How to install PostgreSQL 11 on CentOS 7?

Tuesday, October 30, 2018 15:21

By
angelomarquez
-
October 26, 2018
0
472

PostgreSQL is one of the best relational database managers in the world. Its main virtue is to be a reliable rock. Today, I will show you how to install PostgreSQL 11 on CentOS 7.

In this site we have talked before about PostgreSQL, it is difficult not to do it when you have in front of a database manager as advanced and reliable as this one. Recently, version 11 of this fantastic software was released.

In this new version, it highlights the speed improvements in the processing of very large databases. Something sysadmin will surely appreciate.

On the other hand, PostgreSQL 11 adds the ability to partition data by a hash key. This translates into improved data robustness and security.

Install PostgreSQL 11 on CentOS 7

PostgreSQL 11 is a novelty. It is the first big release since October 2017. Many sysadmins will look for a way to update or install it from 0. The latter is the goal of this article.

1. Upgrade the system

First, you need to update the system. This in order to get the latest security patches available for the system. With this, you gain stability and robustness.

```
:~$ su
:~# yum update
```

1.- Upgrading centOS 7

Now you can continue the installation.

2. Adding the PostgreSQL repository

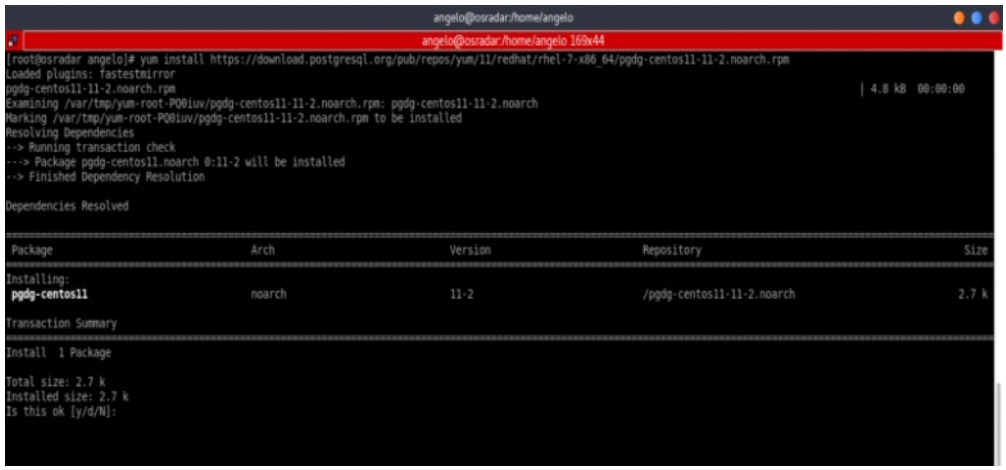
Actually, PostgreSQL is available for many operating systems such as Debian, Ubuntu, or CentOS 7, but the version that comes in the official repositories of each distribution is usually obsolete and outdated.

In short, you need to resort to an external repository to perform the installation. Fortunately, PostgreSQL has one and it is very easy.

First, run this command as root user (you can get the latest rpm url from here):

Here is the rpm chart of PostgreSQL if you want to install it with offline mode.

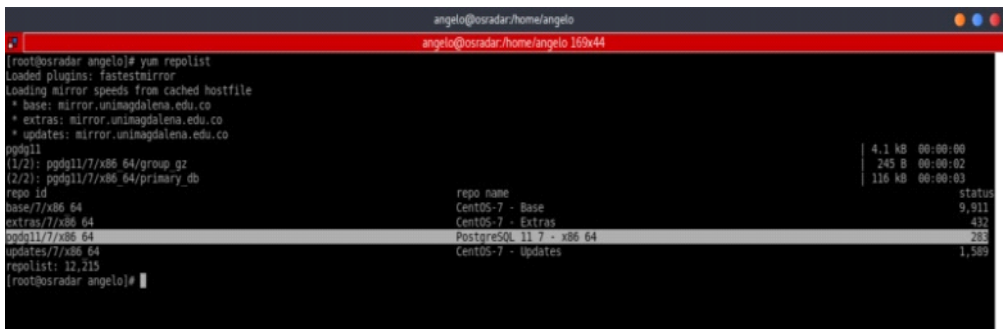
```
:~# yum install https://download.postgresql.org/pub/repos/yum/11/redhat/rhel-7-x86_64/pgdg-centos11-11-2.noarch.rpm
```



2.- Adding the repository
After that, the repository has been added.

You can also verify that.

```
:~# yum repolist
```

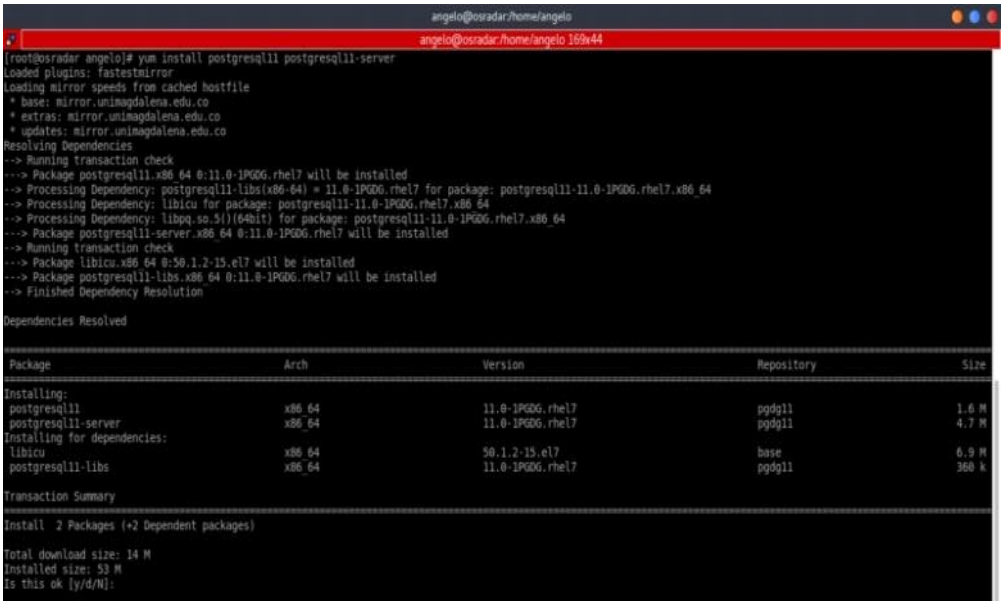


3.- The repositories added
As you can see, the repository is added

3. Install PostgreSQL 11

Thanks to the repository, now, you can install the new version of PostgreSQL in an easy way. Just run this command:

```
:~# yum install postgresql11 postgresql11-server
```



4.- Install PostgreSQL 11

After that, initialize the database:

```
:~# /usr/pgsql-11/bin/postgresql-11-setup initdb
```

Next, start and enable PostgreSQL service:

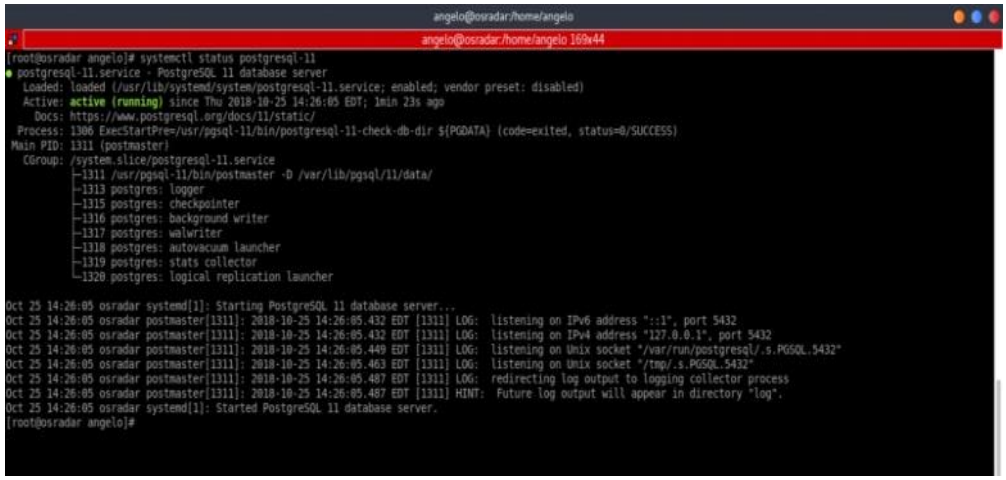
```
:~# systemctl enable postgresql-11
:~# systemctl start postgresql-11
```

5.- Starting the Postgresql 11 service

This process must be done because of the security policies of CentOS. CentOS 7 installs the package but does not start the PostgreSQL service automatically.

