Managing Roles and Access Control in Postgres

Step 1: Creating an Admin Role and Assigning Users

Let’s start by creating a role called `admin`, which has the privileges needed to create databases and roles but without superuser access. Then, we’ll assign multiple users to this role.

Creating the `admin` Role

```
CREATE ROLE admin WITH
  LOGIN
  NOSUPERUSER
  NOINHERIT
  CREATEDB
  CREATEROLE
  NOREPLICATION
  CONNECTION LIMIT 10
  PASSWORD 'securePass123'
  VALID UNTIL '2025-01-01 00:00:00+03';
```

In this example:

- `NOSUPERUSER` prevents the role from having superuser access.
- `CREATEDB` allows the role to create new databases.
- `CREATEROLE` lets the role create other roles.
- `CONNECTION LIMIT 10` restricts the number of simultaneous connections.

This role is useful for trusted users who need to create other roles and databases but should not have superuser privileges.

Adding Users to the `admin` Role

Let’s create users and assign them to the `admin` role:

```
CREATE ROLE kemal WITH LOGIN PASSWORD '**************';
CREATE ROLE ali WITH LOGIN PASSWORD '**************';

GRANT admin TO kemal;
GRANT admin TO ali;
```

Here, we created two users, `kemal` and `ali`, and granted them the admin role. Now, these users can inherit permissions from `admin`, but they still need to `SET ROLE admin` explicitly to access those privileges due to `NOINHERIT` being set.

Step 2: Creating a Group Role for Database Access

To manage access to a specific database folder, let's create a role group called `test`. We'll assign users to this role and control access using the `pg_hba.conf` file.

Creating the `test` Role

```
CREATE ROLE test;
```

This is a group role without login access; it will only be used to group user permissions.

Assigning Users to `test`

We'll assign `user1` and `user2` to `test`:

```
GRANT test TO user1;
GRANT test TO user2;
```

Now, both users are members of `test`. This role can be used to set up specific access permissions, and any new user assigned to this role will automatically inherit these permissions.

Step 3: Configuring Access with `pg_hba.conf`

The `pg_hba.conf` file (PostgreSQL Host-Based Authentication file) controls access to databases based on user roles, IP addresses, and authentication methods.

Granting Access to `test` Using `pg_hba.conf`

Suppose we want to allow `test` members to access a specific database named `db1`. We can add the following line to `pg_hba.conf`:

```
# TYPE  DATABASE      USER          ADDRESS          METHOD
host    db1           +test         10.*.**.**/32    md5
```

Explanation:

- `+test` means all users who are members of `test` can access `db1`.

Now, any user in the `test` role will be able to access `db1` with password authentication.

Step 4: Creating a Data Directory and Granting Access to Group Roles

Let’s create a folder in our database directory that will be accessible only to `test` members.

1. **Create a data directory** in your file system (e.g., `$PG_DATA/roles`).

```
mkdir -p $PG_DATA/test
```

Set directory permissions so only `test` users have access.

```
chown -R postgres:postgres $PG_DATA/test
chmod 640 $PG_DATA/test
```

Allow specific users from `test` to access this directory in PostgreSQL.

```
REVOKE ALL ON SCHEMA public FROM PUBLIC;
GRANT USAGE ON SCHEMA public TO test;
```

This setup ensures that only members of `test` can access or use objects in the public schema within the `db3` database, and they have exclusive access to `$PG_DATA/test` on the server.

Step 5: Using `@` and Role Lists in `pg_hba.conf`

The `pg_hba.conf` file also supports specifying role lists and groups using the `@` symbol and grouping characters.

Using Role Lists and Files

You can simplify multiple role assignments in `pg_hba.conf` by using role lists or external files.

For instance:

```
# Combine access for `test` and other roles
local    db3    @test                md5
```

Here:

- `@test` would refer to a file listing trusted roles allowed to access `db3`. You could create this file with a list of roles:

```
vi $PG_DATA/test

# add user name who want to connect to database
cat $PG_DATA/roles
user1
user2
user3
```

- Then, users in `roles` can access `db3`.

Conclusion

Using roles and the `pg_hba.conf` file in PostgreSQL enables you to manage database access securely and efficiently. With a few configurations, you can control user access based on role memberships, simplify permission management, and secure sensitive data by managing folder and file permissions. Through these examples, you now have a practical guide to set up PostgreSQL roles and configure secure access based on real-world needs. Properly managing roles and access ensures a robust, well-organized database security model, tailored to your organization's specific requirements.