an **ephemeral `postgres:16` client** to your host (macOS alias `host.docker.internal` is handled automatically).

Optional override: set `POSTGRES_CONTAINER=<container-name>` to force a specific container (helpful if multiple are running).

## Script inventory & usage

### macOS / Linux

> Make scripts executable once:
```bash
chmod +x ./scripts/db/*.sh
```

Script	Purpose	Usage	Notes
`managedb.sh`	Core backup/restore engine	`./scripts/db/managedb.sh <backup|restore> [suffix=current] [dir=./backups]`	You normally won’t call this directly; use the small wrappers below.
`backupdb.sh`	Minimal backup (plain SQL)	`../scripts/db/backupdb.sh [suffix] [dir]`	Default suffix `current`, default dir `./backups`. Overwrites existing file.
`restoredb.sh`	Minimal restore (plain SQL)	`../scripts/db/restoredb.sh [suffix] [dir]`	Restores `<db>_<suffix>.sql`.
`backupdbzip.sh`	Backup then gzip	`../scripts/db/backupdbzip.sh [suffix] [dir]`	Produces `<db>_<suffix>.sql.gz`.
`restoredbzip.sh`	Gunzip then restore	`../scripts/db/restoredbzip.sh [suffix] [dir]`	Reads `<db>_<suffix>.sql.gz`, inflates to `.sql`, runs restore, cleans up `.sql`.
`backup_rotate.sh`	Timestamped backup + prune	`../scripts/db/backup_rotate.sh [retention_days=7] [dir]`	Creates `<db>_YYYYMMDD_HHMMSS.sql`, deletes `.sql` older than `retention_days`.
`backupdbzip_rotate.sh`	Timestamped gz backup + prune	`../scripts/db/backupdbzip_rotate.sh [retention_days=7] [dir]`	Creates `<db>_YYYYMMDD_HHMMSS.sql.gz`, prunes old `.sql.gz`.

Common examples (run from your `project/` directory):
```bash
# Minimal backup/restore (plain .sql)
../scripts/db/backupdb.sh
../scripts/db/restoredb.sh

# Name the backup
../scripts/db/backupdb.sh nightly
../scripts/db/restoredb.sh nightly

# Write to a specific directory
../scripts/db/backupdb.sh nightly ./backups

# Gzipped backup/restore
../scripts/db/backupdbzip.sh nightly
../scripts/db/restoredbzip.sh nightly

# Rotation (keep last 7 days by default)
../scripts/db/backup_rotate.sh
../scripts/db/backupdbzip_rotate.sh

# Rotation with custom retention and dir
../scripts/db/backup_rotate.sh 14 ./backups
../scripts/db/backupdbzip_rotate.sh 21 ./backups
```

Optional diagnostics:
```bash
DEBUG=1 ../scripts/db/backupdb.sh nightly
```

### Windows

> The `.bat` wrappers call into `managedb.ps1` through `managedb.bat`. If PowerShell script execution is restricted, you may need to allow it (for the current process only):
> `powershell -NoProfile -ExecutionPolicy Bypass -File scripts\db\managedb.ps1 ...` (handled by the wrapper).

Script	Purpose	Usage	Notes
`managedb.bat` + `managedb.ps1`	Core engine (Windows)	`managedb.bat <backup|restore> [suffix] [dir]`	Normally invoked by the wrappers below.
`backupdb.bat`	Minimal backup (plain SQL)	`..\scripts\db\backupdb.bat [suffix] [dir]`	Overwrites existing file.
`restoredb.bat`	Minimal restore (plain SQL)	`..\scripts\db\restoredb.bat [suffix] [dir]`	Restores `<db>_<suffix>.sql`.
`backupdbzip.bat`	Backup then gzip via PowerShell	`..\scripts\db\backupdbzip.bat [suffix] [dir]`	Produces `<db>_<suffix>.sql.gz`.
`restoredbzip.bat`	Gunzip then restore	`..\scripts\db\restoredbzip.bat [suffix] [dir]`	Decompresses then restores; cleans up `.sql`.
`backup_rotate.bat`	Timestamped backup + prune	`..\scripts\db\backup_rotate.bat [retention_days] [dir]`	Creates `<db>_YYYYMMDD_HHMMSS.sql`, prunes old `.sql`.
`backupdbzip_rotate.bat`	Timestamped gz backup + prune	`..\scripts\db\backupdbzip_rotate.bat [retention_days] [dir]`	Creates `<db>_YYYYMMDD_HHMMSS.sql.gz`, prunes old `.sql.gz`.