Finally, check the service status.

```
1 :~# systemctl status postgresql-11
```



6.- Checking the status of the service

The installation has been successful

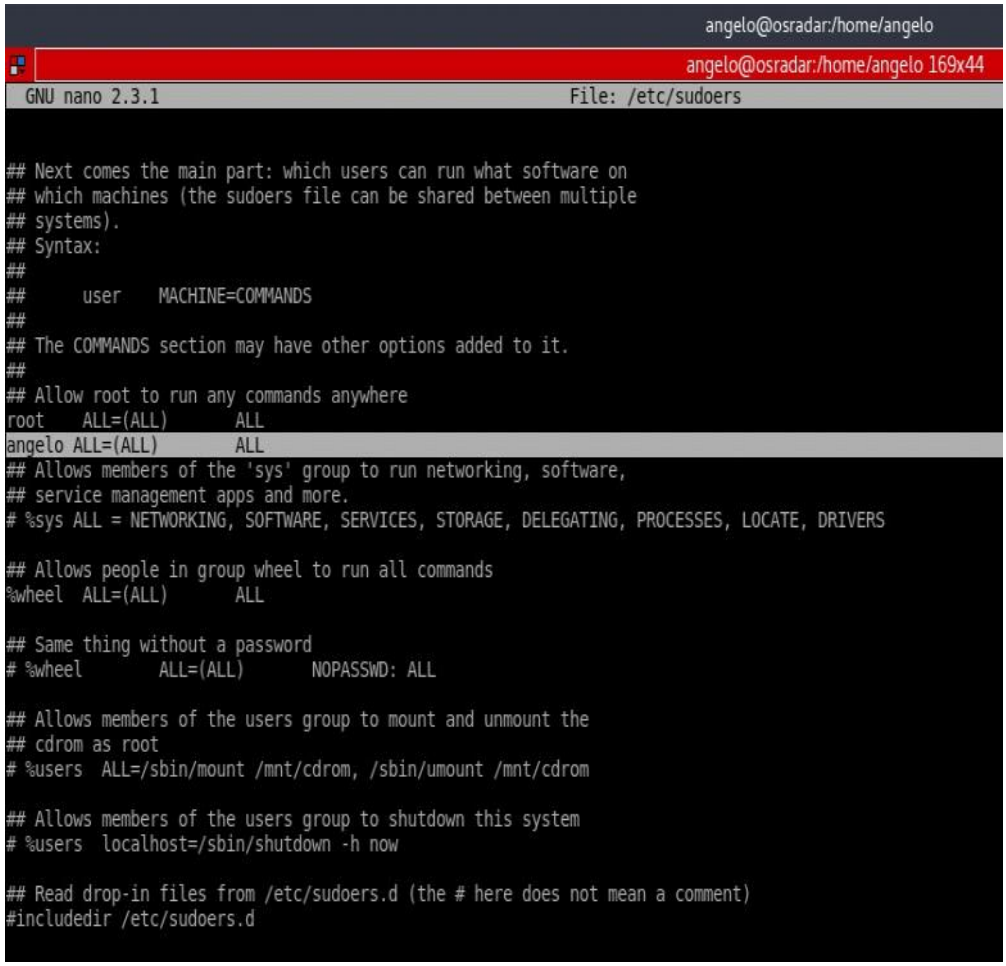
4. Using PostgreSQL 11

~~*Before using PostgreSQL You must install sudo. You also have to add your current user to the sudoers file:*~~

```
:~# yum install sudo
```

~~*Then, open the file `/etc/sudoers` and edit it. See the image for the guide. In "angelo" write your user.*~~

```
:~# nano /etc/sudoers
```



~~*7.- Editing the sudoers file*~~

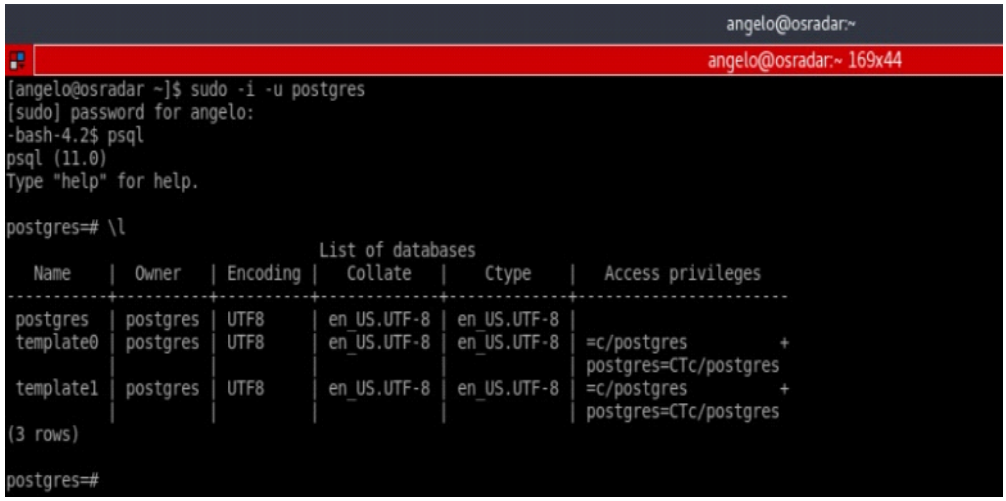
~~*Then, close session and access as postgres user.*~~

```
:~# exit
:~$ sudo -i -u postgres
:~$ psql
```

8.- PostgreSQL log in

Now, you can write some commands. For example, list all database:

```
postgres=# \l
```



9.- List all databases

Conclusion

PostgreSQL is a great database manager. Each time it improves more and more and aims to be the leading open source option in this area. This new version adds performance improvements and installing it in CentOS 7 is not a big complication compared to its benefits.

