DigitalOcean

By: Justin Ellingwood

Subscribe

How To Use Roles and Manage Grant Permissions in PostgreSQL on a VPS

29

Posted August 5, 2013 486.4k POSTGRESQL UBUNTU

What is PostgreSQL

PostgreSQL is an open source database management system that uses the SQL querying language. PostgreSQL is a powerful tool that can be used to manage application and web data on your VPS.

In this gude, we will discuss how to properly manage privileges and user grant permissions. This will allow you to provide your applications the privileges they need without allowing them freedom to affect separate databases.

We will be using PostgreSQL on an Ubuntu 12.04 cloud server, but everything outside of installation should work the same on any modern Linux distribution.

Initial PostgreSQL Setup

If you do not already have PostgreSQL installed on Ubuntu, type the following commands to download and install it:

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

During the installation, PostgreSQL will create a default user to operate under. We will be using this user for the initial steps. Log in with the following command:

```
sudo su - postgres
```

Our environment is now prepared and we can begin learning about how PostgreSQL handles permissions.

SCROLL TO TOP

PostgreSQL Permission Concepts

PostgreSQL (or simply "postgres") manages permissions through the concept of "roles".

Roles are different from traditional Unix-style permissions in that there is no distinction between users and groups. Roles can be manipulated to resemble both of these conventions, but they are also more flexible.

For instance, roles can be members of other roles, allowing them to take on the permission characteristics of previously defined roles. Roles can also own objects and control access to those object for other roles.

How to View Roles in PostgreSQL

We can view the current defined roles in PostgreSQL by logging into the prompt interface with the following command:

```
psql
```

To get a list of roles, type this:

```
\du
```

```
                             List of roles
 Role name |                   Attributes                   | Member of
-----------+------------------------------------------------+-----------
 postgres  | Superuser, Create role, Create DB, Replication | {}
```

As you can see, there is only one default role, and it has many powerful privileges.

How to Create Roles in PostgreSQL

There are a number of different ways to create roles for Postgres. It is possible to create roles from within Postgres, or from the command line.

How to Create Roles From Within PostgreSQL

The most basic way of creating new roles is from within the Postgres prompt interface.

You can create a new role with the following syntax:

```
CREATE ROLE new_role_name;
```

Let's create a new role called "demo_role":

```
CREATE ROLE demo_role;
```

```
CREATE ROLE
```

If we check the defined users again, we will get the following:

```
\du
```

```
                             List of roles
 Role name |                   Attributes                   | Member of
-----------+------------------------------------------------+-----------
 demo_role | Cannot login                                   | {}
 postgres  | Superuser, Create role, Create DB, Replication | {}
```

The only information we are given is that the new role has no login privileges. This is okay for now.

How to Create Roles from the Command Line

An alternative method of creating roles is using the "createuser" command.

Get out of the PostgreSQL command prompt for a moment by typing:

```
\q
```

Create a role called "test_user" with the following command:

```
createuser test_user
```

```
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

You will be asked a series of questions that will define some initial permissions for the new role. If you answer "n" for no to all of these prompts, you will create a user similar to the previous user we created.

We can log back into the Postgres prompt and issue the "\du" command again to see the differences between the two new roles:

```
psql
\du
```

```
                             List of roles
 Role name |                   Attributes                   | Member of
-----------+------------------------------------------------+-----------
 demo_role | Cannot login                                   | {}
 postgres  | Superuser, Create role, Create DB, Replication | {}
 test_user |                                                | {}
```

As you can see, these commands do not produce identical results. The user created from the command line does not have "Cannot login" listed as an attribute.

How to Delete Roles In PostgreSQL

As an exercise, let's try to get "demo_role" to have the same permissions as "test_user". We will attempt this during creation first, and later will learn how to alter the permissions of an existing role.

Before we can practice defining permissions for "demo_role" on creation, we need to destroy the current role so that we can try again.

You can delete a role using the following syntax:

