Restoring Your PostgreSQL Database from Backup

Restoring a database from a backup is a critical skill for disaster recovery, migrating data, or setting up development/testing environments. PostgreSQL provides powerful command-line tools for this: `psql` for plain-text `.sql` backups and `pg_restore` for custom-format (`.bak`) backups.

This document will guide you through the process of restoring your PostgreSQL database using either of these backup file types.

1. Understanding Backup Types and Restoration Tools

Before you begin, it's essential to know which type of backup file you have, as it dictates the tool you'll use:

- `.sql` **(Plain-Text SQL Script):**
  - This file contains standard SQL commands (DDL for schema, DML for data).
  - **Restoration Tool:** `psql` (the PostgreSQL interactive terminal).
  - **Key Requirement:** The target database **must exist and be empty** before you restore into it using `psql`.
- `.bak` **(Custom Format Archive):**
  - This is a compressed, binary archive created with `pg_dump -Fc`. It's not human-readable.
  - **Restoration Tool:** `pg_restore` (a utility specifically designed for this format).
  - **Key Requirement:** `pg_restore` is more flexible; it can **create the database for you** if it doesn't exist, or restore into an existing one. It also allows for selective restoration (e.g., only schema, only data, specific tables).

2. Prerequisites for Restoration

Regardless of the backup type, you'll need:

- **PostgreSQL Client Tools:** Ensure `psql` and/or `pg_restore` are installed and accessible in your system's PATH. (If you could run `pg_dump`, you likely have these).

- **Database Credentials:**
    - **Host:** (`h`): Usually `localhost` or the IP address/hostname of your database server.
    - **Port:** (`p`): Default is `5432`.
    - **Username:** (`U`): A PostgreSQL user with sufficient privileges to create databases (if needed) and write to the target database. The `postgres` superuser is typically used for this, or your application's database administrator user.
    - **Target Database Name:** (`d`): The name of the database where you want to restore the data. This database might need to be created first, depending on your backup type.
- **Your Backup File:** The `.sql` or `.bak` file you created earlier.
- **Sufficient Disk Space:** Ensure the target machine has enough disk space to accommodate the restored database.

3. Restoring from a `.sql` File (Plain-Text SQL Script)

This method involves creating an empty database first, then piping the SQL commands from your backup file into `psql`.

Step-by-Step:

1. **Open Your Terminal/Command Prompt.**
2. **Create a New, Empty Target Database:**
   You must create the database
   before restoring the `.sql` file. Connect to an existing database (like the default `postgres` database) and issue the `CREATE DATABASE` command.

```
# Connect to the default 'postgres' database as a privileged user (e.g., 'postgres')
psql -U your_username -h localhost -p 5432 -d postgres

# Inside the psql prompt (you'll see `postgres=#`):
CREATE DATABASE your_new_target_database_name;
```

```
# Exit psql:
\q
```

Replace `your_username` and `your_new_target_database_name` with your actual values.

Important: If the database already exists and you want to overwrite it, you must first drop it and then re-create it. **This will permanently delete all data in the existing database.**

```
psql -U your_username -h localhost -p 5432 -d postgres
# Inside psql:
DROP DATABASE IF EXISTS your_new_target_database_name;
CREATE DATABASE your_new_target_database_name;
\q
```

3. **Restore the `.sql` Backup:**
   Now, use
   `psql` to execute the commands from your `.sql` backup file into the newly created database. Navigate to the directory where your `.sql` file is located, or provide its full path.

   ```
   psql -U your_username -h localhost -p 5432 -d your_new_target_datab
   ase_name < path/to/your_backup_file.sql
   ```

   - `your_backup_file.sql`: The name/path of your plain-text SQL backup file.
   - `<`: This is the input redirection operator, which feeds the content of the `.sql` file into the `psql` command.

   Example:

   ```
   psql -U myuser -h localhost -p 5432 -d myapp_db_restored < myapp_d
   b_backup_20250625.sql
   ```

4. **Enter Password:** If your user requires a password, `psql` will prompt you for it.
5. **Verify Restoration:** After the command completes, connect to `your_new_target_database_name` using `psql` or a GUI tool (like pgAdmin or DBeaver) and verify that your tables, data, and other database objects are present and correct.

4. Restoring from a `.bak` File (Custom Format Archive)

This method uses `pg_restore`, which is designed specifically for the custom backup format. It's more versatile and handles creating the database schema automatically.

Step-by-Step:

1. **Open Your Terminal/Command Prompt.**
2. **Determine Target Database:**
   - **If the target database does NOT exist:** `pg_restore` can create it for you. You'll need to specify a database that *does* exist (like `postgres`) using the `d` flag, and then also specify the *name of the database you want to create/restore into* as an argument.
   - **If the target database DOES exist and is empty:** You can restore directly into it.
   - **If the target database DOES exist and you want to overwrite its content:** You might need to drop it first if `pg_restore` encounters conflicts, or use its `-clean` option (use with caution). The simplest approach for a full overwrite is often to drop and recreate the database first, similar to the `.sql` method.

   Let's assume you want to restore to `your_new_target_database_name`, and it might not exist yet:

   ```
   # Connect to the default 'postgres' database as a privileged user
   psql -U your_username -h localhost -p 5432 -d postgres

   # Inside psql: Create the database if it doesn't exist, or drop and recrea
   te if overwriting
   DROP DATABASE IF EXISTS your_new_target_database_name;
   CREATE DATABASE your_new_target_database_name;
   \q
   ```

   This explicit creation step, while `pg_restore` can sometimes create it, gives you more control and avoids potential issues.

3. **Restore the `.bak` Backup:**
   Navigate to the directory where your

`.bak` file is located, or provide its full path.

```
pg_restore -U your_username -h localhost -p 5432 -d your_new_target_database_name path/to/your_backup_file.bak
```

- `your_backup_file.bak`: The name/path of your custom format backup file.
- Notice there is **no** `<` **redirection** here, as `pg_restore` reads the file directly via the argument.

Example:

```
pg_restore -U myuser -h localhost -p 5432 -d myapp_db_restored myapp_db_backup_20250625210000.bak
```

4. **Enter Password:** If your user requires a password, `pg_restore` will prompt you for it.
5. **Verify Restoration:** After the command completes, connect to `your_new_target_database_name` and confirm that your database objects and data have been successfully restored.

5. Important Considerations After Restoration

- **User Permissions:** Ensure that your application's database user has the correct permissions on the newly restored database and its objects. Sometimes, permissions need to be explicitly granted after a restore, especially if the roles or users involved in the backup and restore are different.
- **Database Settings:** Check any specific database configurations, extensions, or custom settings that might not be included in a simple data dump.
- **Application Connection:** Update your application's database connection string (e.g., in your Node.js `config/db.js`) if the target database name, host, port, or credentials have changed.
- **Regular Testing:** As mentioned previously, regularly test your backup and restore process. This is the only way to be confident that your disaster recovery plan is viable.

By following these instructions, you should be able to confidently restore your PostgreSQL database from either a `.sql` or `.bak` backup file.