Credits: This note is based on: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04>

**Introduction**

Relational database management systems are a key component of many web sites and applications. They provide a structured way to store, organize, and access information.

[PostgreSQL](https://www.postgresql.org/), or Postgres, is a relational database management system that provides an implementation of the SQL querying language. It is a popular choice for many small and large projects and has the advantage of being standards-compliant and having many advanced features like reliable transactions and concurrency without read locks.

This guide demonstrates how to install Postgres on an Ubuntu 18.04 VPS instance and also provides instructions for basic database administration.

Credits: This note is based on:

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-18-04>

**Prerequisites**

To follow along with this tutorial, you will need one Ubuntu 18.04 server that has been configured by following our [Initial Server Setup for Ubuntu 18.04](https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu-18-04) guide. After completing this prerequisite tutorial, your server should have a non-**root** user with sudo permissions and a basic firewall.

**Step 1 — Installing PostgreSQL**

Ubuntu’s default repositories contain Postgres packages, so you can install these using the apt packaging system.

Since this is your first time using apt in this session, refresh your local package index. Then, install the Postgres package along with a -contrib package that adds some additional utilities and functionality:

* sudo apt update
* sudo apt install postgresql postgresql-contrib

Now that the software is installed, we can go over how it works and how it may be different from similar database management systems you may have used.

My note: v10.14 is installed with Ubuntu 18.04.1. run the following to check its version:

* Sudo -I -u postgres
* Psql >> displays its version.

**Step 2 — Using PostgreSQL Roles and Databases**

By default, Postgres uses a concept called “roles” to handle in authentication and authorization. These are, in some ways, similar to regular Unix-style accounts, but Postgres does not distinguish between users and groups and instead prefers the more flexible term “role”.

Upon installation, Postgres is set up to use *ident* authentication, meaning that it associates Postgres roles with a matching Unix/Linux system account. If a role exists within Postgres, a Unix/Linux username with the same name is able to sign in as that role.

The installation procedure created a user account called **postgres** that is associated with the default Postgres role. In order to use Postgres, you can log into that account.

There are a few ways to utilize this account to access Postgres.

**Switching Over to the postgres Account**

Switch over to the **postgres** account on your server by typing:

* sudo -i -u postgres

You can now access a Postgres prompt immediately by typing:

* psql

This will log you into the PostgreSQL prompt, and from here you are free to interact with the database management system right away.

Exit out of the PostgreSQL prompt by typing:

* \q

This will bring you back to the postgres Linux command prompt.

**Accessing a Postgres Prompt Without Switching Accounts**

You can also run the command you’d like with the **postgres** account directly with sudo.

For instance, in the last example, you were instructed to get to the Postgres prompt by first switching to the **postgres** user and then running psql to open the Postgres prompt. You could do this in one step by running the single command psql as the **postgres** user with sudo, like this:

* sudo -u postgres psql

This will log you directly into Postgres without the intermediary bash shell in between.

Again, you can exit the interactive Postgres session by typing:

* \q

Many use cases require more than one Postgres role. Read on to learn how to configure these.

**Step 3 — Creating a New Role**

Currently, you just have the **postgres** role configured within the database. You can create new roles from the command line with the createrole command. The --interactive flag will prompt you for the name of the new role and also ask whether it should have superuser permissions.

If you are logged in as the **postgres** account, you can create a new user by typing:

* createuser --interactive

If, instead, you prefer to use sudo for each command without switching from your normal account, type:

* sudo -u postgres createuser --interactive

The script will prompt you with some choices and, based on your responses, execute the correct Postgres commands to create a user to your specifications.

Output

Enter name of role to add: sammy

Shall the new role be a superuser? (y/n) y

You can get more control by passing some additional flags. Check out the options by looking at the man page:

* man createuser

Your installation of Postgres now has a new user, but you have not yet added any databases. The next section describes this process.

**Step 4 — Creating a New Database**

Another assumption that the Postgres authentication system makes by default is that for any role used to log in, that role will have a database with the same name which it can access.