Please share this article on your social networks.

From <https://www.osradar.com/how-to-install-postgresql-11-centos-7/>

Change a Password for PostgreSQL on Linux via Command Line

PostgreSQL supports many client authentication methods, but in this case we're only going to concern ourselves with two: **password** and **md5**.
Note: The default authentication method for PostgreSQL is **ident**. If you'd like to change the PostgreSQL authentication method from ident to md5, then visit the linked tutorial!

Pre-Flight Check

- These instructions are intended specifically for changing a password in PostgreSQL.
- I'll be working from a Liquid Web Core Managed CentOS 7 server, and I'll be logged in as root.
- PostgreSQL is installed per our tutorial on: How to Install and Connect to PostgreSQL on CentOS 7.

Step #1: Switch to the PostgreSQL User: postgres

If you're working from a default PostgreSQL installation, then PostgreSQL will be configured with the user **postgres**.
Since we're logged in as **root**, and we're assuming that root doesn't have a user for PostgreSQL, switch to the default PostgreSQL user: **postgres**:

```
su - postgres
```

… then attempt a connection to PostgreSQL:

```
psql
```

… enter your password at the prompt:

```
Password:
```

… the correct, valid response will be similar to:

```
psql (9.3.9)
Type "help" for help.

postgres=#
```

Step #2: Add/Change the Password for the PostgreSQL User: postgres

Use the following command to change the password for your current user, which is now **postgres**:

```
\password
```

Enter your new password, and then enter it again to confirm it:

```
Enter new password:
Enter it again:
```

Now quit the PostgreSQL interface:

```
\q
```

Bonus Information!

You can do all of step one in exactly one command:

```
su -c "psql" - postgres
```

Be Sociable, Share!

From <https://www.liquidweb.com/kb/change-a-password-for-postgresql-on-linux-via-command-line/>

Configure PostgreSQL to allow remote connection

January 23, 2016 - Neeraj Singh

By default PostgreSQL is configured to be bound to “localhost”.

```
$ netstat -nlt
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:11211         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:3737          0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
```

As we can see above port 5432 is bound to 127.0.0.1. It means any attempt to connect to the postgresql server from outside the machine will be refused. We can try hitting the port 5432 by using telnet.

```
$ telnet 107.170.11.79 5432
Trying 107.170.11.79...
telnet: connect to address 107.170.11.79: Connection refused
telnet: Unable to connect to remote host
```

Configuring postgresql.conf

In order to fix this issue we need to find postgresql.conf. In different systems it is located at different place. I usually search for it.

```
$ find \ -name "postgresql.conf"
/var/lib/pgsql/9.4/data/postgresql.conf
```

Open postgresql.conf file and replace line

```
listen_addresses = 'localhost'
```

with

```
listen_addresses = '*'
```

Now restart postgresql server.

```
$ netstat -nlt
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:11211         0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:2812          0.0.0.0:*               LISTEN
tcp6       0      0 ::1:11211               :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
```

```
tcp6       0      0 :::5432                 :::*                    LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
```

Here we can see that “Local Address” for port 5432 has changed to 0.0.0.0.

Configuring pg_hba.conf

Let’s try to connect to remote postgresql server using “psql”.

```
$ psql -h 107.170.158.89 -U postgres
psql: could not connect to server: Connection refused
        Is the server running on host "107.170.158.89" and accepting
        TCP/IP connections on port 5432?
```

In order to fix it, open pg_hba.conf and add following entry at the very end.

```
host    all             all             0.0.0.0/0                 md5
host    all             all             ::/0                      md5
```

The second entry is for IPv6 network.

Do not get confused by “md5” option mentioned above. All it means is that a password needs to be provided. If you want client to allow collection without providing any password then change “md5” to “trust” and that will allow connection unconditionally.

Restart postgresql server.

```
$ psql -h 107.170.158.89 -U postgres
Password for user postgres:
psql (9.4.1, server 9.4.5)
Type "help" for help.
postgres=# \l
```

You should be able to see list of databases.

Now we are able to connect to postgresql server remotely.

Please note that in the real world you should be using extra layer of security by using “iptables”.

From <https://blog.bigbinary.com/2016/01/23/configure-postgresql-to-allow-remote-connection.html>