Create Application Database

Creating a Postgres Database for an Application (CLI)

Option 1: DBA creates the database directly

Use this when the app should **not** be able to create/drop databases itself (recommended for most production apps).

```
sudo -u postgres psql
```

```
-- Create a dedicated role for the app
CREATE ROLE app_user WITH LOGIN PASSWORD 'strong_password_here';

-- Create the database, owned by that role
CREATE DATABASE app_db OWNER app_user;

-- Restrict connections to only this DB for that user
REVOKE ALL ON DATABASE app_db FROM PUBLIC;
GRANT CONNECT ON DATABASE app_db TO app_user;

-- Optional: default schema privileges
\c app_db
GRANT ALL ON SCHEMA public TO app_user;
```

Use case: app just needs a schema to read/write. Least privilege, no CREATEDB right.

Option 2: Role with `CREATEDB` permission

Use this when the app (or a deployment pipeline / ORM migration tool) needs to create its own database, e.g. CI/CD, multi-tenant provisioning.

```
CREATE ROLE app_user WITH LOGIN PASSWORD 'strong_password_here' CREATEDB;
```

Then the app itself (or its migration tool) can run:

```
CREATE DATABASE app_db OWNER app_user;
```

Trade-off: convenient for automation, but `CREATEDB` lets that role create *any* number of databases on the cluster — scope it to a dedicated role per app, never reuse across apps.

Adjusting `pg_hba.conf`

Location depends on install method:

```
sudo -u postgres psql -c "SHOW hba_file;"
```

Add a line **above** any broader/catch-all rule (order matters — first match wins):

```
# TYPE  DATABASE   USER      ADDRESS         METHOD
host    app_db     app_user  10.0.0.0/24     scram-sha-256
```

- Use `scram-sha-256` (not `md5`) unless you have a compatibility reason not to.
- Restrict `ADDRESS` to the actual app subnet/host — avoid `0.0.0.0/0`.
- For local Unix-socket app connections on the same host: `local  app_db  app_user  scram-sha-256`

Reload (no restart needed):

```
sudo -u postgres psql -c "SELECT pg_reload_conf();"
# or
sudo systemctl reload postgresql
```

Managing Multiple Postgres Instances on One Host (e.g. v15 + v18)

Debian/Ubuntu's `postgresql-common` framework handles this natively via **clusters**, each with its own port, data dir, config, and `pg_hba.conf`.

List all clusters:

```
pg_lsclusters
```

Example output:

```
Ver Cluster Port Status Owner    Data directory              Log file
15  main    5432 online postgres /var/lib/postgresql/15/main ...
18  main    5433 online postgres /var/lib/postgresql/18/main ...
```

Key points:

- Each version gets its own port automatically (5432, 5433, ...) — no manual port juggling needed.
- Config files live under `/etc/postgresql/<version>/<cluster>/` — `pg_hba.conf` and `postgresql.conf` are **per-instance**, edit the correct version's file.
- Connect to a specific instance with `-p`:

  ```
  psql -h localhost -p 5433 -U app_user -d app_db
  ```

- Manage individual clusters:

  ```
  sudo pg_ctlcluster 18 main reload
  sudo systemctl restart postgresql@18-main
  ```

- If not using Debian's cluster tooling (e.g. compiled from source or RPM-based), you must manually assign distinct `port`, `data_directory`, and `unix_socket_directories` per instance in

each `postgresql.conf`, and run each as a separate systemd service/data dir.

Recommendation: decide per-app which major version it targets, keep app roles/DBs isolated to one cluster, and never share a `pg_hba.conf` between versions — each instance's file only affects its own cluster.

Revision #1
Created 16 July 2026 17:48:54 by Oliver Karger
Updated 16 July 2026 17:56:57 by Oliver Karger