**Document of Understanding on Postgre SQL**

**Topic Covered:**

* Postgre SQL Configurations
* Bind Address
* Administrative Tools
* Bind Address
* Authentication Methods
* Group & Roles
* Installing PostgreSQL on Ubuntu
* Performing Basic Check & Operation on the PostgreSQL

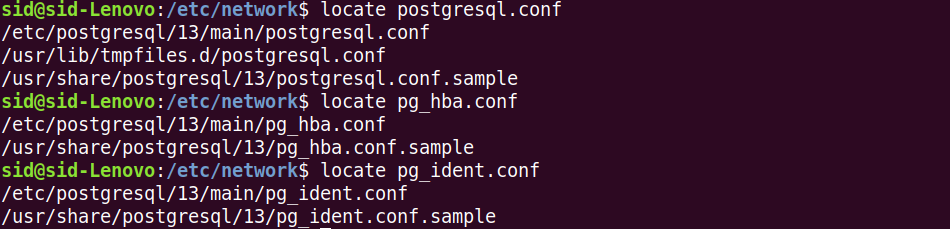
***Note: This document is in the development state and may contain incomplete information.***

**Postgre SQL Configurations**

Three main configuration files control basic operations of a PostgreSQL server instance. These files are all located in the default **PostgreSQL** data folder. You can edit them using your text editor of choice, or using the **admin** pack that comes with **pgAdmin**.

* **postgresql.conf** controls general settings, such as how much memory to allocate, default storage location for new databases, which IPs PostgreSQL listens on, where logs are stored, and so forth.
* **pg\_hba.conf** controls security. It manages access to the server, dictating which users can login into which databases, which IPs or groups of IPs are permitted to connect and the authentication scheme expected.
* **pg\_ident.conf** is the mapping file that maps an authenticated OS login to a PostgreSQL user. This file is used less often, but allows you to map a server account to a PostgreSQL account.   
  For example, people sometimes map the OS root account to the postgre’s super user account. Each authentication line in **pg\_hba.conf** can use a different ***pg\_ident.conf*** file.





**Bind Address**

In general, an address binding is an association between a service (e.g., SSH) and an IP address. A host may have multiple IP addresses (e.g., 127.0. 0.1, 192.168. 1.2). Address binding allows you to run a service on some or all of these addresses.

Suppose your host is configured with two network interfaces, one connected to a trusted network (e.g., 192.168.1.0/24), the other connected to an untrusted network (e.g., 192.168.2.0/24). Say you want your host to accept SSH connections from the trusted network only. In this case, you would bind the SSH service to the host's address on the trusted network only (e.g., 192.168.1.2).

Using localhost as bind address would allow only SSH clients running on the local machine to connect to the SSH service.

**Administrative Tools**

There are three popular tools for managing PostgreSQL and these are supported by PostgreSQL core developers; they tend to stay in synch with PostgreSQL versions.

**psql**

psql is a command-line interface for writing queries and managing PostgreSQL. It comes packaged with some nice extras, such as an import and export commands for delimited files, and a reporting feature that can generate HTML output. psql has been around since the beginning of PostgreSQL and is a favorite of hardcore PostgreSQL users. Newer converts who are more comfortable with GUI tools tend to favor pgAdmin.

**pgAdmin**

This is the widely used, free, graphical administration tool for PostgreSQL. You can download it separately from PostgreSQL. pgAdmin runs on the desktop and can connect to multiple PostgreSQL servers regardless of version or OS. Even if you have your database server on a window-less Unix-based server, install pgAdmin and you’ll find yourself armed with a fantastic GUI.

**PHPPgAdmin**

PHPPgAdmin is a free, web-based administration tool patterned after the popular PHPMyAdmin for MySQL. PostgreSQL has many more kinds of database objects than MySQL, as such PHPPgAdmin is a step up from PHPMyAdmin with additions to manage schemas, procedural languages, casts, operators, and so on. If you’ve used PHPMyAdmin, you’ll find PHPPgAdmin to be nearly identical.

**Authentication Methods**

PostgreSQL has many methods for authenticating users, probably more than any other database. Most people stick with the **four main ones: trust, ident, md5, and password**. There is also a **fifth one: reject**. which performs an immediate deny. Authentication methods stipulated in **pg\_hba.conf** serve as gatekeepers to the entire server. Users or devices must still satisfy individual role and database access restrictions after connecting.

* **trust** is the least secure of the authentication schemes and means you allow people to state who they are and don’t care about the passwords, if any, presented. As long as they meet the IP, user, and database criteria, they can connect. You really should use this only for local connections or private network connections. Even then it’s possible to have IPs spoofed, so the more security-minded among us discourage its use entirely.
* **md5** is the most common and means an md5-encrypted password is required.
* **password** means clear text password authentication.
* **ident** uses the **pg\_ident.conf** to see if the OS account of the user trying to connect has a mapping to a PostgreSQL account. Password is not checked.

