

# Creating a Database

PostgreSQL allows you to create databases using different methods, including the PostgreSQL interactive shell (`psql`), Docker (assuming PostgreSQL is running inside a container), and the command-line interface (`createdb`). This guide explains each method step-by-step, covering required permissions, best practices, and troubleshooting common issues.

## Creating Using psql CLI

PostgreSQL is a database system that stores and manages structured data efficiently. The `psql` tool is an interactive command-line interface (CLI) that allows users to execute SQL commands directly on a PostgreSQL database. Follow these steps to create a database:

### Connect to PostgreSQL

Open terminal on your local system, and if PostgreSQL is installed locally, connect using the following command. If not installed, install from [official website](#):

```
psql -U postgres
```

For a remote database, use:

```
psql -h HOST -U USER -d DATABASE
```

Replace `HOST` with the database server address, `USER` with the PostgreSQL username, and `DATABASE` with an existing database name.

### Create a New Database

Inside the `psql` shell, run:

```
CREATE DATABASE mydatabase;
```

The default settings will apply unless specified otherwise. To customize the encoding and collation, use:

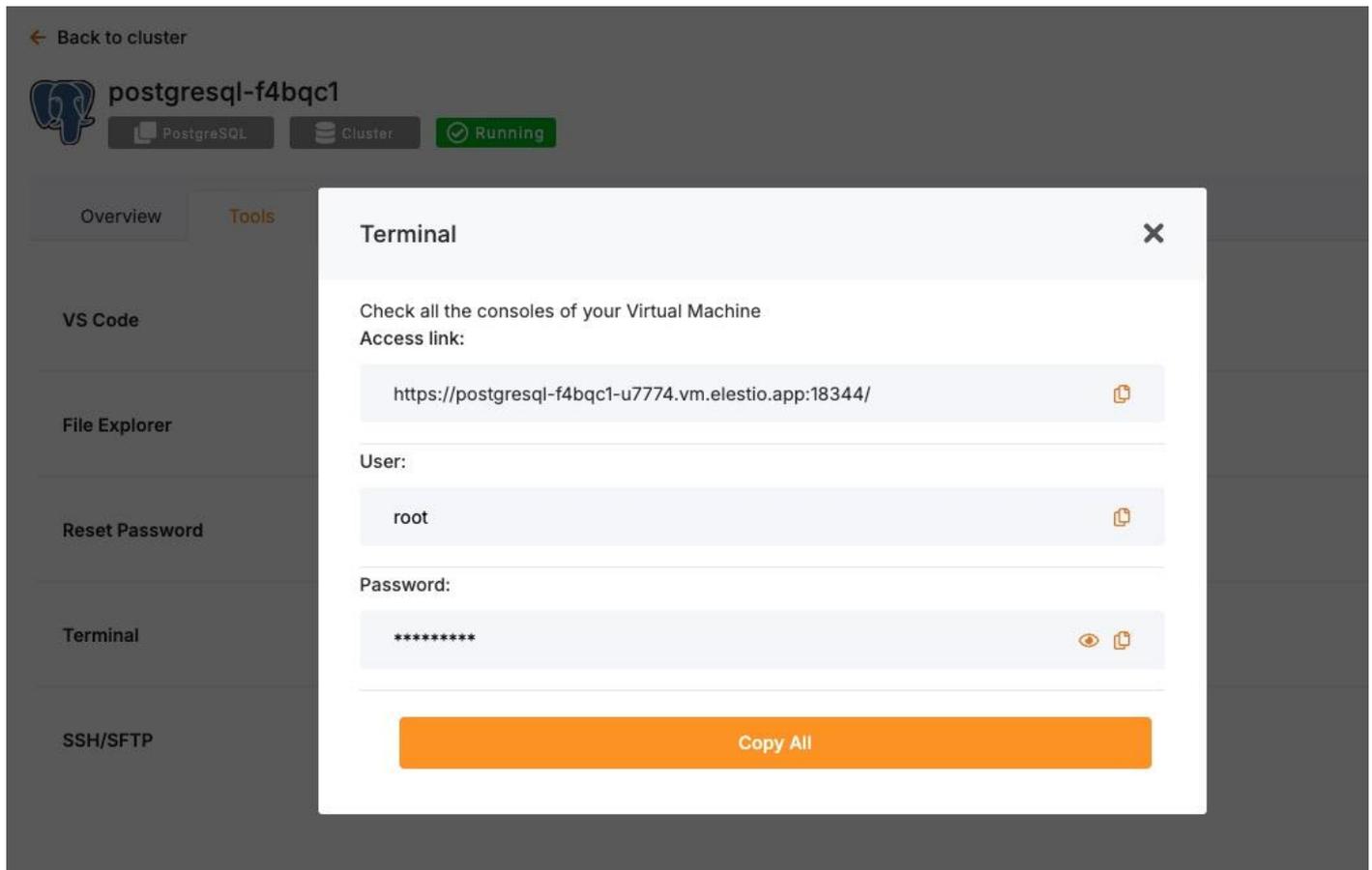
```
CREATE DATABASE mydatabase ENCODING 'UTF8' LC_COLLATE 'en_US.UTF-8' LC_CTYPE 'en_US.UTF-8'  
TEMPLATE template0;
```

# Creating Database in Docker

Docker is a tool that helps run applications in isolated environments called containers. A PostgreSQL container provides a self-contained database instance that can be quickly deployed and managed. If you are running PostgreSQL inside a Docker container, follow these steps:

## Access Elestio Terminal

Head over to your deployed PostgreSQL service dashboard and head over to **Tools > Terminal**. Use the credentials provided there to log in to your terminal.



