PERCONA

TRAINING

PostgreSQL Operations and Troubleshooting

http://www.percona.com/training/

Table Of Contents

Postgresql Operations And Troubleshooting

Before You Start: Lab Setup

Lab Setup

- Each student will be provided with an AWS instance, configured with a three node cluster using LXC.
- Each container node uses CentOS 7 with both postgres and pgbouncer preinstalled in each container.
- You will have the ability to login and administrate each node as you progress through the course.

Network Layout

```
HOST
    A single Amazon cloud instance, login account is "student"


CONTAINERS
    pg1: 192.168.2.11
    pg2: 192.168.2.12
    pg3: 192.168.2.13
```

- Each node, pg1, pg2, pg3 etc, is preconfigured with two accounts:

ACCOUNT	PASSWORD
root	*root*
postgres	*postgres*

What You Need

- SSH client (terminal session)
- Private key "student.pem" to be accessed and used by your SSH client
- IP address cloud instance, *to be provided by the instructor*

```
# Example LOGIN Session
ssh -i student.pem student@54.189.95.186
```

About PG Containers

- Once logged into the cloud instances, test if containers are started:
    - PG1 (192.168.2.11)
    - PG2 (192.168.2.12)
    - PG3 (192.168.2.13)

```
for u in 1 2 3
do
    ping -c 1 pg$u
done
```

About PG Containers, Cont'd

- Housekeeping; update packages on each container:

```
# login and update each container ...
#   password: "root"
#
# EX: pg1
# CENTOS 8
ssh root@pg1

    dnf update -y
    dnf repolist
    updatedb
    systemctl start postgresql-15
    systemctl status postgresql-15
    netstat -tlnp
    exit
```

About PG Containers, Cont'd

Postgres user login to each container:

```
# password: "postgres"
#
# EX: pg1
ssh postgres@pg1

#
psql
```

Node Administration

- As root:

```
systemctl [stop|status|start] [postgresql-15|pgbouncer]

systemctl [enable|disable] [postgresql-15|pgbouncer]

netstat -tlnp
```

Miscellaneous

- Both the postgres and pgbouncer are initialized at their default configurations
- one can su from postgres to root (PW is "root")

```
# as postgres
su - root
```

- GNU Midnight Commander, "mc", Visual shell for Unix-like systems is available
- screen is installed on the HOST node but not installed in the containers
- environment variable PGDATA is already declared for postgres

```
# as postgres
echo $PGDATA
```

Miscellaneous (cont'd)

- update the PAGER environment variable to suit your preference

```
# as postgres
echo "export PAGER=less -S" > $HOME/.pgsql_profile
```

- **ATTENTION**: Disk space is shared by all 3 pg nodes

```
-bash-4.2$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/xvda1       20G  7.6G   12G  40% /
none            492K     0  492K   0% /dev
udev            473M     0  473M   0% /dev/tty
tmpfs           100K     0  100K   0% /dev/lxd
tmpfs           100K     0  100K   0% /dev/.lxd-mounts
tmpfs           488M  1.1M  487M   1% /dev/shm
tmpfs           488M  6.7M  481M   2% /run
tmpfs           488M     0  488M   0% /sys/fs/cgroup
tmpfs            98M     0   98M   0% /run/user/0
tmpfs            98M     0   98M   0% /run/user/26
```

Postgresql Operations And Troubleshooting

Overview Of Postgresql

About PostgreSQL

History of PostgreSQL

Ingres

- Year 1973 - INGRES (INteractive GRaphics Retrieval System) - University of California at Berkeley.
- Year 1979 - Oracle Database first version
- Early 1980’s - Ingres lost to Oracle that used SQL as a preferred query language.
- Year 1985 - UC Berkeley INGRES research project officially ended.

Postgres

- Year 1986 - Postgres introduced as Post-Ingres evolution. POSTQUEL query language until 1994
- Year 1995 - Postgres95 with support for SQL.

PostgreSQL

- Year 1996 - Renamed to PostgreSQL.
- Year 1997 - PostgreSQL first version - PostgreSQL 6.0 released.

PostgreSQL: Features

Portable

- Written in C
- Flexible across all the UNIX platforms, Windows, MacOS and others.
- World’s most advanced open source database. Community driven.
- ANSI/ISO Compliant SQL support.

Reliable

- ACID Compliant
- Supports Transactions
- Uses Write Ahead Logging

Scalable

- MVCC
- Table Partitioning
- Tablespaces
- FDWs (Foreign Data Wrappers)
- Sharding

PostgreSQL: Advanced Features

Security

- Host-Based Access Control
- Object-Level and Row-Level Security
- Logging and Auditing
- Encryption using SSL

High Availability

- Synchronous/Asynchronous Replication and Delayed Standby
- Cascading Replication
- Online Consistent Physical Backups and Logical Backups
- PITR

Additional Features

- Triggers and Functions/Stored Procedures
- Custom Stored Procedural Languages like PL/pgSQL, PL/perl, PL/TCL, PL/php, PL/python, PL/java.
- PostgreSQL Major Version Upgrade using pg_upgrade
- Unlogged Tables
- Materialized Views
- Hot Standby - Slaves accept Reads

PostgreSQL: ACID Compliance

Atomicity

- Either **everything** should succeed in a transaction or **nothing** when something fails.
- BEGIN …SQL1, SQL2, …SQLn…..COMMIT/ROLLBACK/END.

Consistency

- Give me a consistent picture of the data based on Isolation Levels. Example: **READ_COMMITTED**
- Query 1 : select count(*) from employee;

```
9:00 am : Records in employee table : 10000
9:10 am : Query 1 Started by User 1
9:11 am : 2 employee records deleted by User 2.
9:12 am : Query 1 that was started by User 1 Completed.
```

- Result of Query 1 at 9:12 am would still be 10000. A Consistent image as how it was at 9:00 am.

Isolation

- Prevent Concurrent data access through Locking.

Durability

- Once the Data is committed, it must be safe.
- Through WAL's, fsync, synchronous_commit and replication.

PostgreSQL: Terminology

- PostgreSQL was designed in academia
    - Objects are defined in academic terms.
    - Terminology based on relational calculus/algebra

Industry Term	PostgreSQL Term
Table/Index	Relation
Row	Tuple
Column	Attribute
Data Block	Page (when data block is on disk)
Page	Buffer (when data block is in memory)

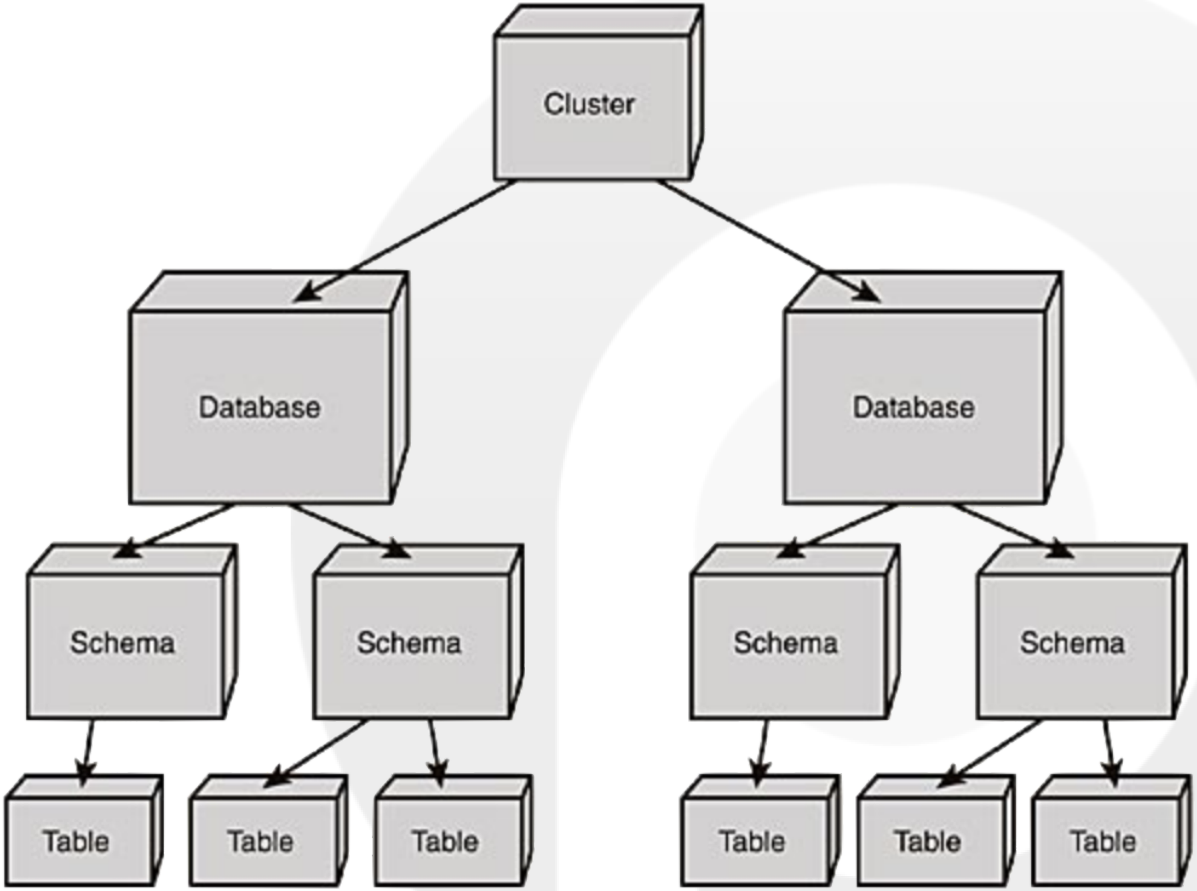
PostgreSQL: References

- Portal: https://www.postgresql.org/
- References And Resources
  - https://www.postgresql.org/docs/current/index.html
  - https://www.postgresql.org/docs/current/bookindex.html
  - https://www.postgresql.org/docs/online-resources/
  - https://www.percona.com/blog/

Postgresql Operations And Troubleshooting

Postgresql Internals

PostgreSQL Cluster Diagram



PostgreSQL Database & Schema

- PostgreSQL Database contains one or more schemas. Default - public schema.

- A Schema groups objects together. An example : A folder/directory that contains tables, index and other objects as files.

- You can always have more than 1 Database with one or more schemas in it.
- A Schema in PostgreSQL groups objects of a certain application logic together. Helps create multiple objects with the same name in one Database.

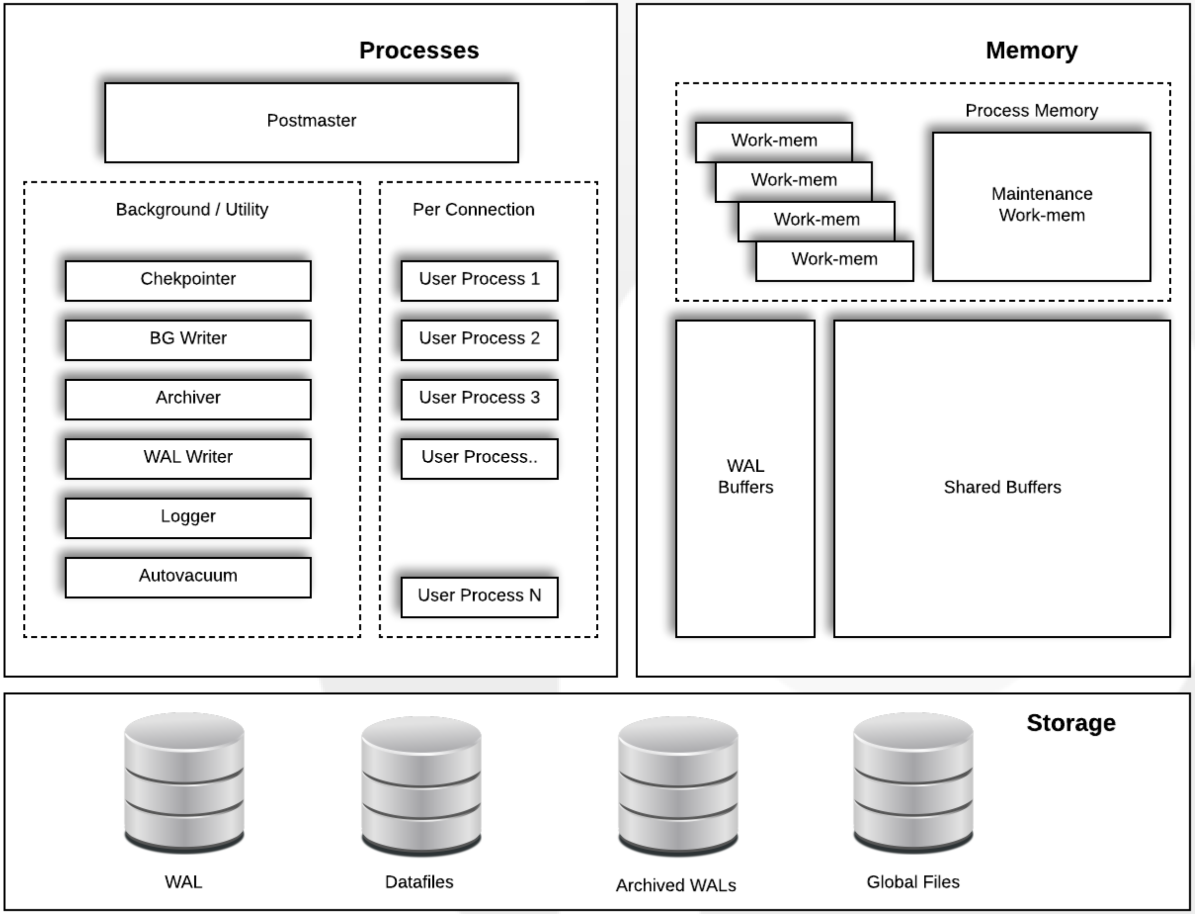
- For example : In a database named **percona**, a table with tablename **employee** can exist in both **scott** and **tiger** schemas.

```
Database   : percona
Schema(s) : scott & tiger
Tables     : scott.employee & tiger.employee
```

- A Fully Qualified Table Name : **schema_name.table_name** must be used to query a particular table in a schema.

```
select * from scott.employee where salary > 10000;
```

Postgresql Architecture



PostgreSQL Server

Multi-Process Architecture.

- Postmaster (Parent PostgreSQL Process)
- Backend Utility Processes
- Per-Connection backend processes
- Every Connection is a Process.
- Whereas each connection is a thread in MySQL - Multi-threaded.

Background Utility Processes

- Start your PostgreSQL server:

  systemctl start postgresql-15

```
$ ps aux|grep postgres:
postgres   771  0.0  0.1 352712  1388 ?        Ss   19:21   0:00 postgres: logger
postgres   772  0.0  0.3 501012  3892 ?        Ss   19:21   0:00 postgres: checkpointer
postgres   773  0.0  0.1 501024  1724 ?        Ss   19:21   0:00 postgres: background writer
postgres   775  0.0  0.5 500884  5740 ?        Ss   19:21   0:00 postgres: walwriter
postgres   776  0.0  0.2 502476  2800 ?        Ss   19:21   0:00 postgres: autovacuum launcher
postgres   777  0.0  0.1 502452  1896 ?        Ss   19:21   0:00 postgres: logical replication lau
```

systemctl status postgresql-15

```
[root@pg1 ~]# systemctl status postgresql-15
● postgresql-15.service - PostgreSQL 15 database server
   Loaded: loaded (/usr/lib/systemd/system/postgresql-15.service; enabled; vendor preset: disabled
  Drop-In: /run/systemd/system/postgresql-15.service.d
           └─zzz-lxc-service.conf
   Active: active (running) since Fri 2023-04-14 19:21:55 UTC; 1h 8min ago
     Docs: https://www.postgresql.org/docs/15/static/
  Process: 759 ExecStartPre=/usr/pgsql-15/bin/postgresql-15-check-db-dir ${PGDATA} (code=exited, s
 Main PID: 766 (postmaster)
    Tasks: 7 (limit: 6052)
   Memory: 29.6M
   CGroup: /system.slice/postgresql-15.service
           ├─766 /usr/pgsql-15/bin/postmaster -D /var/lib/pgsql/15/data/
           ├─771 postgres: logger
           ├─772 postgres: checkpointer
           ├─773 postgres: background writer
           ├─775 postgres: walwriter
           ├─776 postgres: autovacuum launcher
           └─777 postgres: logical replication launcher

Apr 14 19:21:51 pg1 systemd[1]: Starting PostgreSQL 15 database server...
Apr 14 19:21:55 pg1 postmaster[766]: 2023-04-14 19:21:55.147 UTC [766] LOG:  redirecting log outpu
Apr 14 19:21:55 pg1 postmaster[766]: 2023-04-14 19:21:55.147 UTC [766] HINT:  Future log output wi
Apr 14 19:21:55 pg1 systemd[1]: Started PostgreSQL 15 database server.
```

Process Components

Postmaster :

- Master database control process.
- Responsible for startup & shutdown.
- Spawns necessary backend processes.

Postgres backend :

- Dedicated, per-connection server process.
- Responsible for fetching data from disk and communicating with the client.

Utility Processes

- **BGWriter :**
  - Background Writer
  - Writes/Flushes dirty data blocks to disk.
- **WAL Writer :**
  - Writes WAL Buffers to Disk.
  - WAL Buffers are written to WALs(Write-Ahead Logs) on the disk.
- **Autovacuum Launcher:**
  - Starts Autovacuum worker processes to start a vacuum and analyze job in the backend.
- **Checkpointer :**
  - Perform a CHECKPOINT that ensures that all the changes are flushed to disk.
  - Depends on configuration parameters.

Utility Processes

- **Archiver :**
  - Archives Write-Ahead-Logs.
  - Used for High Availability, Backups and PITR (point-in-time-recovery).
- **Logger :**
  - Logs messages, events, error to syslog or log files.
  - Errors, slow running queries, warnings,..etc. are written to log files by this process.
- **Stats Collector :**
  - Collects statistics of relations (tables).
  - Needed by autovacuum launcher process.

Utility Processes

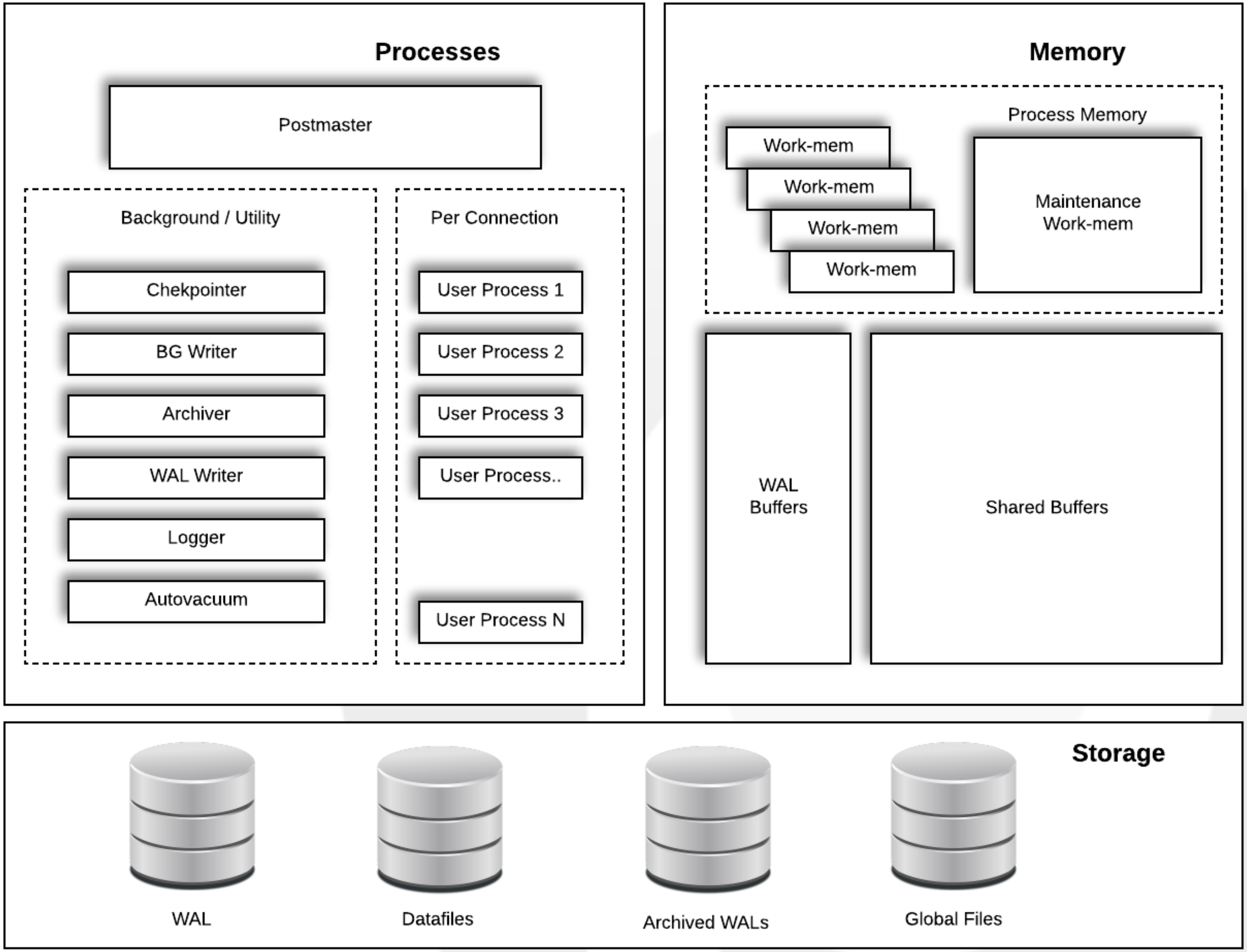
- **WAL Sender :**
    - Sends WALs to Replica(s).
    - One WAL Sender for each Slave connected for Replication.
- **WAL Receiver :**
    - Started on a Slave(aka Standby or Replica) in Replication.
    - Streams WALs from Master.
- **bgworker :**
    - PostgreSQL is extensible to run user-supplied code in separate processes that are monitored by Postgres.
    - Such processes can access PostgreSQL's shared memory area.
    - Connect as a client using libpq.
- **bgworker: logical replication launcher**
    - Logical Replication between a Publisher and a Subscriber.

Memory Components

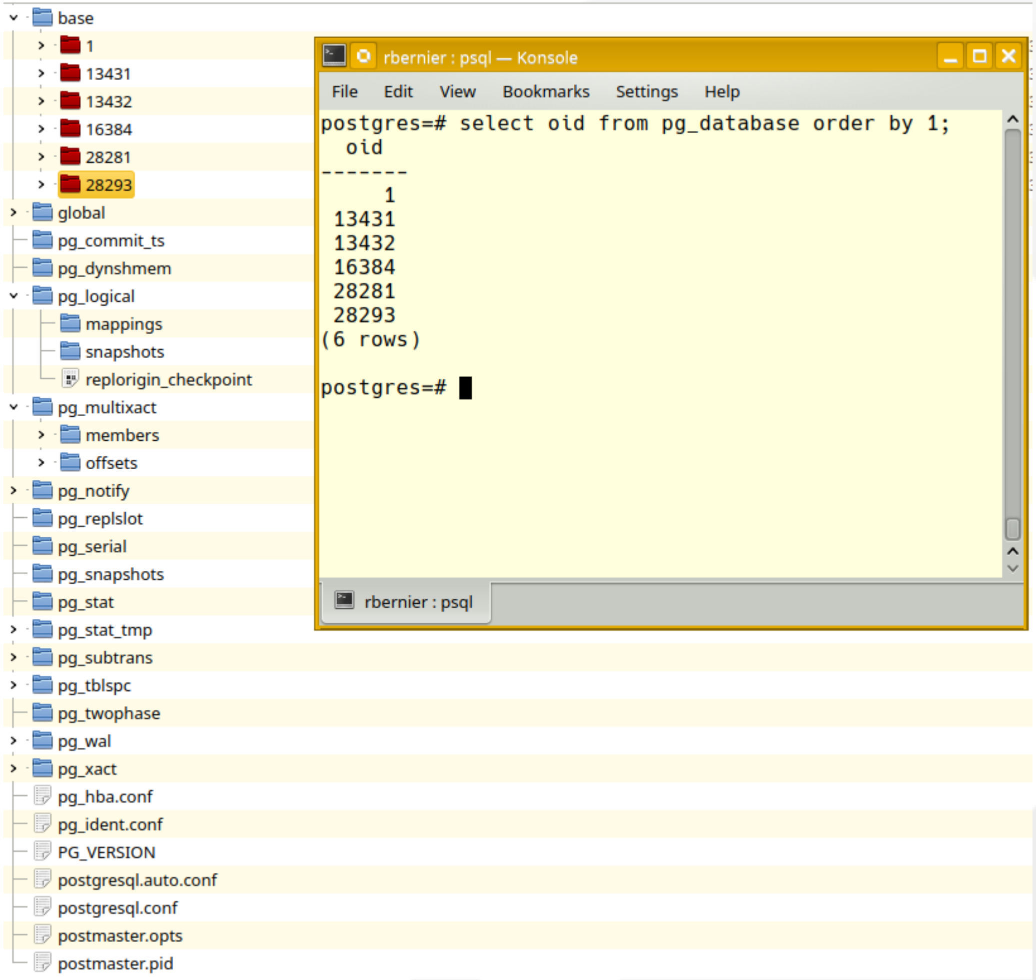
- **Shared Buffers**
  - PostgreSQL Database Memory Area.
  - Shared by all the Databases in the Cluster.
  - Pages are fetched from Disk to Shared Buffers during Reads/Writes.
  - Modified Buffers are also called as Dirty Buffers.
  - Parameter : **shared_buffers** sets the amount of RAM allocated to Shared Buffers.
  - Uses LRU Algorithm to flush less frequently used buffers.
  - Dirty Buffers written to disk after a CHECKPOINT.
- **WAL Buffers :**
  - Stores Write Ahead Log Records.
  - Contains the change vector for a buffer being modified.
  - WAL Buffers written to WAL Segments (On sisk).
- **work_mem :**
  - Memory used by each Query for internal sort operations such as ORDER BY and DISTINCT.
  - Postgres writes to disk (temp files) if this memory is not sufficient.

Memory Components

- **maintenance_work_mem**
  - Amount of RAM used by VACUUM, CREATE INDEX, REINDEX like maintenance operations.
  - Setting this to a bigger value can help in faster database restore.


Processes
Postmaster
Background / Utility
Chekpointer
BG Writer
Archiver
WAL Writer
Logger
Autovacuum
Per Connection
User Process 1
User Process 2
User Process 3
User Process..
User Process N
Memory
Process Memory
Work-mem
Work-mem
Work-mem
Work-mem
Maintenance Work-mem
WAL Buffers
Shared Buffers
Storage
WAL
Datafiles
Archived WALs
Global Files

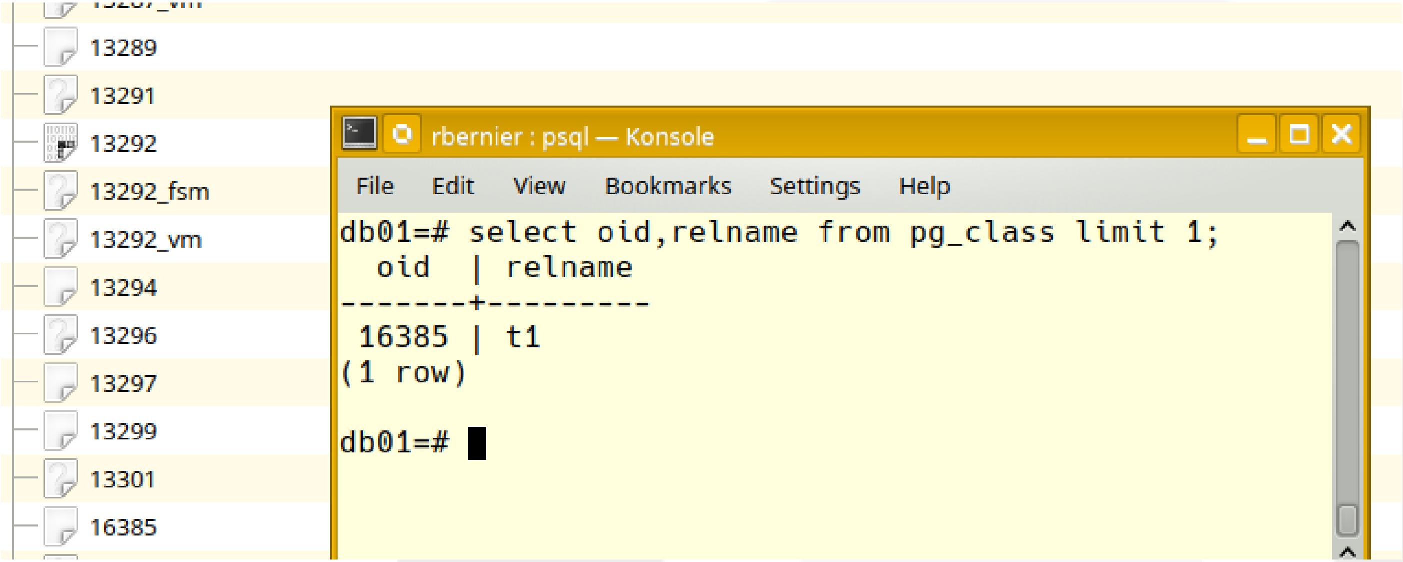
DataCluster: Directory Tree



```
base
    1
    13431
    13432
    16384
    28281
    28293
global
pg_commit_ts
pg_dynshmem
pg_logical
    mappings
    snapshots
    replorigin_checkpoint
pg_multixact
    members
    offsets
pg_notify
pg_replslot
pg_serial
pg_snapshots
pg_stat
pg_stat_tmp
pg_subtrans
pg_tblspc
pg_twophase
pg_wal
pg_xact
pg_hba.conf
pg_ident.conf
PG_VERSION
postgresql.auto.conf
postgresql.conf
postmaster.opts
postmaster.pid
```

```
postgres=# select oid from pg_database order by 1;
  oid
-------
      1
  13431
  13432
  16384
  28281
  28293
(6 rows)

postgres=#
```

DataCluster: Files<->Tables, Indexes



```
13287_vm
13289
13291
13292
13292_fsm
13292_vm
13294
13296
13297
13299
13301
16385
```

```
db01=# select oid,relname from pg_class limit 1;
  oid  | relname
-------+---------
 16385 | t1
(1 row)

db01=# 
```

Postgresql Is Not Direct IO

- When it needs a Page(Data Block), it searches it’s own memory aka Shared Buffers.
- If not found in shared buffers, it will request the OS for the same block.
- The OS fetches the block from the Disk and gives it to Postgres, if the block is not found in OS Cache.
- More important to Caching when Database and Active Data set cannot fit in memory.

Disk Components

- **Data Directory**
  - In MySQL, Data Directory is created when you initialize your MySQL Instance.
  - Initialized using **initdb** in PostgreSQL. Similar to mysqld --initialize.
  - Contains Write-Ahead-Logs, Log Files, Databases, Objects and other configuration files.
  - You can move WAL's and Logs to different directories using symlinks and parameters.
  - Environment Variable : $PGDATA
- **Configuration files in the data directory**
  - postgresql.conf
  - pg_ident.conf
  - pg_hba.conf
  - postgresql.auto.conf
  - recovery.conf # *deprecated as of version 12*

Configuration Files In The Data Directory

- **PG_VERSION**
  - Version String of the Database Cluster.
- **pg_hba.conf**
  - Host-Based access control file (built-in firewall).
- **pg_ident.conf**
  - ident-based access file for OS User to DB User Mapping.
- **postgresql.conf**
  - Primary Configuration File for the Database.
- **postmaster.opts**
  - Contains the options used to start the PostgreSQL Instance.
- **postmaster.pid**
  - The parent process id or the Postmaster process id.

.conf vs auto.conf

- **postgresql.conf**
  - Configuration file for PostgreSQL similar to my.cnf for MySQL.
  - Contains parameters required to run a PostgreSQL Instance.
  - Parameters are set to their default values unless modified.
  - Located in the data directory or /etc. Changes with distribution and is modifiable.
- **postgresql.auto.conf**
  - PostgreSQL gives Oracle like compatibility to modify parameters using **ALTER SYSTEM**.
  - Any parameter modified using ALTER SYSTEM is written to this file for persistence (upon restart of postgres).
  - This is last configuration file read by PostgreSQL, when started. Empty by default.
  - Always located in the data directory.

View And Modify Parameters

- Use **show** to view a value of a parameter

```
$ psql -c "show work_mem"
```

- To see all the settings, use **show all**

```
$ psql -c "show all"
```

- Use `ALTER SYSTEM` to modify a parameter

```
$ psql -c "ALTER SYSTEM SET archive_mode TO ON"
```

- Use the reload function to put changes into effect for parameters not needing `RESTART`

```
$ psql -c "select pg_reload_conf()"
# OR
$ pg_ctl -D $PGDATA reload
```

Base Directory & Datafiles On Disk

- **Base Directory**
  - Contains Sub-Directories for every **database** you create.
  - Every Database Sub-Directory contains files for every Relation/Object created in the database.
- **Datafiles**
  - Base Directory contains Relations.
  - Datafiles are the files for Table Relations in the base directory.
  - Relations stored on Disk as 1GB segments.
  - Each 1GB Datafile is made up of several 8KB (modifiable) Pages that are allocated as needed.
  - Segments are automatically added unlike Oracle.
- **Indexes**
  - Similar properties as Relations/Tables.
  - Base Directory contains Indexes associated with the Tables.
- **Tablespaces**
  - A tablespace allows superusers to define an alternative location on the file system where the data files containing database objects (such as tables and indexes) can reside.
- Reference: https://www.postgresql.org/docs/current/storage-file-layout.html

Base Directory

- **1. Create a database with name as : percona**

```
$ psql -c "CREATE DATABASE percona"
```

- **2. Get the datid for the database and see if it exists in the base directory**

```
$ psql -c "select datid, datname from pg_stat_database where datname = 'percona'"
```

In the following output, you should see that a directory with name as **datid of database** is created under the base directory.

```
$ psql -c "CREATE DATABASE percona"
CREATE DATABASE
$ psql -c "select datid, datname from pg_stat_database where datname = 'percona'"
 datid | datname
-------+---------
 16385 | percona
(1 row)
$ ls -ld $PGDATA/base/16385
drwx------. 2 postgres postgres 8192 Dec 13 13:38 /var/lib/pgsql/15/data/base/16385
```

Base Directory (Schema And Relations)

- **1. Create a schema named : scott**

```
$ psql -d percona -c "CREATE SCHEMA scott"
```

- **2. Create a table named : employee in schema : scott**

```
$ psql -d percona -c "CREATE TABLE scott.employee(id int PRIMARY KEY, name varchar(20))"
```

- **3. Locate the file created for table : scott.employee in the base directory**

```
$ psql -d percona -c "select pg_relation_filepath('scott.employee')"
```

Base Directory (Schema And Relations)

- In the following output, we see that the table : scott.employee (oid = 16387) is created inside the database : percona (datid = 16385)

```
$ psql -d percona -c "CREATE SCHEMA scott"
CREATE SCHEMA
$ psql -d percona -c "CREATE TABLE scott.employee(id int PRIMARY KEY, name varchar(20))"
CREATE TABLE
$ psql -d percona -c "select pg_relation_filepath('scott.employee')"
 pg_relation_filepath
----------------------
 base/16385/16387
(1 row)
$ ls -larth $PGDATA/base/16385/16387
-rw-------. 1 postgres postgres 0 Dec 13 13:54 /var/lib/pgsql/15/data/base/16385/16387
```

- Also observe that the size of file (table : scott.employee) is 0 bytes.

Base Directory (Block Size)

- Check the size of the table in the OS and value of parameter : **block_size**

```
$ psql -c "show block_size"
```

- INSERT a record in the table and see the size of the file

```
$ psql -d percona -c "INSERT INTO scott.employee VALUES (1, 'frankfurt')"
$ ls -larth $PGDATA/base/16385/16387
```

- INSERT more records and check the size difference

```
$ psql -d percona -c "INSERT INTO scott.employee VALUES (generate_series(2,1000), 'junk')"
$ ls -larth $PGDATA/base/16385/16387
```

Base Directory (Block Size)

- In the following output, we see that -
  - the table size is increasing in the multiples of block_size (8 KB here)
  - the table size displayed through \dt+ is slightly higher because that includes the primary key index

```
$ psql -c "show block_size"
 block_size
------------
 8192

$ psql -d percona -c "INSERT INTO scott.employee VALUES (1, 'frankfurt')"
INSERT 0 1
$ ls -larth $PGDATA/base/16385/16387
-rw-------. 1 postgres postgres 8.0K Dec 13 14:10 /var/lib/pgsql/15/data/base/16385/16387

$ psql -d percona -c "INSERT INTO scott.employee VALUES (generate_series(2,1000), 'junk')"
INSERT 0 999
$ ls -larth $PGDATA/base/16385/16387
-rw-------. 1 postgres postgres 48K Dec 13 14:11 /var/lib/pgsql/15/data/base/16385/16387

-- Table size including Indexes

$ psql -d percona -c "\dt+ scott.employee"
                     List of relations
 Schema |   Name   | Type  |  Owner   | Size  | Description
--------+----------+-------+----------+-------+-------------
 scott  | employee | table | postgres | 72 kB |
(1 row)
```

Write Ahead Logs (WAL)

- **WALs**
  - When Client commits a transaction, it is written to WAL Segments (on Disk) before a success message is sent to Client.
  - PostgreSQL WALS vs Oracle REDO Logs.
  - Written by WAL Writer background process.
  - Ensures durability when **fsync** and **synchronous_commit** set to **ON** and **commit_delay** set to **0**.
  - Used during Crash Recovery.
  - Size of each WAL is 16MB. Modifiable during Initialization.
  - WALs are generated in **pg_wal** directory
  - WAL directory exits in data directory by default. Can be modified using Symlinks.
  - WALs are deleted depending on the parameters : **wal_keep_size**, **max_slot_wal_keep_size**, **checkpoint_timeout**, *stray replication slots will also result in WALs piling up*

Archived Logs And Why ?

- **Archived WALs**
  - WALs in pg_wal are gone after a certain threshold. Archiving ensures recoverability and helps a Slave catch-up during replication lag.
  - Archiving in PostgreSQL can be enabled through parameters : **archive_mode** and **archive_command**.
  - Ships WALs to safe locations like a Backup Server or Cloud Storage like S3 or Object Store.
  - WALs are archived by archiver background process.
  - **archive_command** can be set with the appropriate shell command to archive WALs.

Steps To Enable Archiving

- Log in to your PostgreSQL Instance and modify parameters : **listen_addresses**, **archive_mode**, **archive_command**

```
-- Create the archive directory.
$ mkdir ~/archive

-- Modify the PostgreSQL parameters.
$ psql -c "ALTER SYSTEM SET archive_mode TO 'ON'"
$ psql -c "ALTER SYSTEM SET archive_command TO 'cp %p /var/lib/pgsql/archive/%f'"

-- Restart PostgreSQL to get these parameters into effect.
$ pg_ctl -D $PGDATA restart -mf
```

Switch A Wal

- Switch a WAL using the following command

```
$ psql -c "select pg_switch_wal()"
```

- Check if the previous WAL has been safely archived.

```
-- List all the WALs
$ ls -l $PGDATA/pg_wal
-- List all the WALs that have been archived.
$ ls -l /var/lib/pgsql/archive
```

- We should now see the previously generated WAL segment archived.

- **archive_command** can take any shell command/script that can ship a WAL to any destination.

What If Archiving Failed ?

- If archiving has been enabled and the archive_command failed,
  - the WAL segment for which the archiving failed will not be removed from pg_wal or pg_xlog
  - an empty **wal_file_name.ready** file is generated in the **archive_status** directory
  - the background process **archiver** attempts to archive the failed WAL segment until it succeeds.
  - there is a chance that the pg_wal directory can get filled and doesn't allow any more connections to database.

```
-- When archiving is succeeded
$ ls -l $PGDATA/pg_wal/archive_status
-rw-------. 1 postgres postgres  0 Dec 13 19:31 000000010000000000000002.done
-- When archiving is failed
$ ls -l $PGDATA/pg_wal/archive_status
-rw-------. 1 postgres postgres 0 Dec 13 19:32 000000010000000000000003.ready
```

PostgreSQL Operations And Troubleshooting

Administration Basics

Installation

Community: https://www.postgresql.org/download/linux/

- CENTOS 8:

```
dnf search postgres*contrib
dnf install -y postgresql15-contrib
```

- Ubuntu:

```
apt search postgres | grep -E '^postgres' | less
apt install -y postgresql-15
```

https://www.postgresql.org/download/linux/

Installation Cont'd

Remove pre-existing datacluster

- CENTOS:

```
# check status of services
ssh root@pg[123]
systemctl status postgresql-15

# stop services
systemctl stop postgresql-15
netstat -tlnp

# remove pre-existing datacluster
su - postgres
rm -rf $PGDATA
exit
```

Installation Cont'd

Initialize datacluster

- CENTOS:

```
/usr/pgsql-15/bin/postgresql-15-setup initdb
```

- Ubuntu: (automatic, no intervention required)

Start/Stop/Reload/Status

- CENTOS:

```
systemctl start|stop|status postgresql-15
```

- Ubuntu:

```
systemctl start|stop|status postgresql@15-main
```

Configuration Edits

- CENTOS:

```
systemctl edit postgresql-15 [--full]
vi /var/lib/pgsql/15/data/[pg_hba.conf|postgresql.conf]
```

- Ubuntu:

```
systemctl edit postgresql@15-main [--full]
pg_conftool --help
pg_lsclusters
pg_conftool 12 main [pg_hba.conf|postgresql.conf] edit
```

Configuration: pg_hba.conf

Example of host based authentication rules:

```
# PostgreSQL Client Authentication Configuration File
# ===================================================
#
# TYPE  DATABASE        USER            ADDRESS                 METHOD
local   all             postgres                                peer
local   replication     all                                     peer

# host connections:
host    all             all             0.0.0.0/0               md5
host    all             all             ::0/0                   md5

# host connections, replication:
host    replication     all             0.0.0.0/0               md5
host    replication     all             ::0/0                   md5
```

Configuration: postgresql.conf

Example postgres runtime parameters:

```
 listen_addresses = '*'
#listen_addresses = 'localhost'         # what IP address(es) to listen on;

 logging_collector = on
 log_filename = 'postgresql-%a.log'
 log_truncate_on_rotation = on
 log_rotation_size = 0
```

Configuration: Initial Tuning

Minimal Settings For A Newly Initialized Datacluster

```
shared_buffers = 128MB                    # assign RAM between 1/4-1/3
work_mem = 4MB                            # latest industry settings put it typically at 10MB

maintenance_work_mem = 64MB               # depends upon loading of number autovacuum workers etc
                                          #     and DDL operations i.e. CREATE INDEX


fsync = on                                # except for setting to OFF for bulk uploads leave it ON
effective_cache_size = 4GB                # assign max 75% available RAM, can be tricky

autovacuum_max_workers = 3                # depends on amount of load it causes
                                          #     and number of CPUs available
```

Based Upon Hardware Resources i.e. CPU, RAM

About MVCC, VACUUM And ANALYZE

- MVCC : Multi-Version Concurrency Control.
- Maintains Data Consistency Internally.
- Prevents transactions from viewing inconsistent data.
- Readers do not block Writers and Writers do not block Readers.
- MVCC controls which tuples can be visible to transactions via Versions.
- Hidden Column **xmin** that has the transaction ID for every row.
- **UNDO** is not maintained in a Separate UNDO Segment. UNDO is stored as **Older Versions** within the same Table.
- Every Tuple has hidden columns => **xmin** and **xmax** that records the **minimum** and **maximum** transaction ids that are permitted to see the row.
- xmin can be interpreted as the lowest transaction ID that can see this column.
- Just like SELECT statements executing WHERE xmin <= txid_current() AND (xmax = 0 OR txid_current() < xmax)
- Dead rows are the rows that no active or future transaction would see.
- Rows that got deleted would get their xmax with the txid that deleted them.

VACUUM: Dead Tuples

- Due to continuous transactions in the databases and the number of dead rows, there exists a lot of space that can be re-used by future transactions.
- Tuples that are deleted or updated generate dead tuples that are not physically deleted.
  - See view => **pg_stat_user_tables** to check the number of dead tuples
- VACUUM in PostgreSQL would clear off the dead tuples and mark it to free space map so that the future transactions can re-use the space.
  - **VACUUM percona.employee;**
- VACUUM FULL in PostgreSQL would rebuild the entire Table with explicit Locks, releasing the space to File System. Similar to ALTER TABLE in MySQL.
  - **VACUUM FULL percona.employee;**
- Autovacuum in PostgreSQL automatically runs VACUUM on tables depending on the following parameters.
  - **autovacuum_vacuum_scale_factor** and **autovacuum_vacuum_threshold**.

VACUUM: ANALYZE

- ANALYZE colects statistics about the contents of tables in the database, and stores the results in the system catalogs.
- The autovacuum daemon, takes care of automatic analyzing of tables when they are first loaded with data.
- Accurate statistics will help the planner to choose the most appropriate query plan, and thereby improve the speed of query processing.
  - **ANALYZE percona.employee;**
- Autovacuum Launcher Process runs an Analyze on a Table depending on the following parameters
  - **autovacuum_analyze_scale_factor** and **autovacuum_analyze_threshold**.

Table Attributes: Storage Parameters

- fillfactor (integer)
- toast_tuple_target (integer)
- parallel_workers (integer)
- autovacuum_enabled, toast.autovacuum_enabled (boolean)
- vacuum_index_cleanup, toast.vacuum_index_cleanup (boolean)
- vacuum_truncate, toast.vacuum_truncate (boolean)
- autovacuum_vacuum_threshold, toast.autovacuum_vacuum_threshold (integer)
- autovacuum_vacuum_scale_factor, toast.autovacuum_vacuum_scale_factor (float4)
- autovacuum_analyze_threshold (integer)
- autovacuum_analyze_scale_factor (float4)
- autovacuum_vacuum_cost_delay, toast.autovacuum_vacuum_cost_delay (floating point)
- autovacuum_vacuum_cost_limit, toast.autovacuum_vacuum_cost_limit (integer)
- autovacuum_freeze_min_age, toast.autovacuum_freeze_min_age (integer)
- autovacuum_freeze_max_age, toast.autovacuum_freeze_max_age (integer)
- autovacuum_freeze_table_age, toast.autovacuum_freeze_table_age (integer)
- autovacuum_multixact_freeze_min_age, toast.autovacuum_multixact_freeze_min_age (integer)
- autovacuum_multixact_freeze_max_age, toast.autovacuum_multixact_freeze_max_age (integer)
- autovacuum_multixact_freeze_table_age, toast.autovacuum_multixact_freeze_table_age (integer)
- log_autovacuum_min_duration, toast.log_autovacuum_min_duration (integer)
- user_catalog_table (boolean)

Reference:

- CREATE TABLE: https://www.postgresql.org/docs/current/sql-createtable.html
- ALTER TABLE: https://www.postgresql.org/docs/current/sql-altertable.html
- BLOG: https://www.percona.com/blog/

Administration: ROLES

About

PostgreSQL manages database access permissions using the concept of roles. A role can be thought of as either a database user, or a group of database users, depending on how the role is set up. Roles can own database objects (for example, tables and functions) and can assign privileges on those objects to other roles to control who has access to which objects. Furthermore, it is possible to grant membership in a role to another role, thus allowing the member role to use privileges assigned to another role.

The concept of roles subsumes the concepts of “users” and “groups”. In PostgreSQL versions before 8.1, users and groups were distinct kinds of entities, but now there are only roles. Any role can act as a user, a group, or both.