This means that, if the user you created in the last section is called **sammy**, that role will attempt to connect to a database which is also called “sammy” by default. You can create the appropriate database with the createdb command.

If you are logged in as the **postgres** account, you would type something like:

* createdb sammy

If, instead, you prefer to use sudo for each command without switching from your normal account, you would type:

* sudo -u postgres createdb sammy

This flexibility provides multiple paths for creating databases as needed.

**Step 5 — Opening a Postgres Prompt with the New Role**

To log in with ident based authentication, you’ll need a Linux user with the same name as your Postgres role and database.

If you don’t have a matching Linux user available, you can create one with the adduser command. You will have to do this from your non-**root** account with sudo privileges (meaning, not logged in as the **postgres** user):

* sudo adduser sammy

Once this new account is available, you can either switch over and connect to the database by typing:

* sudo -i -u sammy
* psql

Or, you can do this inline:

* sudo -u sammy psql

This command will log you in automatically, assuming that all of the components have been properly configured.

If you want your user to connect to a different database, you can do so by specifying the database like this:

* psql -d postgres

Once logged in, you can get check your current connection information by typing:

* \conninfo

Output

You are connected to database "sammy" as user "sammy" via socket in "/var/run/postgresql" at port "5432".

This is useful if you are connecting to non-default databases or with non-default users.

My notes: adding marvin as a role:

* 1. Sudo -i -u postgres
  2. Createuser –interactive
  3. marvin - name to of the role to add
  4. Createdb winn\_itrac\_1
  5. Exit to switch back to marvin
  6. Psql -d winn-apache2-1 (or any existing db)
* Use \du or \du+ psql command to list all users in the current database server.
* Use the SELECT statement to query the user information from the pg\_catalog.pg\_user catalog.

**SELECT** usename **AS** role\_name,

**CASE**

**WHEN** usesuper **AND** usecreatedb **THEN**

**CAST**('superuser, create database' **AS** pg\_catalog.text)

**WHEN** usesuper **THEN**

**CAST**('superuser' **AS** pg\_catalog.text)

**WHEN** usecreatedb **THEN**

**CAST**('create database' **AS** pg\_catalog.text)

**ELSE**

**CAST**('' **AS** pg\_catalog.text)

**END** role\_attributes

**FROM** pg\_catalog.pg\_user

* **ORDER** **BY** role\_name **desc**;

Change postgres db owner:

ALTER DATABASE name OWNER TO new\_owner; (has to be altered by postgres ?? or original user??)

Postgres=# ALTER DATABASE “winn-apache2-1” OWNER TO winter;

Change database name:

**ALTER** **DATABASE** db **RENAME** **TO** newdb;

Postgres=# **ALTER** **DATABASE** “winn-apache2-1” **RENAME** **TO** winn\_la\_1; (la for linux apache2)

**Step 6 — Creating and Deleting Tables**

Now that you know how to connect to the PostgreSQL database system, you can learn some basic Postgres management tasks.

First, create a table to store some data. As an example, a table that describes some playground equipment.

The basic syntax for this command is as follows:

CREATE TABLE table\_name (

column\_name1 col\_type (field\_length) column\_constraints,

column\_name2 col\_type (field\_length),

column\_name3 col\_type (field\_length)

);

As you can see, these commands give the table a name, and then define the columns as well as the column type and the max length of the field data. You can also optionally add table constraints for each column.

You can learn more about [how to create and manage tables in Postgres](https://digitalocean.com/community/articles/how-to-create-remove-manage-tables-in-postgresql-on-a-cloud-server) here.

For demonstration purposes, create a simple table like this:

CREATE TABLE playground (

equip\_id serial PRIMARY KEY,

type varchar (50) NOT NULL,

color varchar (25) NOT NULL,

location varchar(25) check (location in ('north', 'south', 'west', 'east', 'northeast', 'southeast', 'southwest', 'northwest')),

install\_date date

);

These commands will create a table that inventories playground equipment. This starts with an equipment ID, which is of the serial type. This data type is an auto-incrementing integer. You’ve also given this column the constraint of primary key which means that the values must be unique and not null.