**Group & Roles**

* In PostgreSQL, there is really only one kind of an account and that is a **role**.
* Some roles can log in; when they have login rights, they are called users. Roles can be members of other roles, and when we have this kind of relationship, the containing roles are called **groups**.
* It wasn’t always this way, though: Pre-8.0 users and groups were distinct entities, but the model got changed to be role-centric to better conform to the ANSI-SQL specs.
* If you look at fairly **ANSI-SQL** standard databases such as Oracle and later versions of SQL Server, you’ll notice they also have a **CREATE ROLE** statement, which works similarly as the PostgreSQL one.

**Creating an Account That Can Log In**

* **Postgres** is an account that is created when you first initialize the PostgreSQL data cluster. It has a companion database called postgres. Before you do anything else, you should login as this user via **psql** or **pgAdmin** and create other users. **pgAdmin** has a graphical section for creating user roles, but if you were to do it using standard SQL data control language (DCL)

**Creating Group Roles**

* Group roles are generally roles that have no login rights but have other roles as members. This is merely a convention. There is nothing stopping you from creating a role that can both login and can contain other roles.
* We can create a group role with this SQL DCL statement:

*CREATE ROLE jungle INHERIT;*

* And add a user or other group role to the group with this statement:

*GRANT jungle TO leo;*

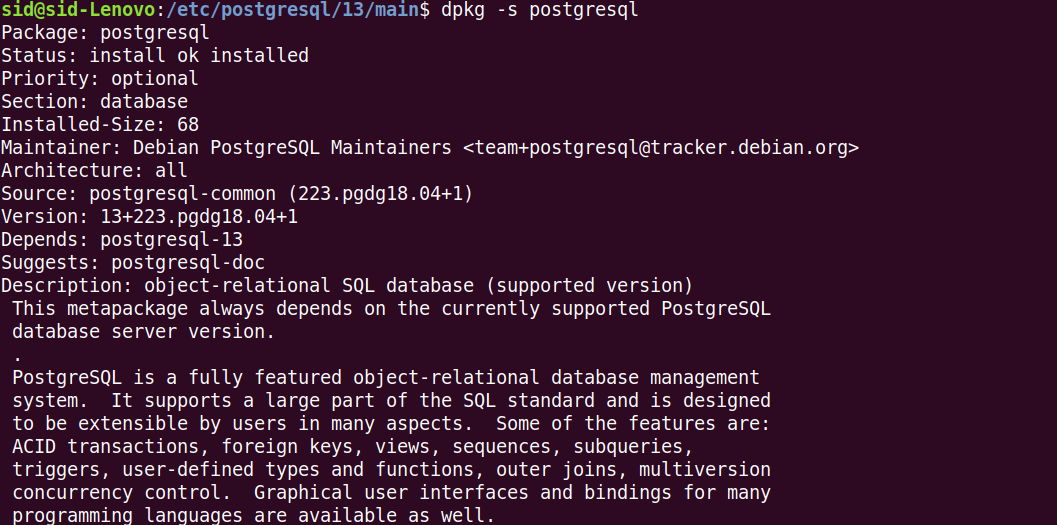
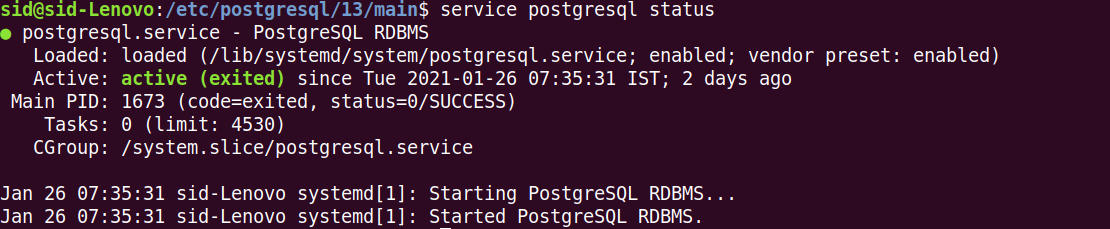
Master-Slave Architecture

Installing PostgreSQL on Ubuntu

To install the PostgreSQL on Ubuntu we can follow the below steps:

* Create the file repository configuration:  
  *sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb\_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'*
* Import the repository signing key:  
  *wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -*
* Update the package lists:  
  *sudo apt-get update*
* Install the latest version of PostgreSQL. If you want a specific version, use 'postgresql-12' or similar instead of 'postgresql':  
  *sudo apt-get -y install postgresql13*

Performing Basic Check & Operation on the PostgreSQL

* PostgreSQL package verification.  
    
  *dpkg -s postgresql*  
    
  
* Checking PostgreSQL services are up and running.   
    
  *service postgresql status*  
    
  
* Connecting to PostgreSQL DB server.  
    
  *sudo su postgres – to change the from sid to postgres (which is a default system user for PostgreSQL)  
    
  psql – to open the Postgre SQL interactive terminal  
    
  \l – lists the number of DBs present by default*  
  