ROLES: Commands

- SQL
  - CREATE ROLE
  - DROP ROLE
  - ALTER ROLE
- Command Line Interface
  - createuser
  - dropuser

CREATE ROLE

Command: CREATE ROLE Description: define a new database role Syntax:

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD _password_ | PASSWORD NULL
    | VALID UNTIL _timestamp_
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

https://www.postgresql.org/docs/15/sql-createrole.html

DROP ROLE

Command: DROP ROLE Description: remove a database role Syntax:

```
DROP ROLE [ IF EXISTS ] name [, ...]
```

https://www.postgresql.org/docs/15/sql-droprole.html

ALTER ROLE

Command: ALTER ROLE Description: change a database role Syntax:

```
ALTER ROLE role_specification [ WITH ] option [ ... ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'

ALTER ROLE name RENAME TO new_name

ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
    SET configuration_parameter { TO | = } { value | DEFAULT }
ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
    SET configuration_parameter FROM CURRENT
ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
    RESET configuration_parameter
ALTER ROLE { role_specification | ALL } [ IN DATABASE database_name ]
    RESET ALL

where role_specification can be:

    role_name
  | CURRENT_USER
  | SESSION_USER
```

https://www.postgresql.org/docs/15/sql-alterrole.html

CLI: createuser

createuser creates a new PostgreSQL role.

```
Usage:
  createuser [OPTION]... [ROLENAME]

Options:
  -c, --connection-limit=N  connection limit for role (default: no limit)
  -d, --createdb            role can create new databases
  -D, --no-createdb         role cannot create databases (default)
  -e, --echo                show the commands being sent to the server
  -g, --role=ROLE           new role will be a member of this role
  -i, --inherit             role inherits privileges of roles it is a
                            member of (default)
  -I, --no-inherit          role does not inherit privileges
  -l, --login               role can login (default)
  -L, --no-login            role cannot login
  -P, --pwprompt            assign a password to new role
  -r, --createrole          role can create new roles
  -R, --no-createrole       role cannot create roles (default)
  -s, --superuser           role will be superuser
  -S, --no-superuser        role will not be superuser (default)
  -V, --version             output version information, then exit
  --interactive             prompt for missing role name and attributes rather
                            than using defaults
  --replication             role can initiate replication
  --no-replication          role cannot initiate replication
  -?, --help                show this help, then exit

Connection options:
  -h, --host=HOSTNAME       database server host or socket directory
  -p, --port=PORT           database server port
  -U, --username=USERNAME   user name to connect as (not the one to create)
  -w, --no-password         never prompt for password
  -W, --password            force password prompt
```

CLI: dropuser

dropuser removes a PostgreSQL role.

```
Usage:
  dropuser [OPTION]... [ROLENAME]

Options:
  -e, --echo                show the commands being sent to the server
  -i, --interactive         prompt before deleting anything, and prompt for
                            role name if not specified
  -V, --version             output version information, then exit
  --if-exists               don't report error if user doesn't exist
  -?, --help                show this help, then exit

Connection options:
  -h, --host=HOSTNAME       database server host or socket directory
  -p, --port=PORT           database server port
  -U, --username=USERNAME   user name to connect as (not the one to drop)
  -w, --no-password         never prompt for password
  -W, --password            force password prompt
```

Examples

- Create a superuser called "root" that is able login and has a password 'apples'

```
create role root with login superuser password 'apples';
```

- Create role "dba" that has permission to create databases but cannot login.

```
create role dba with createdb;
```

- Create role "joe" and assign him as member to "dba" therefore giving him the ability to create new databases.

```
create role joe with login password 'smile' in role dba;
```

- Create a role called "jane" with a password of 'apples', the account will expire at the end of the year 2023.

```
create role jane with login password 'koolaid' valid until '2023-12-31';
```

Assigning Users Roles As Members

- When we create a role, we are essentially creating a permission “map” - since permissions can be assigned to a role, and users can be members of a role, we can use roles to manage groups of permissions.
- Since permissions are assigned to roles (and not directly to users) we can manage entire groups of users simply by managing a role.
- For example, to take away SUPERUSER access from a 'dba' role, we can just issue a 'ALTER ROLE dba with NOSUPERUSER'
- Similarly, NOLOGIN, NOCREATEDB, NOSUPERUSER, NOCREATEROLE, and NOINHERIT clauses can be used to remove access from users.
- By removing access from a role, we effectively remove access from all users that inherit from that role.
- Once a role has been created, the 'GRANT role TO user' and 'REVOKE role FROM user' statements may be used to grant and revoke membership from roles.
- A role can also be granted permissions to manage other roles: GRANT role to user WITH ADMIN OPTION

Understanding Role Access

- A user can be a member of more than one role.
- The 'INHERIT' property of an account allows explicitly gains access to all privileges it would get for all roles it is a member of.
- When INHERIT is not enabled, a user can change his/her current role using the 'SET ROLE' statement.
- After SET ROLE, permissions checking for SQL commands is carried out as though the named role were the one that had logged in originally.
- Inheritance does not apply to the special role attributes set by CREATE ROLE and ALTER ROLE. For example, being a member of a role with CREATEDB privilege does not immediately grant the ability to create databases, even if INHERIT is set; it would be necessary to become that role via SET ROLE before creating a database.
- The RESET ROLE or SET ROLE NONE statements returns the role to the default.
- If a users membership is revoked from a role that they have 'set' themselves to (select current_user shows that role) then that user will retain access to the role until they do a 'reset role' or disconnect.
- The SESSION keyword causes the change in role to apply to the current session only.
- The LOCAL keyword causes the change in role to apply only to the current transaction.
- You can determine your current session user and current role using the following statement: `SELECT SESSION_USER, CURRENT_USER;`
- You can determine the rights you have on a database objects using the '\dp' meta-command.
- You can obtain a list of roles using the '\du' meta command.
- You can obtain a list of database access permissions from the pg_database table.
- You can obtain a list of roles from pg_roles.

Listing Roles

db01=# \dgS

```
db01=# \dgS
                                                     List of roles
         Role name         |                         Attributes            |           Member of
---------------------------+-----------------------------------------------+-----------------------------
 dba                       | Create DB, Cannot login                       | {}
 jane                      | Password valid until 2023-12-31 00:00:00-08   | {}
 joe                       |                                               | {dba}
 pg_execute_server_program | Cannot login                                  | {}
 pg_monitor                | Cannot login                                  | {pg_read_all_settings,
                           |                                               |  pg_read_all_stats,
                           |                                               |  pg_stat_scan_tables}
 pg_read_all_settings      | Cannot login                                  | {}
 pg_read_all_stats         | Cannot login                                  | {}
 pg_read_server_files      | Cannot login                                  | {}
 pg_signal_backend         | Cannot login                                  | {}
 pg_stat_scan_tables       | Cannot login                                  | {}
 pg_write_server_files     | Cannot login                                  | {}
 postgres                  | Superuser, Create role,                       |
                           | Create DB, Replication, Bypass RLS            | {}
 root                      | Superuser                                     | {}
```

https://www.postgresql.org/docs/current/default-roles.html

Listing Roles Cont'd

db01=# \d pg_roles

```
                      View "pg_catalog.pg_roles"
    Column     |           Type           | Collation | Nullable | Default
---------------+--------------------------+-----------+----------+---------
 rolname       | name                     |           |          |
 rolsuper      | boolean                  |           |          |
 rolinherit    | boolean                  |           |          |
 rolcreaterole | boolean                  |           |          |
 rolcreatedb   | boolean                  |           |          |
 rolcanlogin   | boolean                  |           |          |
 rolreplication| boolean                  |           |          |
 rolconnlimit  | integer                  |           |          |
 rolpassword   | text                     |           |          |
 rolvaliduntil | timestamp with time zone |           |          |
 rolbypassrls  | boolean                  |           |          |
 rolconfig     | text[]                   | C         |          |
 oid           | oid                      |           |          |
```

db01=# \d pg_shadow

```
                      View "pg_catalog.pg_shadow"
    Column    |           Type           | Collation | Nullable | Default
--------------+--------------------------+-----------+----------+---------
 usename      | name                     |           |          |
 usesysid     | oid                      |           |          |
 usecreatedb  | boolean                  |           |          |
 usesuper     | boolean                  |           |          |
 userepl      | boolean                  |           |          |
 usebypassrls | boolean                  |           |          |
 passwd       | text                     | C         |          |
 valuntil     | timestamp with time zone |           |          |
 useconfig    | text[]                   | C         |          |
```

Caveat

- passwords (AWS and similar cloud provider environments)
  - cannot be extracted from pg_shadow
  - cannot be dumped TIP: `pg_dumpall --no-role-passwords`

Additional References For Roles

- https://www.postgresql.org/docs/current/view-pg-user.html
- https://www.postgresql.org/docs/current/view-pg-roles.html
- https://www.postgresql.org/docs/current/catalog-pg-user-mapping.html
- https://www.postgresql.org/docs/current/catalog-pg-db-role-setting.html
- https://www.postgresql.org/docs/current/infoschema-applicable-roles.html
- https://www.postgresql.org/docs/current/infoschema-administrable-role-authorizations.html

Administration: Access Control

About The GRANT and REVOKE Statements

The SQL standard defines the GRANT and REVOKE statements in order to facilitate granting and revoking permissions to/from users.

Access Control

About The GRANT Statement

Description: define access privileges
Syntax:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Access Control

GRANT cont'd

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE }
           routine_name [ ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] ) ] [, ...]
         | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA schema_name [, ...] }
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
```

Access Control

GRANT cont'd

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]

where role_specification can be:

    [ GROUP ] role_name
  | PUBLIC
  | CURRENT_USER
  | SESSION_USER

GRANT role_name [, ...] TO role_name [, ...] [ WITH ADMIN OPTION ]
```

Access Control

About The REVOKE Statement

Syntax:

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...] | ALL [ PRIVILEGES ] }
    ON DATABASE database_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Access Control

REVOKE cont'd

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON DOMAIN domain_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN DATA WRAPPER fdw_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON FOREIGN SERVER server_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE }
         function_name [ ( [ [ argmode ] [ arg_name ] arg_type [, ...] ] ) ] [, ...]
         | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE lang_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Access Control

REVOKE cont'd

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | UPDATE } [, ...] | ALL [ PRIVILEGES ] }
    ON LARGE OBJECT loid [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespace_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON TYPE type_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ ADMIN OPTION FOR ]
    role_name [, ...] FROM role_name [, ...]
    [ CASCADE | RESTRICT ]
```

EXAMPLES

Before you start **(version 15 only)**:

```
/*
GRANT { { CREATE | USAGE } [, ...] | ALL [ PRIVILEGES ] }
    ON SCHEMA schema_name [, ...]
    TO role_specification [, ...] [ WITH GRANT OPTION ]
    [ GRANTED BY role_specification ]
*/

grant all privileges on schema PUBLIC to PUBLIC;
```

Example 1: Granting Privileges

```
---------
-- INITIAL SETUP: execute as postgres
create role usr1 with login password 'usr1';
create role usr2 with login password 'usr2';
create role usr3 with login password 'usr3';
set role usr1;
select *,'created by usr1'::text "comments" into t1 from generate_series(1,1e6);
select *,'created by usr1'::text "comments" into t2 from generate_series(1,1e6);
select *,'created by usr1'::text "comments" into t3 from generate_series(1,1e6);
create sequence seq1 as integer increment by 2 maxvalue 100 cycle;

analyze verbose t1,t2,t3;
---------
```

EXAMPLES Cont'd

```
---------
grant select, insert, delete, truncate on table t1, t2 to usr2;
set role usr2;
with a as (select count(*) from t1),
     b as (select count(*) from t2)
select a.count "table t1",b.count "table t2" from a,b;
---------
set role usr1;
grant all privileges (generate_series) on table t3 to usr2;
set role usr2;
-- fails
select generate_series,"comments" from t3 order by random() limit 10;
-- succeeds
select generate_series from t3 order by random() limit 10;
---------
set role usr1;
grant usage on sequence seq1 to usr2;
set role usr2;
select nextval('seq1') from generate_series(1,5);
---------
set role usr1;
grant all privileges on t1,t2,t3,seq1 to usr3;
set role usr3;
with a as (select count(*) from t1),
     b as (select count(*) from t2),
     c as (select count(*) from t3),
select a.count "table t1",b.count "table t2", c.count "table t3" from a,b,c;
with a as (select nextval('seq1') from generate_series(1,1000))
select * from a order by random() limit 10;
```

EXAMPLES Cont'd

```
\dp

               Access privileges
 Schema | Name |   Type   | Access privileges | Column privileges | Policies
--------+------+----------+-------------------+-------------------+----------
 public | seq1 | sequence | usr1=rwU/usr1    +|                   |
        |      |          | usr2=rU/usr1     +|                   |
        |      |          | usr3=rwU/usr1     |                   |
 public | t1   | table    | usr1=arwdDxt/usr1+|                   |
        |      |          | usr2=ardD/usr1   +|                   |
        |      |          | usr3=arwdDxt/usr1 |                   |
 public | t2   | table    | usr1=arwdDxt/usr1+|                   |
        |      |          | usr2=ardD/usr1   +|                   |
        |      |          | usr3=arwdDxt/usr1 |                   |
 public | t3   | table    | usr1=arwdDxt/usr1+| generate_series: +|
        |      |          | usr3=arwdDxt/usr1 |   usr2=arwx/usr1  |
```

https://www.postgresql.org/docs/current/ddl-priv.html

EXAMPLES Cont'd

Example 2:

1. Creating and setting privileges for ROLE *usr1*.
2. Creating and populating table *t1*.

```
-------------------
-- login as superuser "postgres"
drop table if exists t1;

-------------------
BEGIN;
    -- create and populate table
    drop table if exists t1;
    select  generate_series::bigint as id,
        'hello world' as comments,
            now()::timestamp(0) as t_stamp
        into table t1
        from generate_series(1,1e6);
    select * from t1 order by random() limit 10;
COMMIT;
-------------------
```

EXAMPLES Cont'd

Example 2 Cont'd: perform an INSERT

```
-------------------
set role usr1;
\d t1
select * from t1 order by random() limit 10;
-------------------
reset role;
grant select on table t1 to usr1;
-------------------
-- fails
set role usr1;
select * from t1 order by random() limit 10;
with a as (select max(id)+1 as x from t1)
    insert into t1 (id) select x from a returning *;
-------------------
reset role;
grant insert on table t1 to usr1;
-------------------
-- succeeds
set role usr1;
with a as (select max(id)+1 as x from t1)
    insert into t1 (id) select x from a returning *;
-------------------
```

EXAMPLES Cont'd

Example 2 Cont'd: Adding constraints and performing UPDATE

```
-------------------
reset role;
select max(id) from t1;
alter table t1
    add primary key(id),
    alter column id add generated by default as identity (start with 1000103);

\d
\d t1
\d t1_id_seq
-------------------
set role usr1;
select * from t1 order by random() limit 10;

with a as (select max(id)+1 as x from t1)
    insert into t1 values(default,'this row now increments automatically',now()) returning *;

select * from t1 order by id desc limit 5;

update t1 set id=1000002,
              comments='id has been edited',
              t_stamp=now()
    where id=1000002
-------------------
reset role;
grant update on table t1 to usr1;
-------------------
set role usr1;
update t1 set id=1000002,
              comments='id has been edited',
              t_stamp=now()
    where id=1000002
    returning *;
-------------------
```

EXAMPLES Cont'd

Example 2 Cont'd: Moving table *t1* to a new SCHEMA

```
-------------------
-- fails
reset role;
create schema usr1;
alter table t1 set schema usr1;

set role usr1;
show search_path;
\d
\d usr1.
select * from t1 order by random() limit 10;
-------------------
-- succeeds
reset role;
grant usage on schema usr1 to usr1;

set role usr1;
select * from t1 order by random() limit 10;
with a as (select max(id)+1 as x from t1)
    insert into t1 values(default,'this row now increments automatically',now()) returning *;
-------------------
-- review environment
reset role;
select current_user, session_user;
alter role usr1 with login;
set session authorization usr1;
select current_user, session_user;
\d
```

EXAMPLES Cont'd

EXAMPLE 3:

1. replicating the aforementioned operations in a more succinct manner.
2. resetting the default search path.