For two of the columns (equip\_id and install\_date), the commands do not specify a field length. This is because some column types don’t require a set length because the length is implied by the type.

The next two commands create columns for the equipment type and color respectively, each of which cannot be empty. The command after these creates a location column and create a constraint that requires the value to be one of eight possible values. The last command creates a date column that records the date on which you installed the equipment.

You can see your new table by typing:

* \d

Output

List of relations

Schema | Name | Type | Owner

--------+-------------------------+----------+-------

public | playground | table | sammy

public | playground\_equip\_id\_seq | sequence | sammy

(2 rows)

Your playground table is here, but there’s also something called playground\_equip\_id\_seq that is of the type sequence. This is a representation of the serial type which you gave your equip\_id column. This keeps track of the next number in the sequence and is created automatically for columns of this type.

If you want to see just the table without the sequence, you can type:

* \dt

Output

List of relations

Schema | Name | Type | Owner

--------+------------+-------+-------

public | playground | table | sammy

(1 row)

**Step 7 — Adding, Querying, and Deleting Data in a Table**

Now that you have a table, you can insert some data into it.

As an example, add a slide and a swing by calling the table you want to add to, naming the columns and then providing data for each column, like this:

* INSERT INTO playground (type, color, location, install\_date) VALUES ('slide', 'blue', 'south', '2017-04-28');
* INSERT INTO playground (type, color, location, install\_date) VALUES ('swing', 'yellow', 'northwest', '2018-08-16');

You should take care when entering the data to avoid a few common hangups. For one, do not wrap the column names in quotation marks, but the column values that you enter do need quotes.

Another thing to keep in mind is that you do not enter a value for the equip\_id column. This is because this is automatically generated whenever a new row in the table is created.

Retrieve the information you’ve added by typing:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+--------+-----------+--------------

1 | slide | blue | south | 2017-04-28

2 | swing | yellow | northwest | 2018-08-16

(2 rows)

Here, you can see that your equip\_id has been filled in successfully and that all of your other data has been organized correctly.

If the slide on the playground breaks and you have to remove it, you can also remove the row from your table by typing:

* DELETE FROM playground WHERE type = 'slide';

Query the table again:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+--------+-----------+--------------

2 | swing | yellow | northwest | 2018-08-16

(1 row)

You notice that your slide is no longer a part of the table.

**Step 8 — Adding and Deleting Columns from a Table**

After creating a table, you can modify it to add or remove columns relatively easily. Add a column to show the last maintenance visit for each piece of equipment by typing:

* ALTER TABLE playground ADD last\_maint date;

If you view your table information again, you will see the new column has been added (but no data has been entered):

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date | last\_maint

----------+-------+--------+-----------+--------------+------------

2 | swing | yellow | northwest | 2018-08-16 |

(1 row)

Deleting a column is just as simple. If you find that your work crew uses a separate tool to keep track of maintenance history, you can delete of the column by typing:

* ALTER TABLE playground DROP last\_maint;

This deletes the last\_maint column and any values found within it, but leaves all the other data intact.

**Step 9 — Updating Data in a Table**

So far, you’ve learned how to add records to a table and how to delete them, but this tutorial hasn’t yet covered how to modify existing entries.

You can update the values of an existing entry by querying for the record you want and setting the column to the value you wish to use. You can query for the “swing” record (this will match *every* swing in your table) and change its color to “red”. This could be useful if you gave the swing set a paint job:

* UPDATE playground SET color = 'red' WHERE type = 'swing';

You can verify that the operation was successful by querying the data again:

* SELECT \* FROM playground;

Output

equip\_id | type | color | location | install\_date

----------+-------+-------+-----------+--------------

2 | swing | red | northwest | 2010-08-16

(1 row)

As you can see, your slide is now registered as being red.

**Conclusion**

You are now set up with PostgreSQL on your Ubuntu 18.04 server. However, there is still *much* more to learn with Postgres. Here are some more guides that cover how to use Postgres: