PostgreSQL Lab Manual

**CHAPTER - 1 OS Installation and Post RPM Installation**

**CHAPTER - 2 PostgreSQL Instalaltion (source + rpm + yum)**

**CHAPTER - 3 Post Installation Steps**

**CHAPTER - 4 Alert logging + Benchmarking.**

**CHAPTER - 5 Walfiles + Datafiles + Controlfile**

**CHAPTER - 6 Host Based Authentication file**

**CHAPTER - 7 Create database, user and schema and grant privileges**

**CHAPTER - 8 Logical Backups**

**CHAPTER - 9 Physical Backups (4 Activities)**

**CHAPTER - 10 Upgrade your database**

**CHAPTER - 11 EDB Software installation, PGBadger Installation, PGAdmin Installation**

CHAPTER - 1

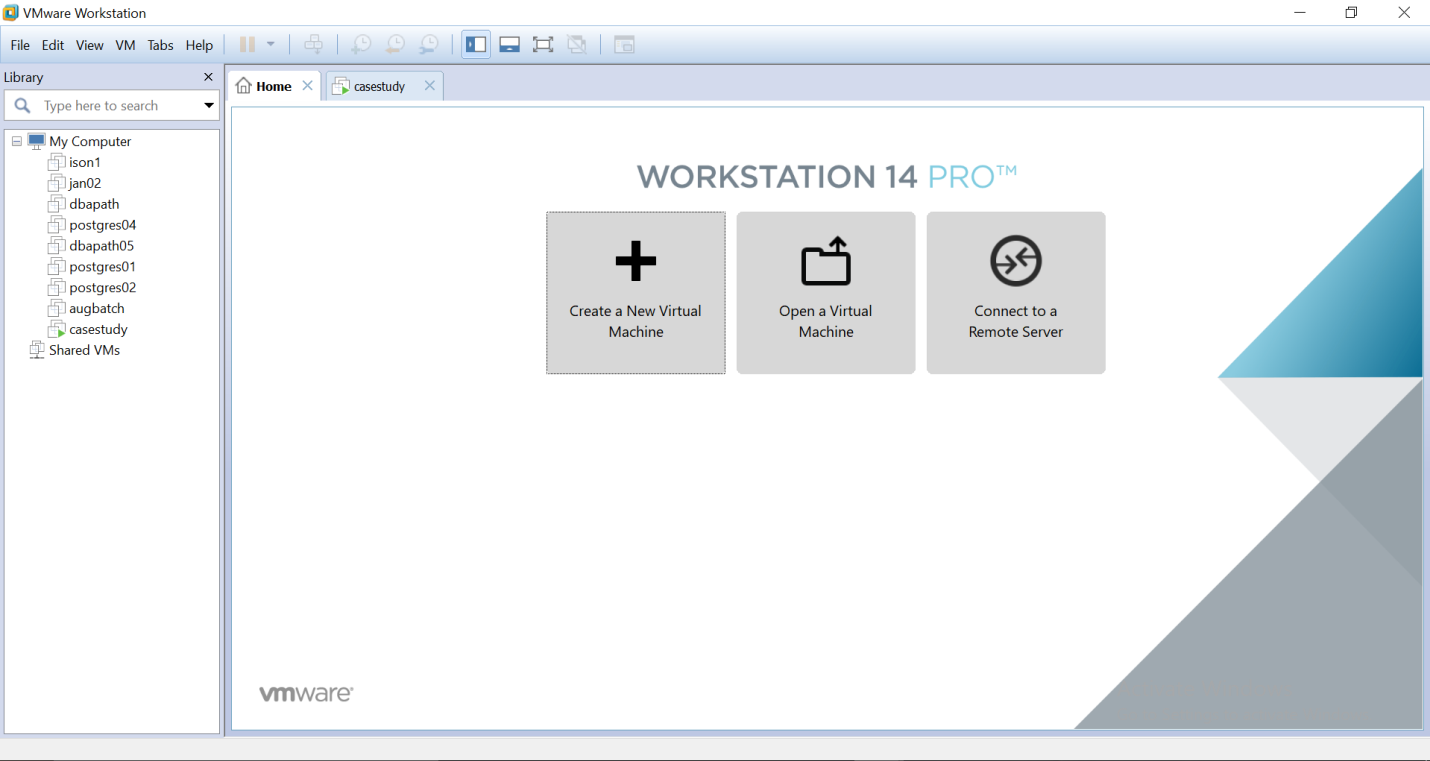
Download VMWare Workstation 16 Pro

Download Operating System (OEL 7)

**How to install VM Ware Workstation 16 Pro?**

**Double click on executable file., Next-Next-Finish.**

Once installation is success, interface you should see something like this.



**How to install Operating system?**

https://drive.google.com/file/d/1CZMt9SA3d-dqWkRuyNq8ksPpcSpwuEW4/view?usp=sharing

**What are the Post Installation Steps ?**

hine.

<https://drive.google.com/file/d/1CFgoWcEucn1y7jl-OG1rKp4NgouyGZZw/view?usp=sharing>

Execute all the below commands to get started with virtual mac

**Video**: Enable internet in your virtual machine and execute below steps

**yum-config-manager –disable centrify.repo**

**yum clean all**

**yum update yum**

**yum update curl**

**yum update openssl**

**yum update nss**

**yum-config-manager –enable centrify.repo**

CHAPTER - 2

**How to install PostgreSQL using Source code.**

Reference : <https://postgreshelp.com/how-to-install-postgresql-in-linux/>

Video : https://drive.google.com/file/d/1HbXGKWH8tv-LL94tYXBAqwwMsaBiYzGL/view?usp=sharing

**Steps: Pre-requisites**

1. Choose the version to download from here

<https://www.postgresql.org/ftp/source/>

1. Copy the software to your virtual machine (use winscp)
2. Extract the software using gunzip .
   1. **[root@casestudy ~]# gunzip postgresql-13.0.tar.gz**
   2. **[root@casestudy ~]# tar -xf postgresql-13.0.tar**
3. Pre-requisites to install PostgreSQL

**make –-version #Make sure the version is > 3.81(1)**

**rpm -qa gcc\* #If not found, yum install gcc\*(2+ repos)**

**rpm -qa readline #If not found, yum install readline\*(2repos)**

**rpm -qa zlib\* #If not found, yum install zlib\*(2repos)**

**Steps: Source installation**

1. Install the database software (**cd postgresql-13.0**)

**./configure**

**make**

**make install**

1. **Initialize and create your cluster.**

**useradd postgres**

**passwd postgres**

**postgres**

**postgres**

**mkdir –p /u01**

**chown -R postgres:postgres /u01**

**su – postgres**

**mkdir -p /u01/pgsql/13**

**chmod 700 /u01/pgsql/13**

**/usr/local/pgsql/bin/initdb –D /u01/pgsql/13**

**/usr/local/pgsql/bin/pg\_ctl start –D /u01/pgsql/13**

**/usr/local/pgsql/bin/psql**

**Setting up bash profile [edit your bash profile by adding colored contents]**

**[postgres@casestudy ~]$ cat .bash\_profile**

**# .bash\_profile**

**# Get the aliases and functions**

**if [ -f ~/.bashrc ]; then**

**. ~/.bashrc**

**fi**

**# User specific environment and startup programs**

**PHOME=/usr/pgsql-14**

**PATH=$PHOME/bin:$PATH:$HOME/bin**

**export PATH**

**EXAMPLE BASH PROFILE FOR v14 Installation**

| **[postgres@lab01 ~]$ cat .bash\_profile**  **# .bash\_profile**  **# Get the aliases and functions**  **if [ -f ~/.bashrc ]; then**  **. ~/.bashrc**  **fi**  **# User specific environment and startup programs**  **PHOME=/usr/pgsql-14**  **PATH=$PHOME/bin:$PATH:$HOME/.local/bin:$HOME/bin**  **export PATH** |
| --- |

**Make the changes to reflect by refreshing bash profile**

[postgres@casestudy ~]$ **. .bash\_profile**

[postgres@casestudy ~]$ **psql**

psql (13.0)

Type "help" for help.

postgres=#

**RPM Installation**

1. Browse required rpms from
   1. <https://yum.postgresql.org/>
2. For instance, v12 rpms can be found here
   1. <https://yum.postgresql.org/12/redhat/rhel-6-x86_64/repoview/>
3. Download the following rpms and copy them to your virtual machine
   1. postgresql12-12.4-1PGDG.rhel6
   2. postgresql12-contrib-12.4-1PGDG.rhel6
   3. postgresql12-devel-12.4-1PGDG.rhel6
   4. postgresql12-libs-12.4-1PGDG.rhel6
   5. postgresql12-server-12.4-1PGDG.rhel6
4. Install the rpms with
   1. **rpm –ivh <rpm\_name.rpm>.**
5. The installation sequence is as follows

**rpm –ivh postgresql12-libs-12.4-1PGDG.rhel6.x86\_64.rpm**

**rpm –ivh postgresql12-12.4-1PGDG.rhel6.x86\_64.rpm**

**rpm –ivh postgresql12-server-12.4-1PGDG.rhel6.x86\_64.rpm**

**rpm –ivh postgresql12-contrib-12.4-1PGDG.rhel6.x86\_64.rpm**

**rpm –ivh postgresql12-devel-12.4-1PGDG.rhel6.x86\_64.rpm**

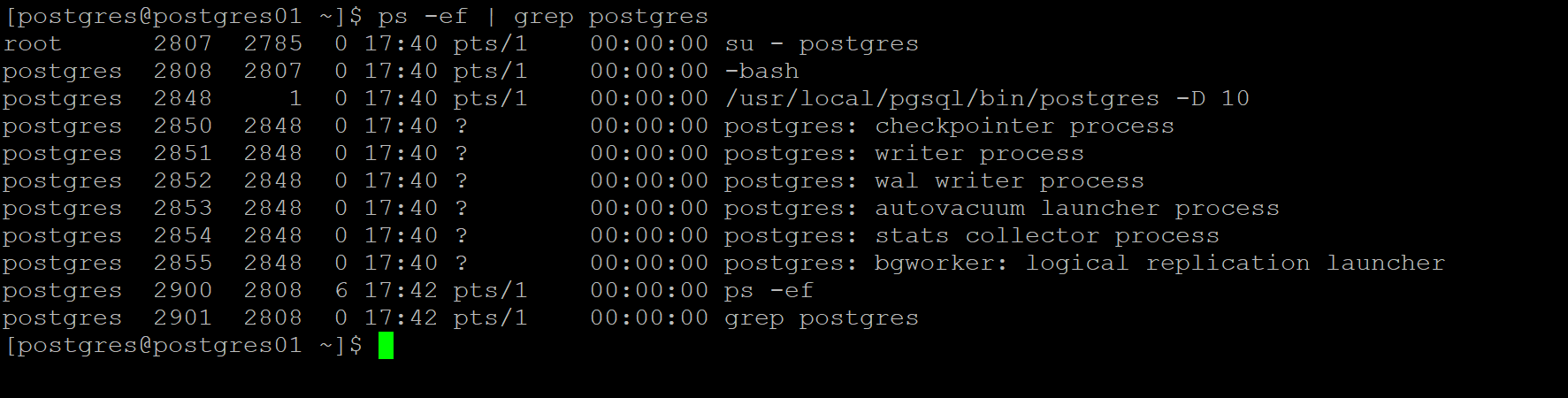
**Yum Installation**

Source : https://postgreshelp.com/postgresql-10-install-in-linux/

1. **yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-6-x86\_64/pgdg-redhat-repo-latest.noarch.rpm**
2. **yum install postgresql11\***
3. Initialize the cluster and start the database.[initdb+pgctl+psql]

Upon successful installation, confirm the installation with

ps –ef | grep postgres



**CHAPTER -3**

Post PostgreSQL Installation steps

1. Change password.

[postgres@postgres01 ~]$ psql

psql (10.4)

Type "help" for help.

postgres=# **\password**

Enter new password:

Enter it again:

postgres=#

1. Edit postgresql.conf file with

**listen\_addresses = '\*' # (change requires restart)**

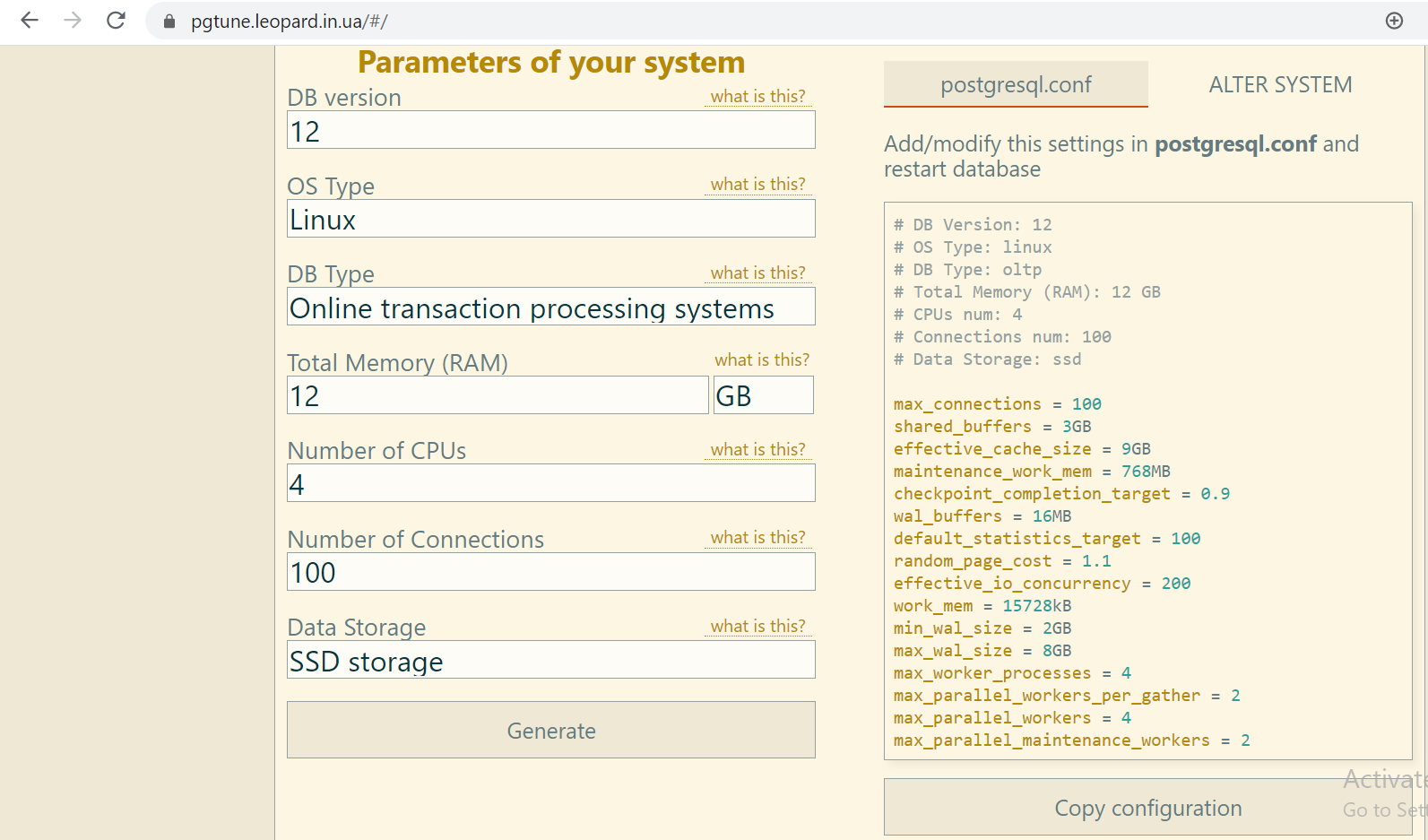
1. Edit pg\_hba.conf with

**host all all 0.0.0.0/0 md5 #reload**

1. Restart the cluster with below command.

**pg\_ctl restart -D /u01/pgsql/10**

1. **Use pgtune to set parameters.**
2. Login to the below website
3. https://pgtune.leopard.in.ua/#/
4. Enter your operating system resource details and get parameter details
5. Make changes in postgresql.conf file.



Below are the parameters you need to set in postgresql.conf file

max\_connections = 100

shared\_buffers = 3GB

effective\_cache\_size = 9GB

maintenance\_work\_mem = 768MB

checkpoint\_completion\_target = 0.9

wal\_buffers = 16MB

default\_statistics\_target = 100

random\_page\_cost = 1.1

effective\_io\_concurrency = 200

work\_mem = 15728kB

min\_wal\_size = 2GB

max\_wal\_size = 8GB

max\_worker\_processes = 4

max\_parallel\_workers\_per\_gather = 2

max\_parallel\_workers = 4

max\_parallel\_maintenance\_workers = 2

Which parameter needs restart?

**postgres=# select distinct context from pg\_settings;**

**context**

**-------------------**

**Postmaster #Need restart**

**superuser-backend #Only super user can change**

**user #Any super user can change with SET**

**internal #Can only be changed with rebuild**

**backend #session level reset**

**sighup #reload**

**superuser #Only super user can change with SET**

(7 rows)

Example.,

**select name, setting,context from pg\_settings order by context;**

**name | setting | context**

**------------------------------------+--------------------------**

**post\_auth\_delay | 0 | backend (POSTGRESQL.CONF)**

**block\_size | 8192 | internal (CAN’T DO)**

**listen\_addresses | \* | postmaster (RESTART)**

**log\_checkpoints | off | sighup (RELOAD)**

**log\_lock\_waits | off | superuser (SET)**

**log\_connections | off | superuser-backend**

**work\_mem | 4096 | user (SET)**

**CHAPTER -4**

How do you enable alert logging and what are the mandatory parameters?

*Set the following parameters in postgresql.conf*

**log\_destination = 'stderr'**

**logging\_collector = on**

**log\_directory = 'log'**

**log\_filename = 'postgresql-%d-%m-%Y.log'**

**log\_line\_prefix = '%t [%p]: [%l-1] user=%u,db=%d,app=%a,client=%h '**

**log\_checkpoints = on**

**log\_connections = on**

**log\_disconnections = on**

**log\_lock\_waits = on**

**log\_temp\_files = 0**

**log\_autovacuum\_min\_duration = 0**

**log\_error\_verbosity = default**

**log\_statement = off**

**lc\_messages='C'**

**Restart the cluster.**

Confirm the changes.

**postgres=# show logging\_collector**

**postgres-# ;**

**logging\_collector**

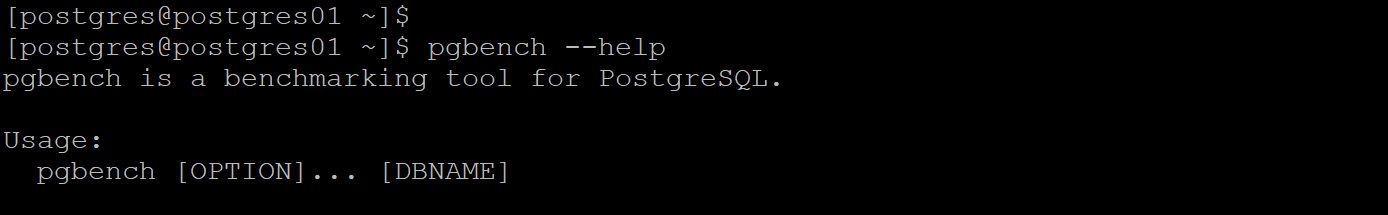
**-------------------**

**on**

**(1 row)**

**postgres=#**

**Benchmarking (pgbench)**



**How to initialize tables with pgbench**

**pgbench -i -s 5 postgres**

[postgres@postgres01 ~]$ pgbench -i -s 5 postgres

creating tables...

100000 of 500000 tuples (20%) done (elapsed 0.16 s, remaining 0.65 s)

200000 of 500000 tuples (40%) done (elapsed 0.39 s, remaining 0.59 s)

300000 of 500000 tuples (60%) done (elapsed 0.57 s, remaining 0.38 s)

400000 of 500000 tuples (80%) done (elapsed 0.77 s, remaining 0.19 s)

500000 of 500000 tuples (100%) done (elapsed 1.10 s, remaining 0.00 s)

vacuum...

set primary keys...

done.

How to benchmark with pgbench with 20 clients doing 100 transactions each on postgresql database.

**pgbench -c 20 -t 100 postgres**

**[postgres@postgres01 ~]$ pgbench -c 20 -t 100 postgres**

**starting vacuum...end.**

**transaction type: <builtin: TPC-B (sort of)>**

**scaling factor: 5**

**query mode: simple**

**number of clients: 20**

**number of threads: 1**

**number of transactions per client: 100**

**number of transactions actually processed: 2000/2000**

**latency average = 45.096 ms**

**tps = 443.500071 (including connections establishing)**

**tps = 444.016123 (excluding connections establishing)**

**CHAPTER -5**

Write – Ahead-Logging

The WAL File is stored under PG\_DATA/pg\_wal

The first file that is created is 000000010000000000000001.

If the first one has been filled up with the writing of XLOG records, the second one 000000010000000000000002 would be provided.

Files of successor is used in ascending order in succession, after 0000000100000000000000FF has been filled up, next one 000000010000000100000000 will be provided. In this way, whenever the last 2-digit carries over, the middle 8-digit number increases one.

How to view current wal write location?

postgres=# select pg\_current\_wal\_lsn();

pg\_current\_wal\_lsn

--------------------

0/546CAA0

(1 row)

How to view current wal file name

postgres=# select pg\_walfile\_name('0/546CAA0');

pg\_walfile\_name

--------------------------

000000010000000000000005

(1 row)

How to switch wal ?

postgres=#

postgres=# select pg\_switch\_wal();

pg\_switch\_wal

---------------

0/546CAB8

(1 row)

**How to view wal content?**

[postgres@postgres01 pg\_wal]$ **pg\_waldump --help**

pg\_waldump decodes and displays PostgreSQL write-ahead logs for debugging.

Usage:

pg\_waldump [OPTION]... [STARTSEG [ENDSEG]]

Options:

-b, --bkp-details output detailed information about backup blocks

-e, --end=RECPTR stop reading at WAL location RECPTR

-f, --follow keep retrying after reaching end of WAL

-n, --limit=N number of records to display

-p, --path=PATH directory in which to find log segment files or a

directory with a ./pg\_wal that contains such files

(default: current directory, ./pg\_wal, $PGDATA/pg\_wal)

-r, --rmgr=RMGR only show records generated by resource manager RMGR;

use --rmgr=list to list valid resource manager names

-s, --start=RECPTR start reading at WAL location RECPTR

-t, --timeline=TLI timeline from which to read log records

(default: 1 or the value used in STARTSEG)

-V, --version output version information, then exit

-x, --xid=XID only show records with transaction ID XID

-z, --stats[=record] show statistics instead of records

(optionally, show per-record statistics)

-?, --help show this help, then exit

pg\_waldump displays the write-ahead log (WAL) and is mainly useful for debugging or educational purposes.

This utility can only be run by the user who installed the server, because it requires read-only access to the data directory.

[postgres@postgres01 pg\_wal]$

[postgres@postgres01 pg\_wal]$ psql -c "select pg\_current\_wal\_lsn(),pg\_walfile\_name(pg\_current\_wal\_lsn()),pg\_walfile\_name\_offset(pg\_current\_wal\_lsn())";

pg\_current\_wal\_lsn | pg\_walfile\_name | pg\_walfile\_name\_offset

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

0/60135C0 | 000000010000000000000006 | (000000010000000000000006,79296)

(1 row)

[postgres@postgres01 pg\_wal]$ **psql -c "insert into emp values ('2','B')";**

INSERT 0 1

[postgres@postgres01 pg\_wal]$ **pg\_waldump -s 0/60135C0 000000010000000000000006**

rmgr: Heap len (rec/tot): 61/ 61, tx: 1574, lsn: 0/060135C0, prev 0/06013588, desc: INSERT off 2, blkref #0: rel 1663/13164/16404 blk 0

rmgr: Transaction len (rec/tot): 34/ 34, tx: 1574, lsn: 0/06013600, prev 0/060135C0, desc: COMMIT 2020-10-01 18:45:10.687383 IST

rmgr: Standby len (rec/tot): 50/ 50, tx: 0, lsn: 0/06013628, prev 0/06013600, desc: RUNNING\_XACTS nextXid 1575 latestCompletedXid 1574 oldestRunningXid 1575

**pg\_waldump: FATAL: error in WAL record at 0/6013628: invalid record length at 0/6013660: wanted 24, got 0**

**Complete wal dump example**

[postgres@postgres01 pg\_wal]$

Simple example.,

**pg\_waldump 000000010000000000000006**

Datafiles

<https://postgreshelp.com/postgresql-page-layout/>

**+----------------+---------------------------------+**

**| PageHeaderData | linp1 linp2 linp3 ...           |**

**+-----------+----+---------------------------------+**

**| ... linpN |                                      |**

**+-----------+--------------------------------------+**

**|           ^ pd\_lower                             |**

**|                                                  |**

**|             v pd\_upper                           |**

**+-------------+------------------------------------+**

**|             | tupleN ...                         |**

**+-------------+------------------+-----------------+**

**|       ... tuple3 tuple2 tuple1 | "special space" |**

**+--------------------------------+-----------------+**

**^ pd\_special**

postgres=# **select pg\_relation\_filepath('emp');**

pg\_relation\_filepath

----------------------

base/13164/16404

(1 row)

postgres=# \q

[postgres@postgres01 ~]$ **cd /u01/pgsql/10/base/13164**

[postgres@postgres01 13164]$ **od -a 16404**

0000000 nul nul nul nul nul 6 soh ack nul nul nul nul sp nul @ us

0000020 nul sp eot sp nul nul nul nul ` us < nul @ us < nul

0000040 nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul nul

\*

0017700 & ack nul nul nul nul nul nul nul nul nul nul nul nul nul nul

0017720 stx nul stx nul stx ht can nul stx nul nul nul enq B nul nul

0017740 % ack nul nul nul nul nul nul nul nul nul nul nul nul nul nul

0017760 soh nul stx nul stx ht can nul soh nul nul nul enq A nul nul

0020000

[postgres@postgres01 13164]$

Control file

**pg\_controldata -D /u01/pgsql/10**

Output excerpt

**[postgres@postgres01 ~]$ pg\_controldata -D /u01/pgsql/10**

**pg\_control version number: 1002**

**Catalog version number: 201707211**

**Database system identifier: 6878623138754341658**

**Database cluster state: in production**

**pg\_control last modified: Thu 01 Oct 2020 06:52:46 PM IST**

**Latest checkpoint location: 0/6047528**

**Prior checkpoint location: 0/6013698**

**Latest checkpoint's REDO location: 0/60474F0**

**..**

**CHAPTER -6**

Host Based Autentication file (pg\_hba.conf)

How do you enable access to external connections with md5 authentication?

host all all 0.0.0.0/0 md5

How do you restrict DEMOUSER database to reject connections?

local all DEMOUSER 0.0.0.0/0 reject

host all all 0.0.0.0/0 md5

**[postgres@postgres01 10]$ psql -U demouser -d postgres**

**psql: FATAL: pg\_hba.conf rejects connection for host "[local]", user "demouser", database "postgres"**

**[postgres@postgres01 10]$**

How do you restrict DEMOUSER database to reject connections if it is from 192.168.1.4?

host all DEMOUSER 192.168.1.4 reject

host all all 0.0.0.0/0 md5

How do you ask DEMOUSER to authenticate and rest ignore authentication?

host all DEMOUSER 0.0.0.0/0 md5

host all all 0.0.0.0/0 trust

How do you set up hba configuration for replica user to take backup from remote machine?

host replication replica 0.0.0.0/0 md5

**CHAPTER -7**

Three things you do when creating a database.

**create user demo\_user with password 'demo\_user';**

**create database demo\_db with owner demo\_user;**

**\q**

**psql -U demo\_user -d demo\_db**

**create schema demo\_user;**

**activity log**

postgres=# create user demo\_user with password 'demo\_user'

postgres-# ;

CREATE ROLE

postgres=# create database demo\_db with owner demo\_user;

CREATE DATABASE

postgres=# \q

[postgres@postgres01 10]$ psql -U demo\_user -d demo\_db

psql (10.4)

Type "help" for help.

demo\_db=> create schema demo\_user;

CREATE SCHEMA

demo\_db=>

[ACTIVITY]

Create two roles, dba and developer and create two developers and two dba’s and now assign read, write permission on production tables to dba role and read only access to developer role.

Also, track the user activity in log.

*[I already have a database “airportdb” with owner “airport” and has a table “emp” in airport schema]*

Create role airport\_read;

Create role airport\_ps;

GRANT CONNECT ON DATABASE airportdb TO airport\_read;

GRANT USAGE ON SCHEMA airport TO airport\_read;

GRANT SELECT ON ALL TABLES IN SCHEMA airport TO airport\_read;

ALTER DEFAULT PRIVILEGES IN SCHEMA airport GRANT SELECT ON TABLES TO airport\_read;

GRANT CONNECT ON DATABASE airportdb TO airport\_ps;

GRANT USAGE ON SCHEMA airport TO airport\_ps;

GRANT ALL ON ALL TABLES IN SCHEMA airport TO airport\_ps;

ALTER DEFAULT PRIVILEGES IN SCHEMA airport GRANT ALL ON TABLES TO airport\_ps;

GRANT ALL ON ALL SEQUENCES IN SCHEMA airport TO airport\_ps;

ALTER DEFAULT PRIVILEGES IN SCHEMA airport GRANT ALL ON SEQUENCES TO airport\_ps;

GRANT ALL ON ALL FUNCTIONS IN SCHEMA airport TO airport\_ps;

ALTER DEFAULT PRIVILEGES IN SCHEMA airport GRANT ALL ON FUNCTIONS TO airport\_ps;

create user ab\_dba1 with password 'ab\_dba1';

create user ab\_dba2 with password 'ab\_dba2';

create user ab\_dev1 with password 'ab\_dev1';

create user ab\_dev2 with password 'ab\_dev2';

grant airport\_ps to ab\_dba1;

grant airport\_ps to ab\_dba2;

grant airport\_read to ab\_dev1;

grant airport\_read to ab\_dev2;

alter user ab\_dba1 set search\_path=airport, public,ab\_dba1;

alter user ab\_dba2 set search\_path=airport, public,ab\_dba2;

alter user ab\_dev1 set search\_path=airport, public,ab\_dev1;

alter user ab\_dev2 set search\_path=airport, public,ab\_dev2;

alter user ab\_dba1 set log\_statement='all';

alter user ab\_dba2 set log\_statement='all';

alter user ab\_dev1 set log\_statement='all';

alter user ab\_dev2 set log\_statement='all';

[LOGIN AND CHECK IF EVERYTHING IS FINE]

**CHAPTER -8**

PostgreSQL Logical Backup

Demo tables

demo=> \dp

Schema | Name | Type |

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

demouser | aircrafts | view |

demouser | aircrafts\_data | table |

demouser | airports | view |

demouser | airports\_data | table |

demouser | boarding\_passes | table |

demouser | bookings | table |

demouser | emp | table |

demouser | flights | table |

demouser | flights\_flight\_id\_seq | sequence |

demouser | flights\_v | view |

demouser | routes | view |

demouser | seats | table |

demouser | ticket\_flights | table |

demouser | tickets | table |

(14 rows)

Take flights table backup with pg\_dump in plain text format.

**pg\_dump -U demouser -d demo -t flights> demo\_plain.sql**

Take flights table backup with pg\_dump in plain text format with –Fp option.

**pg\_dump -U demouser -d demo -Fp -t flights -f demo\_plain.bkp**

Take flights table backup with pg\_dump in tar format with –Ft option.

**pg\_dump -U demouser -d demo -Ft -t flights -f demo\_plain.tar**

Take flights table backup with pg\_dump in custorm format with –Fc option.

**pg\_dump -U demouser -d demo -Fc -t flights -f demo\_plain.custom**

Take flights, seats table backup with pg\_dump in directory format with –Fd option.

**pg\_dump -U demouser -d demo -Fd -t flights -t sets -f twotables**

Take flights, seats table backup with pg\_dump in directory format with 4 parallel processes.

**pg\_dump -U demouser -d demo -Fd -t flights -t sets -f parallel4 -j 4**

Take flights table backup with pg\_dump in custorm format with 4 compression.

**pg\_dump -U demouser -d demo -Fc -t flights -f demo\_plain.custom -Z 3**

**-a, --data-only**

**pg\_dump -U demouser -d demo -t emp -a > dataonly.sql**

**pg\_dump -U demouser -d demo -t emp > includetablecreation.sql**

[postgres@postgres01 dumps]$ diff includetablecreation.sql dataonly.sql

18,33d17

< SET default\_tablespace = '';

<

< SET default\_with\_oids = false;

<

< --

< -- Name: emp; Type: TABLE; Schema: demouser; Owner: demouser

< --

<

< CREATE TABLE demouser.emp (

< id integer,

< name character varying(10)

< );

<

<

< ALTER TABLE demouser.emp OWNER TO demouser;

<

**-c option :**

**pg\_dump -U demouser -d demo -t emp -c > coption.sql**

DROP TABLE demouser.emp;

SET default\_tablespace = '';

SET default\_with\_oids = false;

--

-- Name: emp; Type: TABLE; Schema: demouser; Owner: demouser

--

CREATE TABLE demouser.emp (

id integer,

name character varying(10)

);

ALTER TABLE demouser.emp OWNER TO demouser;

--

-- Data for Name: emp; Type: TABLE DATA; Schema: demouser; Owner: demouser

--

COPY demouser.emp (id, name) FROM stdin;

1 A

2 B

3 C

\.

**-n, --schema=SCHEMA**

**pg\_dump -U demouser -d demo -n demouser > onlydemouser.sql**

**-O, --no-owner** skip restoration of object ownership in plain-text format

**pg\_dump -U demouser -d demo -t emp -O > owner.sql**

ALTER TABLE demouser.emp OWNER TO demouser; (DON’T WRITE THIS LINE)

Take entire database metadata backup in custom format

**pg\_dump -U demouser -d demo -Fc -s -f fullmeta.bkp**

Take entire database backup in custom format

**pg\_dump -U demouser -d demo -Fc -f fulldata.bkp**

Restore commands

Restore table flights from demo\_plain.custom

**pg\_restore -U demouser -d demo demo\_plain.custom**

**check :**

psql -U demouser -d demo -c "select count(1) from flights";

Create a dababase demoview and import all functions from backup fullmeta.bkp

**psql -c "create database demoview with owner demouser";**

**psql -U demouser -d demoview -c "create schema demouser";**

**pg\_restore -l fullmeta.bkp | grep FUNCTION > function\_list**

**pg\_restore -U demouser -d demoview -L function\_list fullmeta.bkp**

Log

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$ psql -c "create database demoview with owner demouser";*

*CREATE DATABASE*

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$ psql -U demouser -d demoview -c "create schema demouser";*

*CREATE SCHEMA*

*[postgres@postgres01 dumps]$ pg\_restore -l fullmeta.bkp | grep FUNCTION > function\_list*

*[postgres@postgres01 dumps]$* ***pg\_restore -U demouser -d demoview -L function\_list fullmeta.bkp***

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$*

*[postgres@postgres01 dumps]$ psql -U demouser -d demoview*

*psql (10.4)*

*Type "help" for help.*

*demoview=>*

*demoview=>*

*demoview=>* ***\df***

*List of functions*

*Schema | Name | Result data type | Argument data types | Type*

*----------+------+------------------+---------------------+--------*

*demouser | lang | text | | normal*

*(1 row)*

*demoview=>*

Restore a single table from backup

**pg\_restore -U demouser -d demoview -t flights fulldata.bkp**

log

[postgres@postgres01 dumps]$ **pg\_restore -U demouser -d demoview -t flights fulldata.bkp**

[postgres@postgres01 dumps]$

[postgres@postgres01 dumps]$

[postgres@postgres01 dumps]$

[postgres@postgres01 dumps]$ psql -U demouser -d demoview

psql (10.4)

Type "help" for help.

demoview=> \dt+

List of relations

Schema | Name | Type | Owner | Size | Description

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

demouser | flights | table | demouser | 21 MB |

(1 row)

demoview=>

**CHAPTER -9**

Physical backup

Activity 1 : Take a full backup and restore

**pg\_basebackup -D /u01/backrest -Ft --checkpoint=fast –P**

[postgres@postgres01 10]$

[postgres@postgres01 10]$ **pg\_basebackup -D /u01/backrest -Ft --checkpoint=fast -P**

214000/214000 kB (100%), 1/1 tablespace

[postgres@postgres01 10]$ **cd /u01/backrest**

[postgres@postgres01 backrest]$ **ls -lrt**

total 230396

-rw-rw-r--. 1 postgres postgres 219137536 Oct 1 22:57 base.tar

-rw-------. 1 postgres postgres 16779264 Oct 1 22:57 pg\_wal.tar

[postgres@postgres01 backrest]$ **tar -xf base.tar**

[postgres@postgres01 backrest]$ **tar -xf pg\_wal.tar**

[postgres@postgres01 backrest]$ **mv 0000000\* pg\_wal**

[postgres@postgres01 backrest]$

[postgres@postgres01 backrest]$

[postgres@postgres01 backrest]$

[postgres@postgres01 backrest]$ **rm -rf base.tar pg\_wal.tar**

[postgres@postgres01 backrest]$

**Start the cluster**

[postgres@postgres01 backrest]$ pg\_ctl start -D /u01/pgsql/10

waiting for server to start....2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= LOG: listening on IPv4 address "0.0.0.0", port 5432

2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= LOG: listening on IPv6 address "::", port 5432

2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= LOG: listening on Unix socket "/tmp/.s.PGSQL.5432"

2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= LOG: redirecting log output to logging collector process

2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= HINT: Future log output will appear in directory "log".

done

server started

[postgres@postgres01 backrest]$

Alert log

2020-10-01 22:59:18 IST [18456]: db=,user=,app=,client= LOG: database system was shut down at 2020-10-01 22:58:40 IST

2020-10-01 22:59:18 IST [18454]: db=,user=,app=,client= LOG**: database system is ready to accept connections**

**Activity 2 : Take full backup and restore with few archive logs generated**

1. Take a full backup

**pg\_basebackup -D /u01/backrest -Ft --checkpoint=fast –P**

1. Create few archive logs

**pgbench -i -s 10 postgres**

1. Prepare backup restore location

**cd backrest/**

**tar -xf base.tar**

**tar -xf pg\_wal.tar**

**mv 0000000\* pg\_wal**

**chmod 700 /u01/backrest**

1. Create a recovery.file inside new cluster extracted location with below parameter.

**restore\_command='cp /u01/archivelogs/%f %p'**

1. Start the cluster.

**pg\_ctl start -D /u01/backrest**

**Alert log information**

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: database system was interrupted; last known up at 2020-10-01 23:13:37 IST

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: starting archive recovery

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000D6" from archive

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: redo starts at 0/D6000028

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: consistent recovery state reached at 0/D60000F8

2020-10-01 23:15:21 IST [18782]: db=,user=,app=,client= LOG: database system is ready to accept read only connections

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000D7" from archive

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000D8" from archive

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000D9" from archive

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DA" from archive

2020-10-01 23:15:21 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DB" from archive

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DC" from archive

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DD" from archive

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DE" from archive

cp: cannot stat `/u01/archivelogs/0000000100000000000000DF': No such file or directory

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: redo done at 0/DEB1FC90

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: last completed transaction was at log time 2020-10-01 23:13:53.206534+05:30

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: restored log file "0000000100000000000000DE" from archive

cp: cannot stat `/u01/archivelogs/00000002.history': No such file or directory

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: selected new timeline ID: 2

cp: cannot stat `/u01/archivelogs/00000001.history': No such file or directory

2020-10-01 23:15:22 IST [18784]: db=,user=,app=,client= LOG: archive recovery complete

2020-10-01 23:15:22 IST [18786]: db=,user=,app=,client= LOG: checkpoint starting: end-of-recovery immediate wait

2020-10-01 23:15:23 IST [18786]: db=,user=,app=,client= LOG: checkpoint complete: wrote 16084 buffers (98.2%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.583 s, sync=0.024 s, total=0.612 s; sync files=35, longest=0.022 s, average=0.000 s; distance=147456 kB, estimate=147456 kB

2020-10-01 23:15:23 IST [18782]: db=,user=,app=,client= LOG: database system is ready to accept connections

**Activity 3: archive + pitr**

1. Take full backup

[postgres@postgres01 backrest]$ pg\_basebackup -D /u01/backrest -Ft --checkpoint=fast -P

213320/213320 kB (100%), 1/1 tablespace

1. Create few archivelogs

[postgres@postgres01 backrest]$ pgbench -i -s 10 postgres

creating tables...

100000 of 1000000 tuples (10%) done (elapsed 0.18 s, remaining 1.59 s)

200000 of 1000000 tuples (20%) done (elapsed 0.37 s, remaining 1.47 s)

300000 of 1000000 tuples (30%) done (elapsed 0.58 s, remaining 1.35 s)

400000 of 1000000 tuples (40%) done (elapsed 0.93 s, remaining 1.40 s)

500000 of 1000000 tuples (50%) done (elapsed 1.15 s, remaining 1.15 s)

600000 of 1000000 tuples (60%) done (elapsed 1.50 s, remaining 1.00 s)

700000 of 1000000 tuples (70%) done (elapsed 1.82 s, remaining 0.78 s)

800000 of 1000000 tuples (80%) done (elapsed 2.15 s, remaining 0.54 s)

900000 of 1000000 tuples (90%) done (elapsed 2.35 s, remaining 0.26 s)

1000000 of 1000000 tuples (100%) done (elapsed 2.53 s, remaining 0.00 s)

vacuum...

set primary keys...

done.

1. View archive log location

Oct 1 23:20 0000000100000000000000E0.00000028.backup

Oct 1 23:21 0000000100000000000000E1

Oct 1 23:21 0000000100000000000000E2

Oct 1 23:21 0000000100000000000000E3

Oct 1 23:21 0000000100000000000000E4

Oct 1 23:21 0000000100000000000000E5

Oct 1 23:21 0000000100000000000000E6

Oct 1 23:21 0000000100000000000000E7

1. Prepare restore location.

cd /u01/backrest

tar -xf base.tar

tar -xf pg\_wal.tar

mv 0000000\* pg\_wal

sed -i 's/#port = 5432/port = 5444/g' postgresql.conf

chmod 700 /u01/backrest

cat recovery.conf

restore\_command='cp /u01/archivelogs/%f %p'

recovery\_target\_lsn='0/E4FF4F18'

1. Start backrest

pg\_ctl start -D /u01/backrest/

1. When invoked in alert log to resume run below command.

select pg\_wal\_replay\_resume();

Alert log

LOG: database system was interrupted; last known up at 2020-10-01 23:20:55 IST

LOG: starting point-in-time recovery to WAL location (LSN) "0/E4FF4F18"

LOG: restored log file "0000000100000000000000E0" from archive

LOG: redo starts at 0/E0000028

LOG: consistent recovery state reached at 0/E00000F8

LOG: database system is ready to accept read only connections

LOG: restored log file "0000000100000000000000E1" from archive

LOG: restored log file "0000000100000000000000E2" from archive

LOG: restored log file "0000000100000000000000E3" from archive

LOG: restored log file "0000000100000000000000E4" from archive

LOG: recovery stopping after WAL location (LSN) "0/E4FF4F18"

LOG: recovery has paused

HINT: Execute pg\_wal\_replay\_resume() to continue.

LOG: connection received: host=[local]

LOG: connection authorized: user=postgres database=postgres

LOG: redo done at 0/E4FF4F18

LOG: last completed transaction was at log time 2020-10-01 23:21:17.297319+05:30

LOG: restored log file "00000002.history" from archive cp: cannot stat `/u01/archivelogs/00000003.history': No such file or directory

LOG: selected new timeline ID: 3 cp: cannot stat `/u01/archivelogs/00000001.history': No such file or directory

LOG: archive recovery complete

LOG: checkpoint starting: end-of-recovery immediate wait

LOG: checkpoint complete: wrote 10246 buffers (62.5%); 0 WAL file(s) added, 0 removed, 0 recycled; write=0.694 s, sync=0.000 s, total=0.701 s; sync files=22, longest=0.000 s, average=0.000 s; distance=81882 kB, estimate=81882 kB

LOG: database system is ready to accept connections

LOG: disconnection: session time: 0:00:25.661 user=postgres database=postgres host=[local]

**Activity 4, Don’t start standby but let it wait for next archive log to come in.**

Same as above, but add below modify recovery.conf

**standby\_mode='on'**

**restore\_command='scp 192.168.1.128:/u01/archivelogs/%f %p'**

**trigger\_file='/tmp/postgresql.trigger.5432'**

**CHAPTER – 10**

Upgrade your database:

**To check if your system is compatible?**

/usr/pgsql-11/bin/pg\_upgrade -d /u01/pgsql/10 -D /u01/pgsql/11 -b /usr/local/pgsql/bin/ -B /usr/pgsql-11/bin/ -c

**To upgrade using copy method?**

/usr/pgsql-11/bin/pg\_upgrade -d /u01/pgsql/10 -D /u01/pgsql/11 -b /usr/local/pgsql/bin/ -B /usr/pgsql-11/bin/

**To upgrade using link method?**

/usr/pgsql-11/bin/pg\_upgrade -d /u01/pgsql/10 -D /u01/pgsql/11 -b /usr/local/pgsql/bin/ -B /usr/pgsql-11/bin/ -k

**CHAPTER 11**

Download and install EDB and PGAdmin. (Next, Next..Finish)

Download and install pgbadger. (make && make install)

Generate pgbadger report?

Pgbadger <logfile> -o output.html

Day to Day activity commands

https://github.com/postgreshelp/help\_commands/blob/master/postgres\_commands.sql

What’s next ?

<https://github.com/postgreshelp/awesome-postgres>