Examples (run from your `project\` folder):
```bat
..\scripts\db\backupdb.bat
..\scripts\db\restoredb.bat current .\backups

..\scripts\db\backupdbzip.bat nightly .\backups
..\scripts\db\restoredbzip.bat nightly .\backups

..\scripts\db\backup_rotate.bat 14 .\backups
..\scripts\db\backupdbzip_rotate.bat 21 .\backups
```

## File naming & overwrite behavior

- Base name: **`<PGDATABASE>_<suffix>`**; default suffix `current`.
- Minimal backup writes **`<db>_current.sql`** and **overwrites** the file if it exists.
- Zip wrappers write **`<db>_<suffix>.sql.gz`**.
- Rotation scripts stamp names as **`<db>_YYYYMMDD_HHMMSS.sql[.gz]`** and **prune** older files by modification time.

## Environment variables you can set

- `DATABASE_URL` – full Postgres URL (preferred if you don’t set individual vars)
- `PGHOST`, `PGPORT`, `PGUSER`, `PGPASSWORD`, `PGDATABASE` – individual connection fields

- `POSTGRES_CONTAINER` – (optional) force a specific container name/ID
- `DEBUG=1` – verbose diagnostics (shows detection path, versions, `\conninfo`, absolute output path)

## Requirements

- **Docker** (optional) – for auto-detection and ephemeral client fallback
- **PostgreSQL client tools** (optional) – `pg_dump`, `psql` in your PATH if you’re not using Docker
- macOS: `brew install libpq` and add `libpq/bin` to `PATH`
- **Windows**: PowerShell 5+ (built-in) is used for the engine (`managedb.ps1`) and gzip

---

## Troubleshooting

No file / zero-byte file
- Run with `DEBUG=1` to see error output.
- Common causes:
- Wrong credentials or DB name → check `.env` content.
- Local `pg_dump` missing and Docker not running → install client tools or start Docker.
- Multiple Postgres containers and the wrong one is picked → set `POSTGRES_CONTAINER=<name>`.

“pg_dump: command not found”
- Local path missing: install client tools (or let Docker path be used).
- Selected DB container doesn’t have client tools → script automatically switches to an ephemeral `postgres:16` client.

Connecting to host from Docker on macOS
- The script maps `localhost` to `host.docker.internal` automatically for ephemeral host-mode.

Windows editor corrupted batch special chars
- Save batch/PowerShell scripts as **UTF-8 (no BOM)** or ANSI; avoid smart quotes.

## Safety notes

- These are **logical dumps** (schema & data) created via `pg_dump --clean --if-exists --no-owner --no-privileges`.
Restores run with `psql -v ON_ERROR_STOP=1`.
- Minimal wrappers **overwrite** `current` dumps by design.
- Always test restore in a non-production environment before relying on backups.

## Handy one-liners

Mac/Linux:

```bash
# Quick daily with timestamp + gzip + 14-day retention
../scripts/db/backupdbzip_rotate.sh 14 ./backups

# Debug a failing backup
DEBUG=1 ../scripts/db/backupdb.sh nightly
```

Windows:
```bat
REM Quick daily with timestamp + gzip + 14-day retention
..\scripts\db\backupdbzip_rotate.bat 14 .\backups
```