```
DROP ROLE role_name;
```

Delete the "demo_role" role by typing:

```
DROP ROLE demo_role;
```

```
DROP ROLE
```


If we issue the command on a non-existant user, we will receive this error:

```
DROP ROLE demo_role;
```

```
ERROR:  role "demo_role" does not exist
```

To avoid this situation and make the drop command delete a user if present and quietly do nothing if the user does not exist, use the following syntax:

```
DROP ROLE IF EXISTS role_name;
```

As you can see, with this option specified, the command will complete successfully regardless of the validity of the role:

```
DROP ROLE IF EXISTS demo_role;
```

```
NOTICE:  role "demo_role" does not exist, skipping
DROP ROLE
```

How to Define Privileges Upon Role Creation

Now, we are ready to recreate the "demo_role" role with altered permissions. We can do this by specifying the permissions we want after the main create clause:

```
CREATE ROLE role_name WITH optional_permissions;
```

You can see a full list of the options by typing:

```
\h CREATE ROLE
```

We want to give this user the ability to log in, so we will type:

```
CREATE ROLE demo_role WITH LOGIN;
```

```
CREATE ROLE
```

If we check the attributes again, we SCROLL TO TOP › users now have identical privileges:

```
\du

                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 demo_role |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication             | {}
 test_user |                                                            | {}
```

If we want to get to this state without specifying the "login" attribute with every role creation, we can actually use the following command instead of the "CREATE ROLE" command:

```
CREATE USER role_name;
```

The only difference between the two commands is that "CREATE USER" automatically gives the role login privileges.

How to Change Privileges of Roles in PostgreSQL

To change the attributes of an already created role, we use the "ALTER ROLE" command.

This command allows us to define privilege changes without having to delete and recreate users as we demonstrated earlier.

The basic syntax is:

```
ALTER ROLE role_name WITH attribute_options;
```

For instance, we can change "demo_role" back to its previous state by issuing this command:

```
ALTER ROLE demo_role WITH NOLOGIN;


ALTER ROLE
```

We can see the privileges have reverted to their previous state:

```
\du
```

```
                             List of roles
 Role name |                   Attributes                   | Member of
-----------+------------------------------------------------+-----------
 demo_role | Cannot login                                   | {}
 postgres  | Superuser, Create role, Create DB, Replication | {}
 test_user |                                                | {}
```

We can easily change it back with the following command:

```
ALTER ROLE demo_role WITH LOGIN;
```

How to Log In as a Different User in PostgreSQL

By default, users are only allowed to login locally if the system username matches the PostgreSQL username.

We can get around this by either changing the login type, or by specifying that PostgreSQL should use the loopback network interface, which would change the connection type to remote, even though it is actually a local connection.

We will discuss the second option. First, we need to give the user we'd like to connect as a password so that we can authenticate.

Give the "test_user" a password with the following command:

```
\password test_user
```

You will be prompted to enter and confirm a password. Now, exit the PostgreSQL interface and exit back to your normal user.

```
\q
exit
```

PostgreSQL assumes that when you log in, you will be using a username that matches your operating system username, and that you will be connecting to a database with the same name as well.

This is not the case with the situation we are demonstrating, so we will need to explicitly specify the options we want to use. Use the following syntax:

```
psql -U user_name -d database_name -h 127.0.0.1 -W
```

The "user_name" should be replaced with the username we want to connect with. Similarly, the "database_name" should be the name of an existing database that you have access to.

The "-h 127.0.0.1" section is the part that specifies that we will be connecting to the local machine, but through a network interface, which allows us to authenticate even though our system username does not match. The "-W" flag tells PostgreSQL that we will be entering a password.

To log in with our "test_user" we can issue the following command:

```
psql -U test_user -d postgres -h 127.0.0.1 -W
```

```
Password for user test_user:
```

You will need to enter the password you configured. In our example, we use the database "postgres". This is the default database set up during install.

If you attempt to perform some actions in this session, you will see that you don't have the ability to do many things. This is because we did not give "test_user" permissions to administer many things.

Let's exit and get back into the administrative session:

```
\q
sudo su - postgres
psql
```

How to Grant Permissions in PostgreSQL

When a database or table is created, usually only the role that created it (not including roles with superuser status) has permission to modify it. We can alter this behavior by granting permissions to other roles.

We can grant permissions using the "GRANT" command. The general syntax is here:

```
GRANT permission_type ON table_name TO role_name;
```

Create a simple table to practice these concepts:

```
CREATE TABLE demo (
name varchar(25),
id serial,
start_date date);
```

```
NOTICE:  CREATE TABLE will create implicit sequence "demo_id_seq" for seri
CREATE TABLE
```

We can see the result with:

```
\d
```

```
             List of relations
 Schema |    Name     |   Type   |  Owner
--------+-------------+----------+----------
 public | demo        | table    | postgres
 public | demo_id_seq | sequence | postgres
(2 rows)
```

We can now grant some privileges to the new "demo" table to "demo_role". Give the user "UPDATE" privileges with the following command:

```
GRANT UPDATE ON demo TO demo_role;
```

We can grant full permissions to a user by substituting the permission type with the word "all":

```
GRANT ALL ON demo TO test_user;
```

If we want to specify permissions for every user on the system, we can use the word "public" instead of a specific user:

```
GRANT INSERT ON demo TO PUBL
```

To view the grant table, use the following command:

```
\z
```

```
                                  Access privileges
 Schema |    Name     |   Type   |     Access privileges      | Column acce
--------+-------------+----------+----------------------------+------------
 public | demo        | table    | postgres=arwdDxt/postgres +|
        |             |          | demo_role=w/postgres      +|
        |             |          | test_user=arwdDxt/postgres+|
        |             |          | =a/postgres                |
 public | demo_id_seq | sequence |                            |
(2 rows)
```

This shows all of the grant permissions we have just assigned.

How to Remove Permissions in PostgreSQL

You can remove permissions using the "REVOKE" command. The revoke command uses almost the same syntax as grant:

```
REVOKE permission_type ON table_name FROM user_name;
```

Again, we can use the same shorthand words (all and public) to make the command easier:

```
REVOKE INSERT ON demo FROM PUBLIC;
```

How to Use Group Roles in PostgreSQL

Roles are flexible enough to allow grouping of other roles to allow for widespread permissions control.

For instance, we can create a new role called "temporary_users" and then add "demo_role" and "test_user" to that role:

```
CREATE ROLE temporary_users;
GRANT temporary_users TO demo_role;
GRANT temporary_users TO test_user;
```

SCROLL TO TOP

Now these two users can have their permissions managed by manipulating the "temporary_users" group role instead of managing each member individually.

We can see the role membership information by typing:

```
\du
```

```
                                   List of roles
    Role name    |                   Attributes                   |     Mer
-----------------+------------------------------------------------+--------
 demo_role       |                                                | {tempo
 postgres        | Superuser, Create role, Create DB, Replication | {}
 temporary_users | Cannot login                                   | {}
 test_user       |                                                | {tempo
```

Any member of a group role can act as the group role they are a member of by using the "set role" command.

Since the "postgres" user we are logged into currently has superuser privileges, we can use "set role" even though we are not a member of that group:

```
SET ROLE temporary_users;
```

Now, any tables that are created are owned by the temporary_users role:

```
CREATE TABLE hello (
name varchar(25),
id serial,
start_date date);
```

We can check the table ownership by issuing this command:

```
\d
```

```
                List of relations
 Schema |     Name     |   Type   |      Owner
--------+--------------+----------+-----------------
 public | demo         | table    | postgres
 public | demo_id_seq  | sequence | postgres
 public | hello        | table    | temporary_users
 public | hello_id_seq | sequence | temporary_users
```