Once you are in your terminal, run the following command to head over to the correct directory to perform the next steps

```
cd /opt/app/
```

## Access the PostgreSQL Container Shell

Instead of pulling an image or running the container manually, use Docker Compose to interact with your running container. As you are using Elestio, it will already be a Docker compose:

```
docker-compose exec postgres bash
```

This opens a shell session inside the running PostgreSQL container.

## Use Environment Variables to Connect via psql

Once inside the container shell, if environment variables like `POSTGRES_USER` and `POSTGRES_DB` are already set in the stack, you can use them directly:

```
psql -U "$POSTGRES_USER" -d "$POSTGRES_DB"
```

Or use the default one:

```
psql -U postgres
```

## Create Database

Now, to create a database, use the following command. This command tells PostgreSQL to create a new logical database called `mydatabase`. By default, it inherits settings like encoding and collation from the template database (template1), unless specified otherwise.

```
CREATE DATABASE mydatabase;
```

You can quickly list the database you just created using the following command

```
/l
```

# Creating Using createdb CLI

The `createdb` command simplifies database creation from the terminal without using `psql`.

## Ensure PostgreSQL is Running

Check the PostgreSQL service status, this ensures that the PostgreSQL instance is running on your local instance:

```
sudo systemctl status postgresql
```

If not running, start it:

```
sudo systemctl start postgresql
```

## Create a Database

Now, you can create a simple database using the following command:

```
createdb -U postgres mydatabase
```

To specify encoding and collation:

```
createdb -U postgres --encoding=UTF8 --lc-collate=en_US.UTF-8 --lc-ctype=en_US.UTF-8  
mydatabase
```

## Verify Database Creation

List all databases using the following commands, as it will list all the databases available under your PostgreSQL:

```
psql -U postgres -l
```

## Connect to the New Database

Next, you can easily connect with the database using the psql command and start working on it.

```
psql -U postgres -d mydatabase
```

# Required Permissions for Database Creation

Creating a database requires the `CREATEDB` privilege. By default, the `postgres` user has this privilege. To grant it to another user:

```
ALTER USER username CREATEDB;
```

For restricted access, assign specific permissions:

```
CREATE ROLE newuser WITH LOGIN PASSWORD 'securepassword';  
GRANT CONNECT ON DATABASE mydatabase TO newuser;  
GRANT USAGE ON SCHEMA public TO newuser;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO newuser;
```

# Best Practices for Creating Databases

- **Use Meaningful Names:** Choosing clear and descriptive names for databases helps in organization and maintenance. Avoid generic names like `testdb` or `database1`, as they do not indicate the database's purpose. Instead, use names that reflect the type of data stored, such as `customer_data` or `sales_records`. Meaningful names make it easier for developers and administrators to understand the database's function without extra documentation.
- **Follow Naming Conventions:** A standardized naming convention ensures consistency across projects and simplifies database management. PostgreSQL is case-sensitive, so using lowercase letters and underscores (e.g., `order_details`) is recommended to avoid unnecessary complexities. Avoid spaces and special characters in names, as they require additional quoting in SQL queries.
- **Restrict User Permissions:** Granting only the necessary permissions improves database security and reduces risks. By default, users should have the least privilege required for their tasks, such as read-only access for reporting tools. Superuser or administrative privileges should be limited to trusted users to prevent accidental or malicious changes. Using roles and groups simplifies permission management and ensures consistent access control.
- **Enable Backups:** Regular backups ensure data recovery in case of accidental deletions, hardware failures, or security breaches. PostgreSQL provides built-in tools like `pg_dump` for single-database backups and `pg_basebackup` for full-instance backups. Automating backups using cron jobs or scheduling them through a database management tool reduces the risk of data loss.
- **Monitor Performance:** Monitoring database performance helps identify bottlenecks, optimize queries, and ensure efficient resource utilization. PostgreSQL provides system views like `pg_stat_activity` and `pg_stat_database` to track query execution and database usage. Analyzing slow queries using `EXPLAIN ANALYZE` helps in indexing and optimization.

```
SELECT datname, numbackends, xact_commit, blks_read FROM pg_stat_database;
```

## Common Issues and Troubleshooting

| Issue  | Possible Cause                                   | Solution  |
|--|--|---|
| <code>ERROR: permission denied to create database</code> | User lacks <code>CREATEDB</code> privileges      | Grant permission using <code>ALTER USER username CREATEDB;</code>                         |
| <code>ERROR: database "mydatabase" already exists</code> | Database name already taken                      | Use a different name or drop the existing one with <code>DROP DATABASE mydatabase;</code> |
| <code>FATAL: database "mydatabase" does not exist</code> | Attempting to connect to a non-existent database | Verify creation using <code>\l</code>   |

| Issue  | Possible Cause                    | Solution   |
|--|-----------------------------------|--|
| <code>psql: could not connect to server</code>     | PostgreSQL is not running         | Start PostgreSQL with <code>sudo systemctl start postgresql</code>                     |
| <code>ERROR: role "username" does not exist</code> | The specified user does not exist | Create the user with <code>CREATE ROLE username WITH LOGIN PASSWORD 'password';</code> |

---

Revision #4

Created 24 March 2025 06:53:39 by kaiwalya

Updated 8 April 2025 07:37:15 by kaiwalya