```
(4 rows)
```

As you can see, the new table (and the sequence associated with the serial data type) is owned by the "temporary_users" role.

We can get back to our original role permissions with the following command:

```
RESET ROLE;
```

If we give a user the "inherit" property with the "alter role" command, that user will automatically have all of the privileges of the roles they belong to without using the "set role" command:

```
ALTER ROLE test_user INHERIT;
```

Now test_user will have every permission of the roles it is a member of.

We can remove a group role (or any role) with the "drop role" command:

```
DROP ROLE temporary_users;
```

```
ERROR:  role "temporary_users" cannot be dropped because some objects depe
DETAIL:  owner of table hello
owner of sequence hello_id_seq
```

This will give you an error, because we created a table that is owned by "temporary_users". We can solve this problem by transferring ownership to a different role:

```
ALTER TABLE hello OWNER TO demo_role;
```

If we check, we can see that "temporary_users" no longer owns any of the tables:

```
\d
```

```
              List of relations
 Schema |     Name     |   Type   |   Owner
--------+--------------+-----     '         ---
 public | demo         | tab
 public | demo_id_seq  | sequence | postgres
```

```
 public | hello        | table    | demo_role
 public | hello_id_seq | sequence | demo_role
(4 rows)
```

We can now drop the "temporary_users" role successfully by issuing the command again:

```
DROP ROLE temporary_users;
```

This will destroy the temporary_users role. The former members of temporary_users are not removed.

Conclusion

You should now have the basic skills necessary to administer your PostgreSQL database permissions. It is important to know how to manage permissions so that your applications can access the databases they need, while not disrupting data used by other applications.

By Justin Ellingwood

By: Justin Ellingwood

Related Tutorials

How To Customize the PostgreSQL Prompt with psqlrc on Ubuntu 14.04

How to Use Full-Text C... ...SQL on Ubuntu 16.04

SCROLL TO TOP

How To Install Concourse CI on Ubuntu 16.04

How To Secure PostgreSQL Against Automated Attacks

How To Set Up Django with Postgres, Nginx, and Gunicorn on Debian 8

6 Comments

Leave a comment...

Log In to Comment

david484130 *February 20, 2014*

1 great article. helped me a lot!

alanhroomph *October 23, 2014*

1 Excellent, easy-to-follow intro to roles and permissions in PostgreSQL. Thanks Justin.

A small note. On Ubuntu 14.04, after logging in as 'test_user', then reverting to the admin session and typing **sudo su - postgres** from the command line, I found I was asked for the postgres password, which I didn't set / didn't know. Problem solved by opening a new terminal and entering the same command - then I was asked for my standard Ubuntu password.

acandaele *November 7, 2014*

0 Hi, I'm trying to create a user with permissions to create a database. I'm logged in as the deploy user and I'm able to create the user 'deploy'

However, when I try to alter the role and give it CREATEDB permissions with the command:

```
ALTER ROLE deploy WITH CREA
```

SCROLL TO TOP

and then check on the changed permissions with \du, nothing has happened. The 'deploy' user is there, but it has no CREATEDB permissions.

I also tried this as a root user, with the same result.

I also tried to create the user from the commandline with:

createuser deploy

according to this tutorial I should get these questions:

```
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) n
Shall the new role be allowed to create more new roles? (y/n) n
```

but I didn't, the user gets created and I'm back at the prompt.

at the prompt I'm also seeing:

```
psql (9.4beta3)
```

Does that mean my version of Postgres is 9.4? That's odd, because I installed version 9.3

thanks for your help,

Anthony

acandaele *November 7, 2014*

0 my bad, I for to add a ';' after my sql commands

saikirankv *July 18, 2017*

0 This article helped me a lot, Thanks

asgowrisankar *September 1, 2017*

0 Thanks! Didn't know a default user is created and its name is postgres

SCROLL TO TOP