```
-------------------
\c - postgres

revoke all privileges on schema usr1 from usr2 cascade;

revoke all privileges
    on all tables in schema public,usr1
from usr2 cascade;

revoke all privileges
    on all sequences in schema public,usr1
from usr2 cascade;
-------------------
reassign owned by usr2 to usr1;
drop role if exists usr2;
create role usr2 with login password 'usr2';
-------------------
grant select, insert, update on table usr1.t1 to usr2;
grant usage on schema usr1 to usr2;
alter role usr2 set search_path=usr1,public;
-------------------
\c 'host=localhost user=usr2 password=usr2 dbname=db01'
select * from usr1.t1 order by random() limit 10;
-------------------
```

EXAMPLES Cont'd

EXAMPLE 4: Leveraging ROLE membership

```
-------------------
\c 'host=localhost user=postgres password=postgres dbname=db01'
grant usage on schema usr1 to usr3;
grant all privileges on all tables in schema usr1 to usr3;
grant all privileges on all sequences in schema usr1 to usr3;
alter role usr3 with login password 'usr3';
alter database db01 set search_path=usr1,public;
show search_path;
\c
show search_path;
-------------------
\c 'host=localhost user=usr3 password=usr3 dbname=db01'
\dp+
select * from t1 order by random() limit 10;

with a as (select max(id)+1 as x from t1)
    insert into t1 values(default,'this row was inserted by usr3',now()) returning *;
-------------------
```

EXAMPLES Cont'd

EXAMPLE 5: REVOKING PRIVILEGES Restricting access to a database

```
-- ATTENTION: THIS WAS THE FORMER METHOD OF HARDENING PRIVILEGES
-- FOR VERSIONS OLDER THAN POSTGRES VERSION 15

-------------------
-- strip all privileges, valid for postgres < version 15
\c - postgres
create database db02
revoke all privileges on database db02 from PUBLIC cascade;
\c db02
drop schema public;
------------------------
# works
psql 'host=pg1 dbname=db01 user=usr1 password=usr1'
# fails
psql 'host=pg1 dbname=db02 user=usr1 password=usr1'
psql 'host=pg1 dbname=db02 user=usr2 password=usr2'
psql 'host=pg1 dbname=db02 user=usr3 password=usr3'
------------------------
-- as postgres
drop schema public;
create schema usr1 authorization usr1;
create schema usr2 authorization usr2;
create schema usr3 authorization usr3;
grant all privileges on database db02 to usr1, usr2;

# works
psql 'host=pg1 dbname=db02 user=usr1 password=usr1' \
    -c "select *,'created by usr1'::text into t1 from generate_series(1,1e6)"
psql 'host=pg1 dbname=db02 user=usr2 password=usr2' \
    -c "select *,'created by usr2'::text into t1 from generate_series(1,1e6)"
psql 'host=pg1 dbname=db02 user=usr3 password=usr3' \
    -c "select *,'created by usr3'::text into t1 from generate_series(1,1e6)"
```

Access Control

CAVEAT

- A special user class called 'PUBLIC' exists.
- Before version 15: All users had access to schema 'PUBLIC'.
- As of version 15: Users no longer have default access privileges.
  - Restores pre-version 15 behaviour: `grant all privileges on schema PUBLIC to PUBLIC;`
- The addition of 'WITH GRANT OPTION' allows that role to pass on its privilege to others (GRANT that privilege)
- Since function overloading is supported in PostgreSQL, the arguments provided to a function may be supplied when granting privileges to a function.
- In addition to the SQL-standard privilege system available through GRANT, tables can have row security policies that restrict, on a per-user basis, which rows can be returned by normal queries or inserted, updated, or deleted by data modification commands.

PostgreSQL Operations

Monitoring

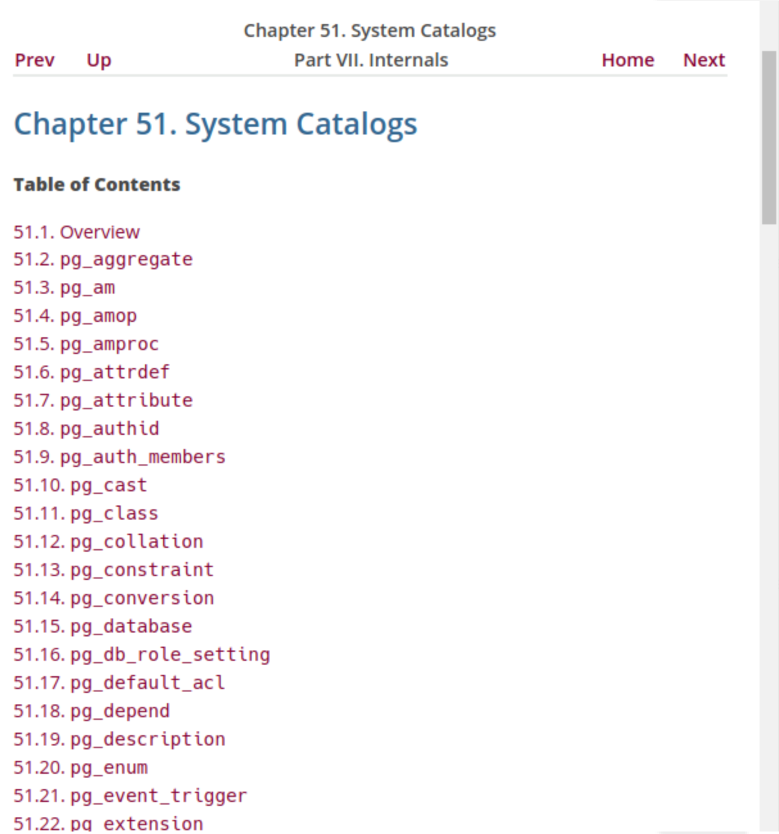
Monitoring Postgres Metrics

About

- The Postgres System Catalogs & Information Schema
- The Statistics Collector
- Commonly Used Metrics
- Extensions
    - About
    - Where To Get It
    - Commonly Used Extensions
- PostgreSQL Administration ROLEs
- Command Line Utilities
- Analysis/EXPLAIN

Monitoring Postgres Metrics Cont'd

PostgreSQL System Catalogs & Information Schema



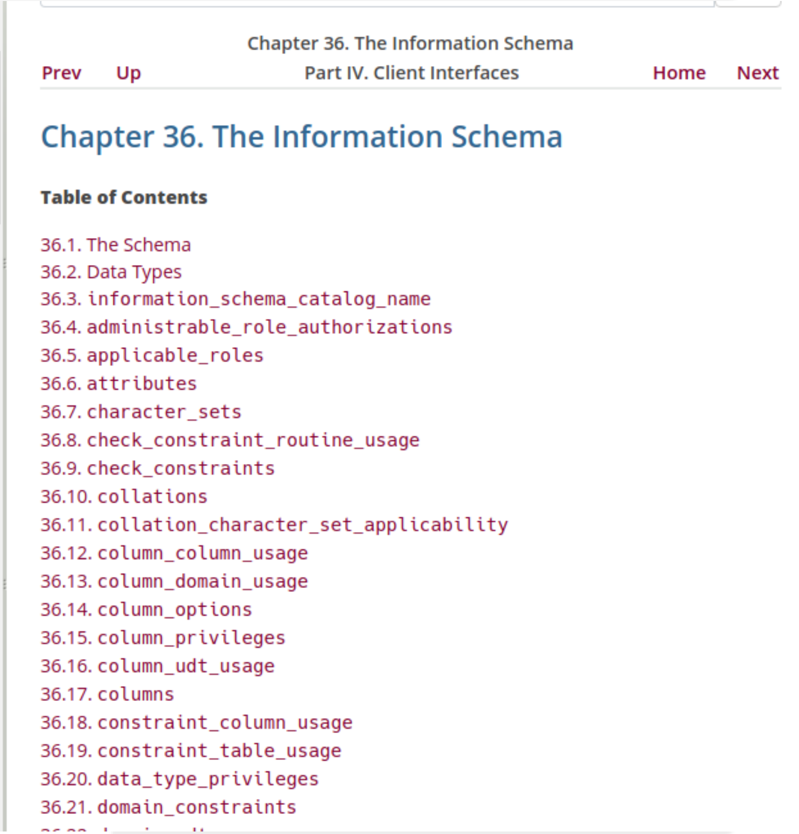
Chapter 51. System Catalogs

Prev Up Part VII. Internals Home Next

Chapter 51. System Catalogs

Table of Contents

51.1. Overview
51.2. pg_aggregate
51.3. pg_am
51.4. pg_amop
51.5. pg_amproc
51.6. pg_attrdef
51.7. pg_attribute
51.8. pg_authid
51.9. pg_auth_members
51.10. pg_cast
51.11. pg_class
51.12. pg_collation
51.13. pg_constraint
51.14. pg_conversion
51.15. pg_database
51.16. pg_db_role_setting
51.17. pg_default_acl
51.18. pg_depend
51.19. pg_description
51.20. pg_enum
51.21. pg_event_trigger
51.22. pg_extension



Chapter 36. The Information Schema

Prev Up Part IV. Client Interfaces Home Next

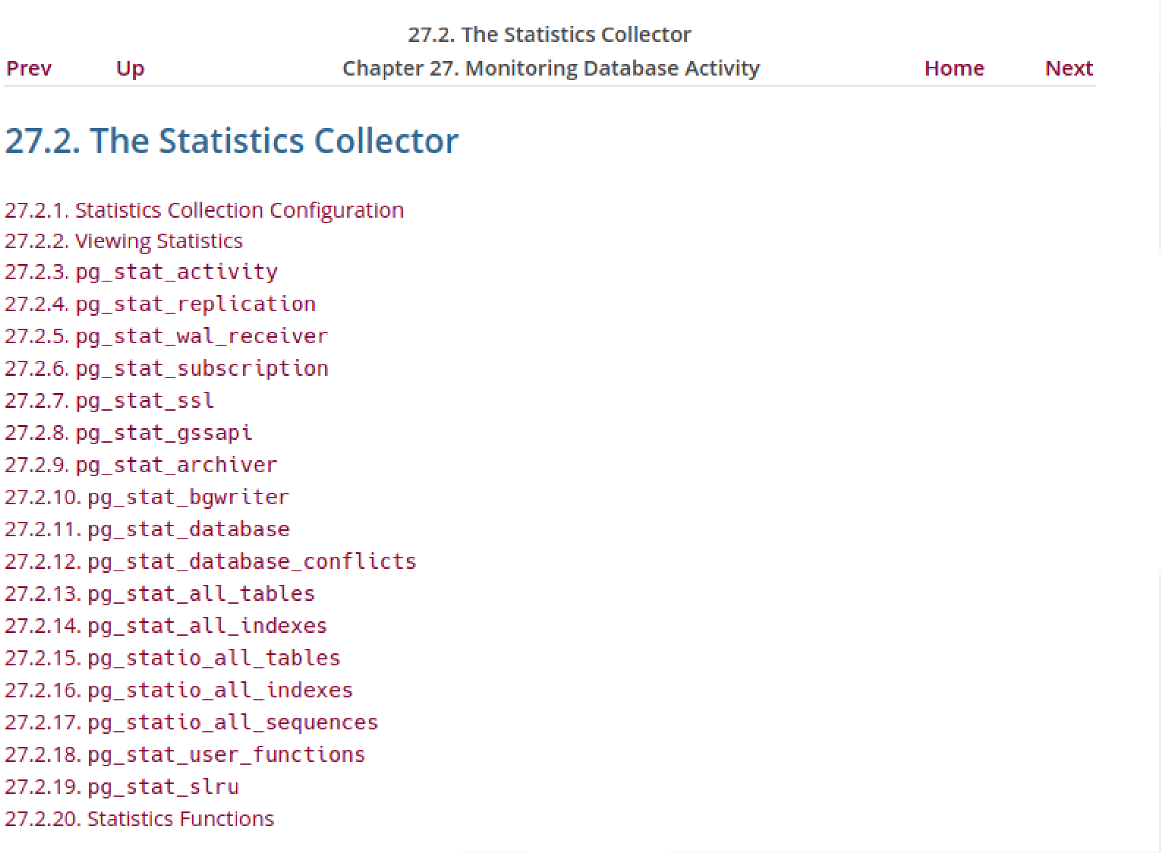
Chapter 36. The Information Schema

Table of Contents

36.1. The Schema
36.2. Data Types
36.3. information_schema_catalog_name
36.4. administrable_role_authorizations
36.5. applicable_roles
36.6. attributes
36.7. character_sets
36.8. check_constraint_routine_usage
36.9. check_constraints
36.10. collations
36.11. collation_character_set_applicability
36.12. column_column_usage
36.13. column_domain_usage
36.14. column_options
36.15. column_privileges
36.16. column_udt_usage
36.17. columns
36.18. constraint_column_usage
36.19. constraint_table_usage
36.20. data_type_privileges
36.21. domain_constraints

Monitoring Postgres Metrics Cont'd

Metrics: The Statistics Collector



27.2. The Statistics Collector

Prev Up Chapter 27. Monitoring Database Activity Home Next

27.2. The Statistics Collector

https://www.postgresql.org/docs/current/monitoring.html

Monitoring Postgres Metrics Cont'd

Commonly Used Metrics

Real-time metrics:

- pg_locks
- pg_stat_activity

Commonly used metrics for analysis:

- pg_class
- pg_roles
- pg_settings (SHOW status)
- pg_stat_all_tables (pg_stat_user_tables)
- pg_stat_all_indexes (pg_stat_user_indexes)
- pg_statio_all_tables (pg_statio_user_indexes)
- pg_statio_all_indexes (pg_statio_user_indexes)
- pg_stat_bgwriter
- pg_stat_database
- pg_stat_progress_vacuum
- pg_stat_ssl
- pg_stat_user_functions

https://www.postgresql.org/docs/current/catalogs.html

Monitoring Postgres Metrics Cont'd

System Views, Functions And Logging

- statistics monitoring of relations
  - pg_stat_user_indexes
  - pg_stat_user_tables
  - pg_statio_user_tables
  - pg_statio_user_indexes
- replication

  ```
  select * from pg_stat_replication;
  select * from pg_replication_slots;
  select * from pg_get_replication_slots();
  --
  select
  case
      when pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn()
      then 0
  else extract (EPOCH FROM now() - pg_last_xact_replay_timestamp())
  end as log_delay;
  ```
- postgres logging

https://www.postgresql.org/docs/current/monitoring.html

Monitoring Postgres Metrics Cont'd

Extensions:

PostgreSQL is designed to be easily extensible: PostgreSQL extensions are SQL objects bundled into a single package which can be loaded or removed from your database at will. Once loaded, extensions can enhance the RDBMS with new and enhanced functionality.

Where To Get It:

1. The PostgreSQL Community Repository Portal
2. The contributions modules shipped with PostgreSQL, https://www.postgresql.org/docs/current/contrib.html
3. The PostgreSQL Extension Network, https://pgxn.org/
4. Third party portals/projects.

Commonly Used Extensions (Monitoring And Tuning):

- pg_stat_statement, https://www.postgresql.org/docs/current/pgstatstatements.html
- auto_explain, https://www.postgresql.org/docs/current/auto-explain.html
- pg_repack, https://github.com/reorg/pg_repack
- pg_stat_monitor, https://pgxn.org/dist/pg_stat_monitor/

Monitoring Postgres Metrics Cont'd

Extensions

```
EXTENSION           COMMENTS
autoexplain         Provides a means for logging execution plans
pageinspect         Allows you to inspect the contents of database pages
pg_buffercache      Examine what's happening in the shared buffer cache in real time
pg_freespacemap     Examine the free space map
pgrowlocks          Show row locking information for a specified table
pg_stat_statements  Tracking execution statistics of all SQL statements executed by the ser
pgstattuple         Obtain tuple-level statistics
pg_visibility       Examining the visibility and integrity map  and page-level visibility i
sslinfo             Provides information about the SSL certificate of client connection dur
```

https://www.postgresql.org/docs/current/contrib.html

Monitoring Postgres Metrics Cont'd

Administration:

```
CREATE EXTENSION [ IF NOT EXISTS ] extension_name
    [ WITH ] [ SCHEMA schema_name ]
             [ VERSION version ]
             [ CASCADE ]

DROP EXTENSION [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]

ALTER EXTENSION name UPDATE [ TO new_version ]
ALTER EXTENSION name SET SCHEMA new_schema
ALTER EXTENSION name ADD member_object
ALTER EXTENSION name DROP member_object
```

Querying Extensions:

```
select * from pg_extension order by 2;
select * from pg_available_extensions() order by 1,2;
select * from pg_available_extension_versions() order by 1,2;
```

Monitoring Postgres Metrics Cont'd

PostgreSQL Administration ROLES

```
                                    List of roles
      Role name          |                    Attributes                    |          Member of
-------------------------+--------------------------------------------------+----------------------------
 pg_execute_server_program | Cannot login                                  | {}
 pg_monitor              | Cannot login                                     | {pg_read_all_settings,
                         |                                                  |  pg_read_all_stats,
                         |                                                  |  pg_stat_scan_tables}
 pg_read_all_settings    | Cannot login                                     | {}
 pg_read_all_stats       | Cannot login                                     | {}
 pg_read_server_files    | Cannot login                                     | {}
 pg_signal_backend       | Cannot login                                     | {}
 pg_stat_scan_tables     | Cannot login                                     | {}
 pg_write_server_files   | Cannot login                                     | {}
```

https://www.postgresql.org/docs/current/default-roles.html

Monitoring Postgres Metrics Cont'd

pg_execute_server_program: Allow executing programs on the database server as the user the database runs as with COPY and other functions which allow executing a server-side program.

Step 1:

```
COPY table_name [ ( column_name [, ...] ) ]
    FROM { 'filename' | PROGRAM 'command' | STDIN }
    [ [ WITH ] ( option [, ...] ) ]
    [ WHERE condition ]
```

Step 2:

```
(for u in $(seq 0 100)
do
    echo -e "$u \t hello world"
done) | gzip -> /tmp/data.gz
```

Step 3:

```
create role usr1 login in role pg_execute_server_program;
set session authorization usr1;

create table t1(c1 integer,c2 varchar(25));
copy t1 from program 'gunzip < /tmp/data.gz';
```

Monitoring Postgres Metrics Cont'd

pg_monitor: Read/execute various monitoring views and functions. This role is a member of pg_read_all_settings, pg_read_all_stats and pg_stat_scan_tables.

```
create role usr2 login in role pg_monitor;
set session authorization usr2;

db01=> select * from pg_stat_user_tables;
-[ RECORD 1 ]-------+-------
relid               | 35249
schemaname          | public
relname             | t1
seq_scan            | 1
seq_tup_read        | 101
idx_scan            |
idx_tup_fetch       |
n_tup_ins           | 101
n_tup_upd           | 0
n_tup_del           | 0
n_tup_hot_upd       | 0
n_live_tup          | 101
n_dead_tup          | 0
n_mod_since_analyze | 101
n_ins_since_vacuum  | 101
last_vacuum         |
last_autovacuum     |
last_analyze        |
last_autoanalyze    |
vacuum_count        | 0
autovacuum_count    | 0
analyze_count       | 0
autoanalyze_count   | 0
```

Monitoring Postgres Metrics Cont'd

pg_read_all_settings: Read all pg*stat** views and use various statistics related extensions, even those normally visible only to superusers.

```
create role usr4 login in role pg_read_all_stats;
set session authorization usr4;

select * from pg_statio_all_tables;
```

Monitoring Postgres Metrics Cont'd

pg_read_all_stats: Read all pg*stat** views and use various statistics related extensions, even those normally visible only to superusers.

```
create role usr4 login in role pg_read_all_stats;
set session authorization usr4;

select * from pg_statio_all_tables;
```

Monitoring Postgres Metrics Cont'd

pg_read_server_files: Allow reading files from any location the database can access on the server with COPY and other file-access functions.

```
(for u in $(seq 0 100)
do
    echo -e "$u \t hello world"
done) > /tmp/data
```

```
create role usr5 login in role pg_read_server_files;
set session authorization usr5;

create table t2(c1 integer,c2 varchar(25));

copy t2 from '/tmp/data';
```

Monitoring Postgres Metrics Cont'd

pg_signal_backend: Signal another backend to cancel a query or terminate its session.

Step 1:

```
--
-- session 1
--
create role usr6 login in role pg_signal_backend;
set session authorization usr6;
```

Step 2:

```
--
-- session 2
--
\c 'dbname=db01 application_name=sleep user=usr1'
select pg_sleep_for('1 day');
```

Step 3:

```
--
-- session 1
--
select pg_cancel_backend(pid) from pg_stat_activity where application_name='sleep';
```

Monitoring Postgres Metrics Cont'd

pg_stat_scan_tables: Execute monitoring functions that may take ACCESS SHARE locks on tables, potentially for a long time.

```
create extensions pgstattuple;
```

```
create role usr7 login in role pg_stat_scan_tables;
set session authorization usr7;
```

```
select * from pgstattuple('pg_catalog.pg_proc');
```

Monitoring Postgres Metrics Cont'd

pg_write_server_files: Allow writing to files in any location the database can access on the server with COPY and other file-access functions.

```
create role usr8 login in role pg_write_server_files;
set session authorization usr8;
```

```
select *,'hello world'::text into t3 from (select * from generate_series(1,1e6))t;
copy t3 to '/tmp/data.out';
```

Monitoring Postgres Metrics Cont'd

Analysis: About The Query Planner

- Server recieves query
- The parser scans through the query and checks it for syntax errors
- The parser converts it into a parse tree
- Query optimizer develops a plan of execution for the query:
    - Examines data in tables
    - Reviews all possible execution plans
    - Determines order to execute including use of indexes
    - Breaks down complex queries into simple steps and finding the “cheapest” ones
    - Different query operators have different cost estimates
    - Query cost:
        - Disk IO
        - CPU usage

https://www.postgresql.org/docs/current/planner-stats.html

Monitoring Postgres Metrics Cont'd

Analysis: About EXPLAIN

```
Command:     EXPLAIN
Description: show the execution plan of a statement
Syntax:
EXPLAIN [ ( option [, ...] ) ] statement
EXPLAIN [ ANALYZE ] [ VERBOSE ] statement

where option can be one of:

    ANALYZE [ boolean ]
    VERBOSE [ boolean ]
    COSTS [ boolean ]
    SETTINGS [ boolean ]
    BUFFERS [ boolean ]
    WAL [ boolean ]
    TIMING [ boolean ]
    SUMMARY [ boolean ]
    FORMAT { TEXT | XML | JSON | YAML }
```

https://www.postgresql.org/docs/current/using-explain.html

Monitoring Postgres Metrics Cont'd

Analysis: About PostgreSQL Query Operators

- Example/DEMO relations
- seq scan
- sort
- aggregate
- unique
- union
- setop: EXCEPT
- result
- subquery scan
- tid scan
- index scan
- limit
- hashaggregate
- group by
- append
- setop: INTERSECT
- nestedloop
- recursive

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

Demonstration:

```
select generate_series as id, 'hello world'::text as comments into t1
    from (select * from generate_series(1,1e6))t;

select generate_series as id, 'hello world'::text as comments into t2
    from (select * from generate_series(1,1000))t;

alter table t1 add primary key(id);
alter table t2 add foreign key(id) references t1(id);
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- seq scan
db01=# explain select * from t1;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=18)

db01=# explain select * from t1 where comments='hello';
                              QUERY PLAN
----------------------------------------------------------------------
 Gather  (cost=1000.00..12578.43 rows=1 width=18)
   Workers Planned: 2
   ->  Parallel Seq Scan on t1  (cost=0.00..11578.33 rows=1 width=18)
         Filter: (comments = 'hello'::text)
```

```
-- index scan
db01=# explain select * from t1 where id=1000;
                            QUERY PLAN
-------------------------------------------------------------------
 Index Scan using t1_pkey on t1  (cost=0.42..8.44 rows=1 width=18)
   Index Cond: (id = '1000'::numeric)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- sort operator
db01=# explain select * from t1 order by comments desc;
                            QUERY PLAN
---------------------------------------------------------------------
 Sort  (cost=136537.84..139037.84 rows=1000000 width=18)
   Sort Key: comments DESC
   ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=18)
```

```
-- limit
db01=# explain select * from t1 limit 10;
                            QUERY PLAN
---------------------------------------------------------------------
 Limit  (cost=0.00..0.16 rows=10 width=18)
   ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=18)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- aggregate
db01=# explain select count(id) from t1;
                                      QUERY PLAN
--------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=12578.55..12578.56 rows=1 width=8)
   ->  Gather  (cost=12578.33..12578.54 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=11578.33..11578.34 rows=1 width=8)
               ->  Parallel Seq Scan on t1  (cost=0.00..10536.67 rows=416667 width=6)
```

```
-- alternate aggregate behaviour
db01=# explain select min(id), max(id) from t1;
                                          QUERY PLAN
-----------------------------------------------------------------------------------------------
 Result  (cost=0.46..0.47 rows=1 width=32)
   InitPlan 1 (returns $0)
     ->  Limit  (cost=0.42..0.46 rows=1 width=6)
           ->  Index Only Scan using t1_pkey on t1  (cost=0.42..34853.43 rows=1000000 width=6)
                 Index Cond: (id IS NOT NULL)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- hashaggregate, group key operators
db01=# explain select distinct (comments) from t1;
                            QUERY PLAN
-------------------------------------------------------------------
 HashAggregate  (cost=18870.00..18870.01 rows=1 width=12)
   Group Key: comments
   ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=12)
```

```
-- unique operator
db01=# explain select distinct * from t1;
                               QUERY PLAN
-------------------------------------------------------------------------
 Unique  (cost=136537.84..144037.84 rows=1000000 width=18)
   ->  Sort  (cost=136537.84..139037.84 rows=1000000 width=18)
         Sort Key: id, comments
         ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=18)

db01=# explain select distinct (id) from t1;
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Unique  (cost=0.42..34853.43 rows=1000000 width=6)
   ->  Index Only Scan using t1_pkey on t1  (cost=0.42..32353.42 rows=1000000 width=6)
```

PERCONA
TRAINING

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- group by operator on column without an index
db01=# explain select comments from t1 group by comments;
                                         QUERY PLAN
---------------------------------------------------------------------------------------------
 Group  (cost=12578.38..12578.62 rows=1 width=12)
   Group Key: comments
   ->  Gather Merge  (cost=12578.38..12578.61 rows=2 width=12)
         Workers Planned: 2
         ->  Sort  (cost=11578.35..11578.36 rows=1 width=12)
               Sort Key: comments
               ->  Partial HashAggregate  (cost=11578.33..11578.34 rows=1 width=12)
                     Group Key: comments
                     ->  Parallel Seq Scan on t1  (cost=0.00..10536.67 rows=416667 width=12)
```

```
-- group by operator on column with an index
db01=# explain select id from t1 group by id;
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Group  (cost=0.42..34853.43 rows=1000000 width=6)
   Group Key: id
   ->  Index Only Scan using t1_pkey on t1  (cost=0.42..32353.42 rows=1000000 width=6)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- append operator, union
db01=# explain select id from t1 union select id from t2;
                                       QUERY PLAN
----------------------------------------------------------------------------------------
 Unique  (cost=155118.72..160123.72 rows=1001000 width=32)
   ->  Sort  (cost=155118.72..157621.22 rows=1001000 width=32)
         Sort Key: t1.id
         ->  Append  (cost=0.00..31403.50 rows=1001000 width=32)
               ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=6)
               ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..28.50 rows=1000 width=32)
                     ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4)
```

```
-- append operator, union all
db01=# explain select id from t1 union all select id from t2;
                                 QUERY PLAN
----------------------------------------------------------------------------
 Append  (cost=0.00..31403.50 rows=1001000 width=32)
   ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=6)
   ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..28.50 rows=1000 width=32)
         ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- SetOp EXCEPT
db01=# explain select id from t1 except select id from t2;
                                          QUERY PLAN
----------------------------------------------------------------------------------------------
 SetOp Except  (cost=185915.22..190920.22 rows=1000000 width=36)
   ->  Sort  (cost=185915.22..188417.72 rows=1001000 width=36)
         Sort Key: "*SELECT* 1".id
         ->  Append  (cost=0.00..31403.50 rows=1001000 width=36)
               ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..26370.00 rows=1000000 width=10)
                     ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=6)
               ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..28.50 rows=1000 width=36)
                     ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4)
```

```
-- SetOp INTERSECT
db01=# explain select id from t1 intersect select id from t2;
                                       QUERY PLAN
----------------------------------------------------------------------------------------
 HashSetOp Intersect  (cost=0.00..33906.00 rows=1000 width=36)
   ->  Append  (cost=0.00..31403.50 rows=1001000 width=36)
         ->  Subquery Scan on "*SELECT* 2"  (cost=0.00..28.50 rows=1000 width=36)
               ->  Seq Scan on t2  (cost=0.00..16.00 rows=1000 width=4)
         ->  Subquery Scan on "*SELECT* 1"  (cost=0.00..26370.00 rows=1000000 width=10)
               ->  Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=6)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- result operator
-- EX 1: no result operator used
db01=# explain select * from t1 where 1=1;
                         QUERY PLAN
-------------------------------------------------------------
 Seq Scan on t1  (cost=0.00..16370.00 rows=1000000 width=18)

-- EX 2
db01=# explain select * from t1 where 1<1;
                QUERY PLAN
------------------------------------------
 Result  (cost=0.00..0.00 rows=0 width=0)
   One-Time Filter: false

-- EX 3
db01=# explain select * from t1 where 'lunch'='dinner';
                QUERY PLAN
------------------------------------------
 Result  (cost=0.00..0.00 rows=0 width=0)
   One-Time Filter: false
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- nested loop: natural join
db01=# explain select a.id,a.comments, b.comments
from t1 as a
natural join t2 as b;
                                 QUERY PLAN
----------------------------------------------------------------------------
 Nested Loop  (cost=0.43..7569.50 rows=1000 width=30)
   ->  Seq Scan on t2 b  (cost=0.00..16.00 rows=1000 width=16)
   ->  Index Scan using t1_pkey on t1 a  (cost=0.43..7.54 rows=1 width=18)
         Index Cond: (id = (b.id)::numeric)
         Filter: (b.comments = comments)
```

```
db01=# explain select a.id,a.comments, b.comments
from t1 as a
join t2 as b using(id);
                                 QUERY PLAN
----------------------------------------------------------------------------
 Nested Loop  (cost=0.43..7557.00 rows=1000 width=30)
   ->  Seq Scan on t2 b  (cost=0.00..16.00 rows=1000 width=16)
   ->  Index Scan using t1_pkey on t1 a  (cost=0.43..7.54 rows=1 width=18)
         Index Cond: (id = (b.id)::numeric)

db01=# explain select a.id,a.comments,b.comments from t1 as a ,t2 as b where a.id=b.id;
                                 QUERY PLAN
----------------------------------------------------------------------------
 Nested Loop  (cost=0.43..7557.00 rows=1000 width=30)
   ->  Seq Scan on t2 b  (cost=0.00..16.00 rows=1000 width=16)
   ->  Index Scan using t1_pkey on t1 a  (cost=0.43..7.54 rows=1 width=18)
         Index Cond: (id = (b.id)::numeric)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- subquery scan and subplan operators
-- EX 1: subselect
db01=# explain select * from t1 where id in (select id from t2 order by random() limit 10);
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop  (cost=40.69..124.81 rows=10 width=18)
   ->  HashAggregate  (cost=40.26..40.36 rows=10 width=4)
         Group Key: ("ANY_subquery".id)::numeric
         ->  Subquery Scan on "ANY_subquery"  (cost=40.11..40.23 rows=10 width=4)
               ->  Limit  (cost=40.11..40.13 rows=10 width=12)
                     ->  Sort  (cost=40.11..42.61 rows=1000 width=12)
                           Sort Key: (random())
                           ->  Seq Scan on t2  (cost=0.00..18.50 rows=1000 width=12)
   ->  Index Scan using t1_pkey on t1  (cost=0.43..8.45 rows=1 width=18)
         Index Cond: (id = ("ANY_subquery".id)::numeric)
```

```
-- EX 2: Common Table Expression
db01=# explain with a as (select id from t2 order by random() limit 10)
select * from t1 join a using (id);
                                QUERY PLAN
---------------------------------------------------------------------------
 Nested Loop  (cost=40.56..124.78 rows=10 width=18)
   CTE a
     ->  Limit  (cost=40.11..40.13 rows=10 width=12)
           ->  Sort  (cost=40.11..42.61 rows=1000 width=12)
                 Sort Key: (random())
                 ->  Seq Scan on t2  (cost=0.00..18.50 rows=1000 width=12)
   ->  CTE Scan on a  (cost=0.00..0.20 rows=10 width=4)
   ->  Index Scan using t1_pkey on t1  (cost=0.43..8.45 rows=1 width=18)
         Index Cond: (id = (a.id)::numeric)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- recursive
db01=# explain with recursive t(n) as (
db01(#     values (1)
db01(#   union all
db01(#     select n+1 from t where n < 100
db01(# )
db01-# select sum(n) from t;
                               QUERY PLAN
-------------------------------------------------------------------------
 Aggregate  (cost=3.65..3.66 rows=1 width=8)
   CTE t
     ->  Recursive Union  (cost=0.00..2.95 rows=31 width=4)
           ->  Result  (cost=0.00..0.01 rows=1 width=4)
           ->  WorkTable Scan on t t_1  (cost=0.00..0.23 rows=3 width=4)
                 Filter: (n < 100)
   ->  CTE Scan on t  (cost=0.00..0.62 rows=31 width=4)
```

Monitoring Postgres Metrics Cont'd

Analysis: PostgreSQL Query Operators Cont'd

```
-- tid scan operator
db01=# select ctid,id from t1 order by random() limit 5;
    ctid    |   id
------------+--------
 (1677,10)  | 263299
 (320,154)  |  50394
 (1052,144) | 165308
 (123,95)   |  19406
 (4967,1)   | 779820
```

```
db01=# explain select * from t1 where id=263299;
                            QUERY PLAN
-------------------------------------------------------------------
 Index Scan using t1_pkey on t1  (cost=0.42..8.44 rows=1 width=18)
   Index Cond: (id = '263299'::numeric)
```

```
db01=# explain select * from t1 where ctid='(1677,10)';
                    QUERY PLAN
---------------------------------------------------
 Tid Scan on t1  (cost=0.00..4.01 rows=1 width=18)
   TID Cond: (ctid = '(1677,10)'::tid)
```

PostgreSQL Operations

Troubleshooting

Troubleshooting

OS COMMAND LINE UTILITY	COMMENTS
atop	Advanced System & Process Monitor
dstat	Versatile Tool For Generating System Resource Statistics
htop	Interactive Process Viewer
iotop	Simple Top-Like I/O Monitor Does Not Work Inside A Container
iostat	Report Central Processing Unit (Cpu) Statistics And Input/Output Statistics For Devices And Partitions
netstat	Print Network Connections, Routing Tables, Interface Statistics, Masquerade Connections, And Multicast Memberships
ps	Report A Snapshot Of The Current Processes
sar	Collect, Report, Or Save System Activity Information
ss	Another Utility To Investigate Sockets
top	Display Linux Processes
vmstat	Report Virtual Memory Statistics

POSTGRES CLI	COMMENTS
pg_activity	htop like application for PostgreSQL server activity monitoring
pg_controldata	displays control information of a PostgreSQL database cluster.
pg_isready	test if accepting connections
pg_repack	repack tables and indexes
pg_top	monitors a PostgreSQL database cluster
pg_ctl	a utility to initialize, start, stop, or control a PostgreSQL server

ATTENTION:
- package name in CENTOS it is "pg_activity", in Ubuntu it is "pg-activity"
- package name CENTOS it is "pg_top", in Ubuntu it is "pgtop"

Troubleshooting Cont'd

EXAMPLE: CLI Installation

```
# CENTOS 8
dnf update -y
dnf install -y epel-release
dnf update -y

dnf install -y atop dstat htop iotop sysstat net-tools pg_activity pg_repack_15

updatedb
```

Troubleshooting Cont'd

COMMONLY USED MONITORING UTILITIES

LINUX

- ps
- free
- top
- sar
- dstat

POSTGRES

- pg_activity
- pg_top

Troubleshooting Cont'd

BENCHMARKING EXAMPLE

```
#
# Recall, initialize:
#   pgbench -h <myhost> -i <mydatabase>
#
#   export PGHOST=pg1 PGUSER=postgres PGPASSWORD=postgres
#
pgbench -i pgbench

#
# Alternate initialization
#
pgbench -i --foreign-keys -s 3 pgbench

#
# Review database
#
pg_dump -Fc pgbench -f /dev/stdout

#
# Benchmarking
#
pgbench -h pg1 -c 4 -j 1 -T 600 -b tpcb-like pgbench
```

TIP: monitoring all processes as LINUX user "postgres"

Troubleshooting Cont'd

MOST COMMON MITIGATION ISSUES

- Logging

```
- tail -f <postgres.log> | grep -E 'ERROR|FATAL'
```

- Terminating Session Connections

```
- select pg_cancel_backend(pid) from pg_stat_activity where ...;
- select pg_terminate_backend(pid) from pg_stat_activity where ...;
- kill -15 <PID>
- DANGER!: kill -09 <PID>
```

- Terminating Service

```
- systemctl stop ...
- pg_ctl -m [smart|fast|immediate] stop -D PGDATA
- killall -w [postgres (debian) | postmaster (redhat)]
```

- Bloat Mitigation

```
- pg_repack -h <host> -U postgres -a
```

- Wraparound TXID

Troubleshooting Cont'd

TERMINATING CONNECTIONS

```
View "pg_catalog.pg_stat_activity"
      Column      |           Type
------------------+--------------------------
 datid            | oid
 datname          | name
 pid              | integer
 usesysid         | oid
 usename          | name
 application_name | text
 client_addr      | inet
 client_hostname  | text
 client_port      | integer
 backend_start    | timestamp with time zone
 xact_start       | timestamp with time zone
 query_start      | timestamp with time zone
 state_change     | timestamp with time zone
 wait_event_type  | text
 wait_event       | text
 state            | text
 backend_xid      | xid
 backend_xmin     | xid
 query            | text
 backend_type     | text
```

Troubleshooting Cont'd

TERMINATING CONNECTIONS Cont'd

- Cancel a backend's current query:

```
select pg_cancel_backend(pid) from pg_stat_activity;
```

- Terminate a backend (superuser):

```
select pg_terminate_backend(pid) from pg_stat_activity;
```

- Terminating A Connection That's Lived Too Long:

```
select pg_terminate_backend(pid) from pg_stat_activity where backend_start < now()-'1 day'::interval;
```

Troubleshooting Cont'd

BLOAT MITIGATION

```
Usage:
  pg_repack [OPTION]... [DBNAME]
Options:
  -a, --all                 repack all databases
  -t, --table=TABLE         repack specific table only
  -I, --parent-table=TABLE  repack specific parent table and its inheritors
  -c, --schema=SCHEMA       repack tables in specific schema only
  -s, --tablespace=TBLSPC   move repacked tables to a new tablespace
  -S, --moveidx             move repacked indexes to TBLSPC too
  -o, --order-by=COLUMNS    order by columns instead of cluster keys
  -n, --no-order            do vacuum full instead of cluster
  -N, --dry-run             print what would have been repacked
  -j, --jobs=NUM            Use this many parallel jobs for each table
  -i, --index=INDEX         move only the specified index
  -x, --only-indexes        move only indexes of the specified table
  -T, --wait-timeout=SECS   timeout to cancel other backends on conflict
  -D, --no-kill-backend     don't kill other backends when timed out
  -Z, --no-analyze          don't analyze at end
  -k, --no-superuser-check  skip superuser checks in client
  -C, --exclude-extension   don't repack tables which belong to specific extension

Generic options:
  -e, --echo                echo queries
  -E, --elevel=LEVEL        set output message level
  --help                    show this help, then exit
  --version                 output version information, then exit
```

Troubleshooting Cont'd

WRAP-AROUND TXID

```
WITH max_age AS (SELECT 2000000000 as max_old_xid,
                             setting AS autovacuum_freeze_max_age
                      FROM pg_catalog.pg_settings
                      WHERE name = 'autovacuum_freeze_max_age' ),
           per_database_stats AS (SELECT datname
                                        , m.max_old_xid::int
                                        , m.autovacuum_freeze_max_age::int
                                        , age(d.datfrozenxid) AS oldest_current_xid
                                    FROM pg_catalog.pg_database d
                                    JOIN max_age m ON (true)
                                    WHERE d.datallowconn )
SELECT max(oldest_current_xid) AS oldest_current_xid
    , max(ROUND(100*(oldest_current_xid/max_old_xid::float))) AS percent_towards_wraparound
    , max(ROUND(100*(oldest_current_xid/autovacuum_freeze_max_age::float))) AS percent_towards_emergency_autovac
FROM per_database_stats;
```

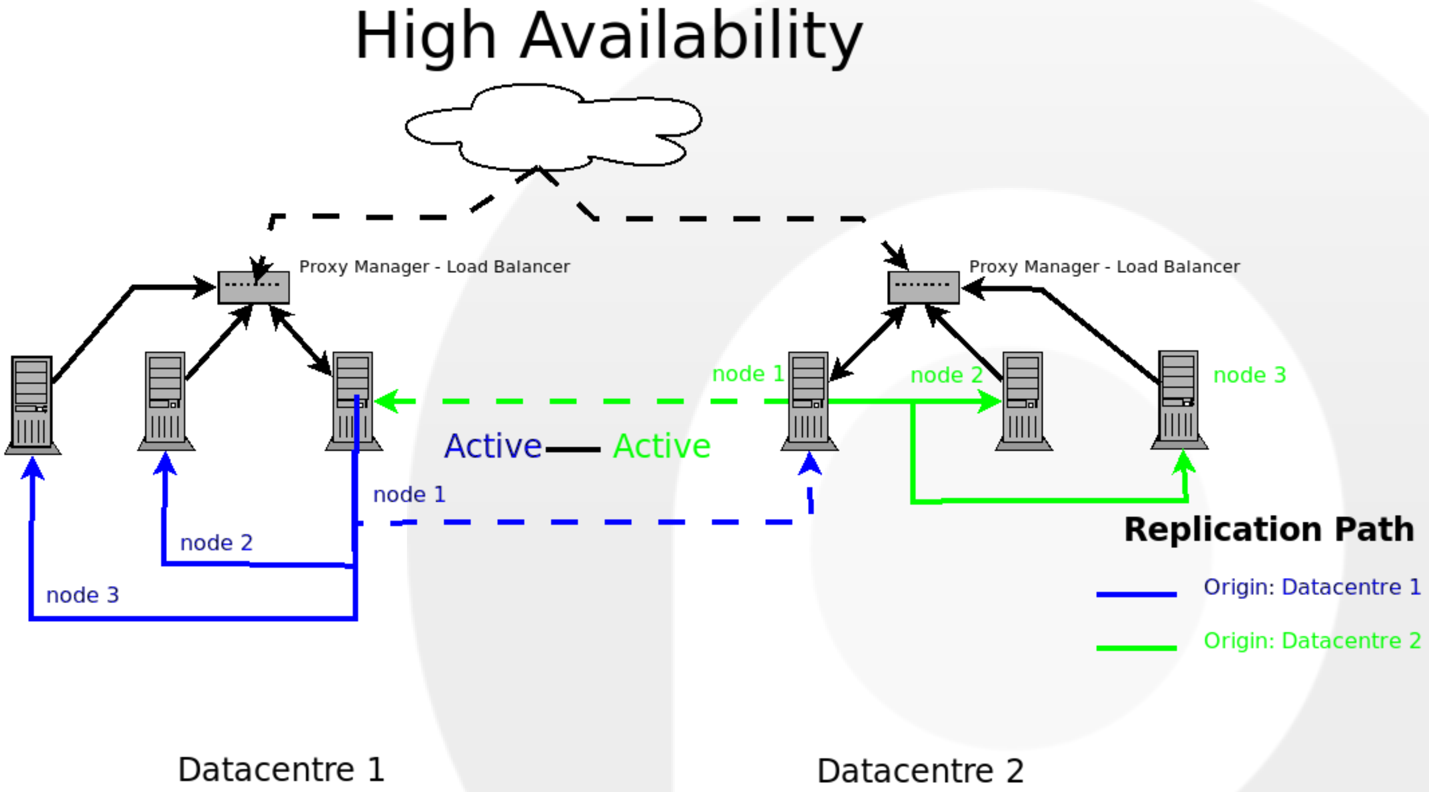
Example

```
- wraparound txid limit = 2 billion transactions    # forced server shutdown
- default autovacuum_freeze_max_age = 200,000,000   # maximum XID age before forced vacuum

 oldest_current_xid | percent_towards_wraparound | percent_towards_emergency_autovac
--------------------+----------------------------+-----------------------------------
        194,623,673 |                         10 |                                97
```

PostgreSQL Operations And Troubleshooting

High Availability

Overview

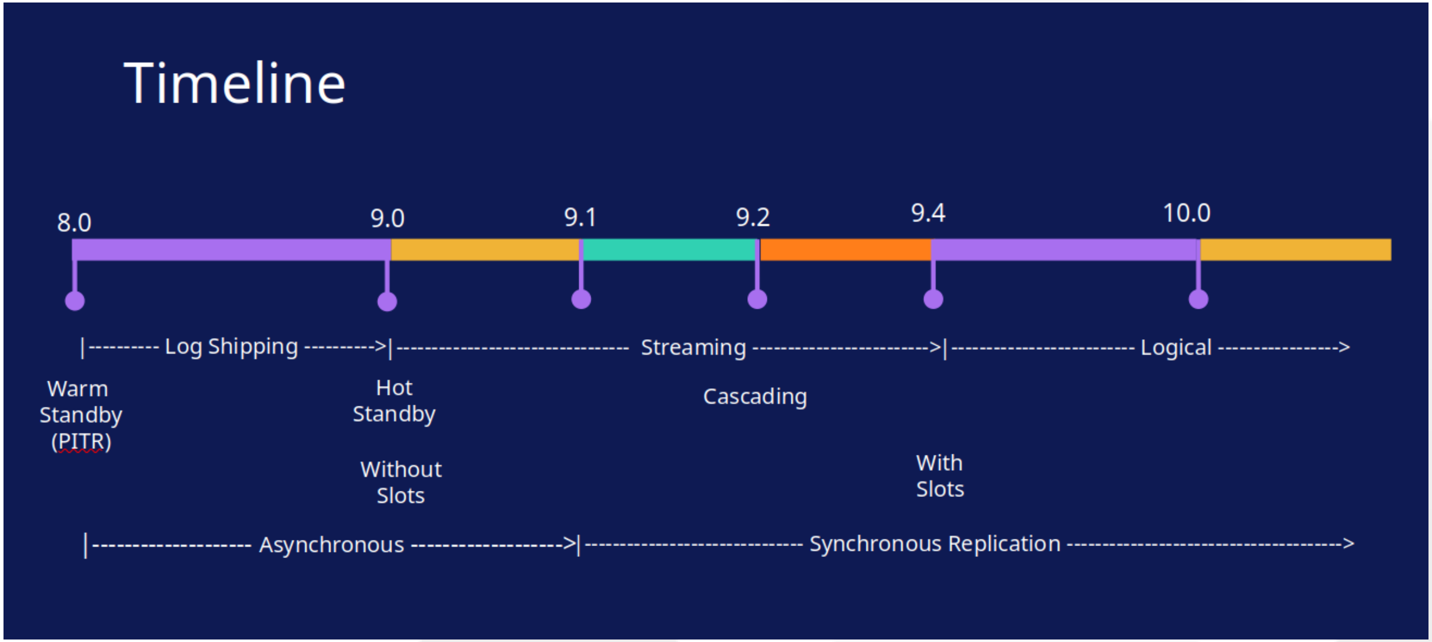

High Availability
Proxy Manager - Load Balancer
Proxy Manager - Load Balancer
node 1
node 2
node 3
Active
Active
node 1
node 2
node 3
Replication Path
Origin: Datacentre 1
Origin: Datacentre 2
Datacentre 1
Datacentre 2

About High Availability (HA)

Topics

- WAL Log Shipping
- BaseBackups
- Replication via Log Shipping
- Replication via Streaming
    - Without slots
    - With slots
    - Variations
        - Log Shipping & Streaming Hybrid
        - Warm Standby vs Hot Standby
- Cascading Replication
- Synchronous Replication
- Logical Replication
- Caveat

Replication History


Timeline
8.0
9.0
9.1
9.2
9.4
10.0
|--------- Log Shipping --------->|---------------------------- Streaming ---------------------->|---------------------- Logical --------------->
Warm Standby (PITR)
Hot Standby
Cascading
Without Slots
With Slots
|------------------ Asynchronous ----------------->|-------------------------- Synchronous Replication --------------------------------->

Configuration Settings

Sending Servers

max_wal_senders	**max_replication_slots**	**wal_keep_size**
wal_sender_timeout	track_commit_timestamp	

Master Server

synchronous_standby_names	vacuum_defer_cleanup_age

Standby Servers

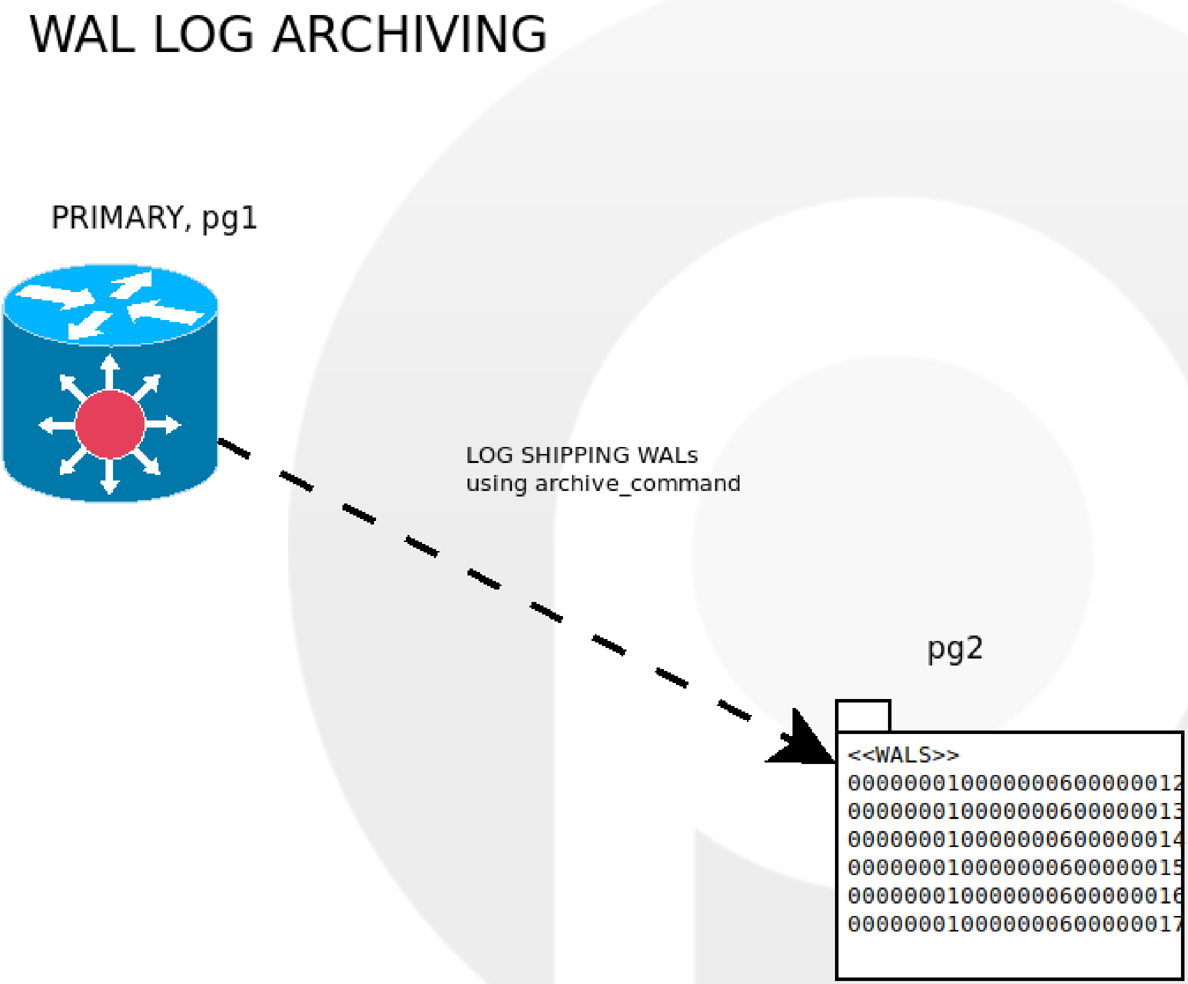
primary_conninfo	**primary_slot_name**	promote_trigger_file
hot_standby	max_standby_archive_delay	max_standby_streaming_delay
wal_receiver_status_interval	hot_standby_feedback	wal_receiver_timeout
wal_retrieve_retry_interval	recovery_min_apply_delay	

Subscribers

max_logical_replication_workers	max_sync_workers_per_subscription

https://www.postgresql.org/docs/current/runtime-config-replication.html

Log Shipping



WAL LOG ARCHIVING

PRIMARY, pg1

LOG SHIPPING WALs
using archive_command

pg2

```
<<WALS>>
000000010000000600000012
000000010000000600000013
000000010000000600000014
000000010000000600000015
000000010000000600000016
000000010000000600000017
```

Log Shipping

Generate public key for postgres on PRIMARY and copy to REPLICA

```
ssh postgres@pg1
```

```
-bash-4.2$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/pgsql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /var/lib/pgsql/.ssh/id_rsa.
Your public key has been saved in /var/lib/pgsql/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7Ik+QQzwmZMrtW5HTtZAAEymAV5arTY8B0z5tZC4JmI postgres@pg1
The key's randomart image is:
...
```

```
# copy public key to hosts pg2 and pg3
ssh-copy-id postgres@pg2
ssh-copy-id postgres@pg3
```

Log Shipping Cont'd

Host pg2: Create WAL directory on REPLICA

```
ssh postgres@pg2
mkdir -p $HOME/WAL
exit
```

Host pg1: setup WAL Log shipping

```
# as root, sudo as postgres
ssh root@pg1
su - postgres
```

```
-- update system as superuser postgres
alter system set archive_mode = on;
alter system set archive_command = 'scp %p pg2:WAL/%f';
alter system set wal_keep_size = 100;
alter system set wal_log_hints = 'on';
```

```
# as root, restart postgres service
systemctl restart postgresql-15
```

Log Shipping Cont'd

Host pg1: Generate WALS

```
-- Login pg1, as postgres superuser and perform the following
dropdb database if exists db01;
create database db01;

\c db01

select *,'hello world'::text as comments
    into table t1
    from (select * from generate_series(1,1e6))t;

-- Flush data files to disk
checkpoint;
-- Force switch to a new write-ahead log file
select pg_walfile_name(pg_switch_wal());
```

Host pg1: remote LOGIN host pg2

```
su - postgres
ssh postgres@pg2 ls -l WAL
```

Basebackups

About Basebackups

```
pg_basebackup --help

Usage:
  pg_basebackup [OPTION]...

Options controlling the output:
  -D, --pgdata=DIRECTORY receive base backup into directory
  -F, --format=p|t       output format (plain (default), tar)
  -r, --max-rate=RATE    maximum transfer rate to transfer data directory
                         (in kB/s, or use suffix "k" or "M")
  -R, --write-recovery-conf
                         write configuration for replication
  -T, --tablespace-mapping=OLDDIR=NEWDIR
                         relocate tablespace in OLDDIR to NEWDIR
      --waldir=WALDIR    location for the write-ahead log directory
  -X, --wal-method=none|fetch|stream
                         include required WAL files with specified method
  -z, --gzip             compress tar output
  -Z, --compress=0-9     compress tar output with given compression level
```

Basebackups Cont'd

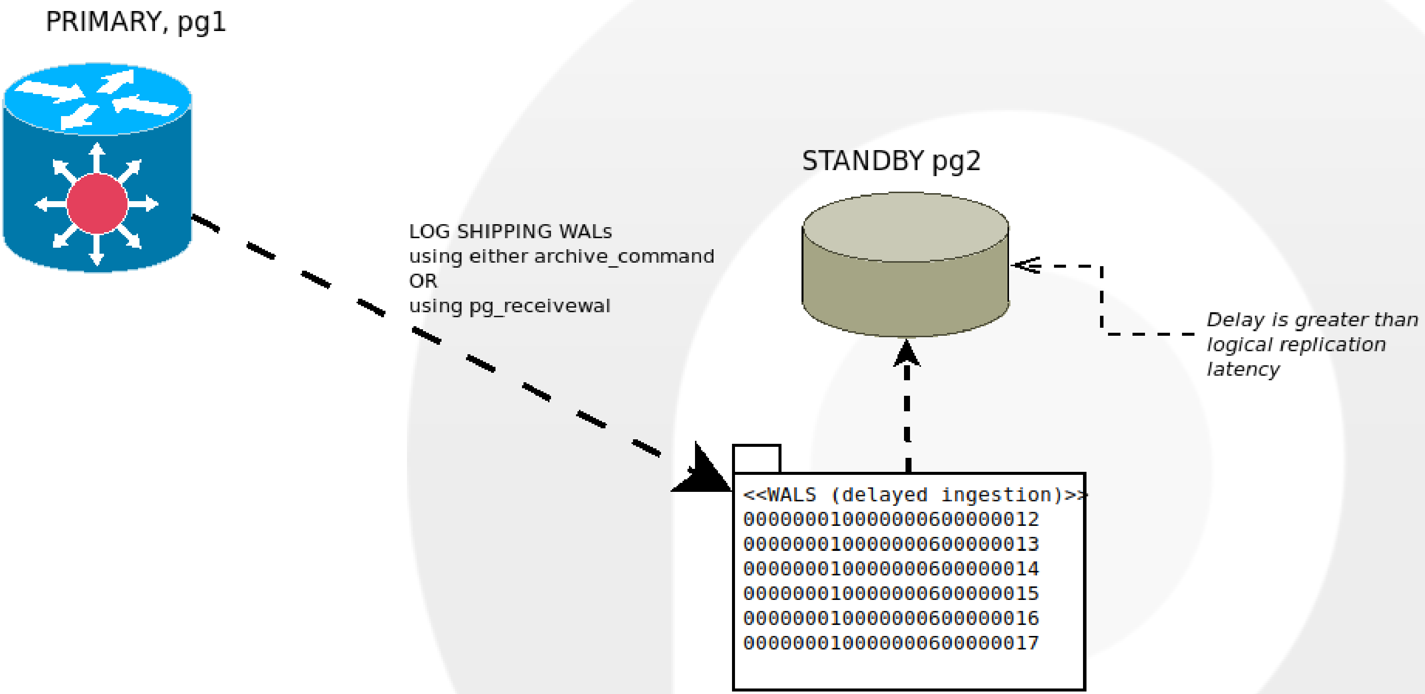
```
General options:
  -c, --checkpoint=fast|spread
                         set fast or spread checkpointing
  -C, --create-slot      create replication slot
  -l, --label=LABEL      set backup label
  -n, --no-clean         do not clean up after errors
  -N, --no-sync          do not wait for changes to be written safely to disk
  -P, --progress         show progress information
  -S, --slot=SLOTNAME    replication slot to use
  -v, --verbose          output verbose messages
  -V, --version          output version information, then exit
      --no-slot          prevent creation of temporary replication slot
      --no-verify-checksums
                         do not verify checksums
  -?, --help             show this help, then exit
```

Basebackups Cont'd

```
Connection options:
  -d, --dbname=CONNSTR   connection string
  -h, --host=HOSTNAME    database server host or socket directory
  -p, --port=PORT        database server port number
  -s, --status-interval=INTERVAL
                         time between status packets sent to server (in seconds)
  -U, --username=NAME    connect as specified database user
  -w, --no-password      never prompt for password
  -W, --password         force password prompt (should happen automatically)
```

Log Shipping Replication

REPLICATION VIA WAL LOG ARCHIVING



PRIMARY, pg1

STANDBY pg2

LOG SHIPPING WALs
using either archive_command
OR
using pg_receivewal

*Delay is greater than
logical replication
latency*

```
<<WALS (delayed ingestion)>>
000000010000000600000012
000000010000000600000013
000000010000000600000014
000000010000000600000015
000000010000000600000016
000000010000000600000017
```

Replication Via Log Shipping

Confirm remote connectivity access

```
# on pg1: check for listening service
[root@pg1 ~]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      297/sshd
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN      1629/postmaster
tcp6       0      0 :::22                   :::*                    LISTEN      297/sshd
tcp6       0      0 :::5432                 :::*                    LISTEN      1629/postmaster
```

```
# on pg2: attempt test login
[root@pg2 ~] psql 'host=pg1 user=postgres password=postgres' -c "select 1 as ping"
 ping
------
    1
```

Check Server Status On Host pg2

- Method 1:

```
systemctl status postgresql-15  # CENTOS
systemctl status postgresql     # Ubuntu
```

- Method 2:

```
ps aux | grep postgres
```

Replication Via Log Shipping Cont'd

Delete any pre-existing data cluster

```
rm -rf /var/lib/pgsql/15/data/*     # CENTOS

pg_dropcluster --stop 12 main       # Ubuntu
```

Perform BaseBackup

```
su - postgres
export PGDATA=/var/lib/pgsql/15/data    # CENTOS
cd $HOME
```

Configure REPLICA

```
echo "
hot_standby = 'on'
recovery_target_timeline = 'latest'
# CENTOS
restore_command='cp /var/lib/pgsql/WAL/%f "%p"'
archive_cleanup_command = '/usr/pgsql-15/bin/pg_archivecleanup /var/lib/pgsql/WAL %r'
# UBUNTU
#restore_command='cp /var/lib/postgresql/WAL/%f "%p"'
#archive_cleanup_command = '/usr/pgsql-15/bin/pg_archivecleanup /var/lib/postgresql/WAL %r'
" >> $PGDATA/postgresql.auto.conf

touch $PGDATA/standby.signal
```

Replication Via Log Shipping Cont'd

As root: REPLICA service start

```
systemctl start postgresql-15

netstat -tlnp

Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      297/sshd
tcp        0      0 0.0.0.0:5432            0.0.0.0:*               LISTEN      867/postmaster
tcp6       0      0 :::22                   :::*                    LISTEN      297/sshd
tcp6       0      0 :::5432                 :::*                    LISTEN      867/postmaster
```

Create table and populate records on host pg1

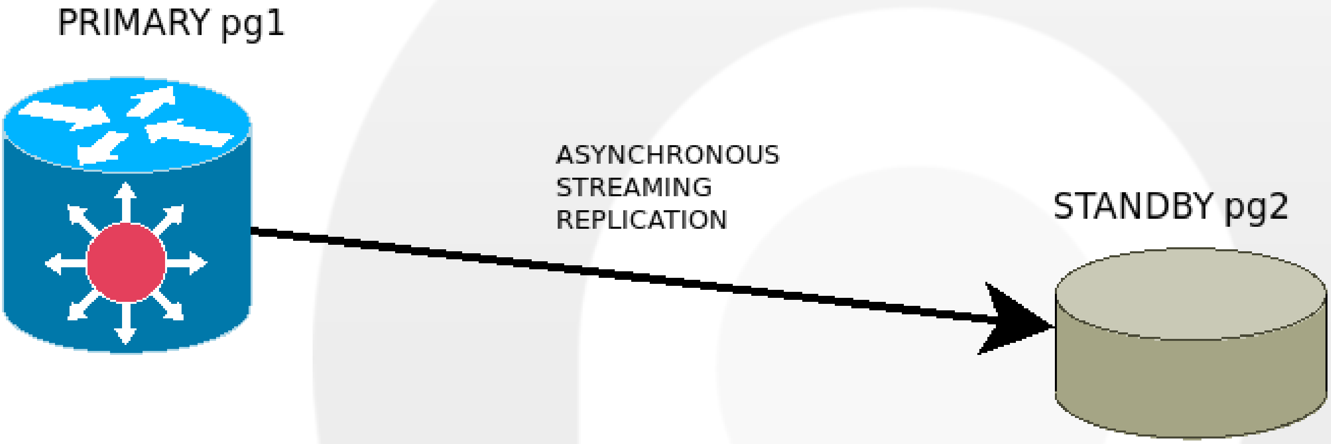
```
SQL="select * into table t2 from generate_series(1,1e6)"
psql 'host=pg1 dbname=db01 user=postgres password=postgres'<<<$SQL
psql 'host=pg1 dbname=db01 user=postgres password=postgres' -c 'checkpoint;select pg_switch_wal()'
```

Confirm replication on host pg2

```
[root@pg2 ~] psql 'host=pg2 dbname=db01 user=postgres password=postgres' -c '\dt+'
                   List of relations
 Schema | Name | Type  |  Owner   | Size  | Description
--------+------+-------+----------+-------+-------------
 public | t1   | table | postgres | 50 MB |
 public | t2   | table | postgres | 35 MB |
```

Streaming Replication Without Slots

Async Streaming Replication without slots



Replication Via Streaming, Without Slots

Execute on PRIMARY, host pg1

```
-- Add a replicating ROLE
create role replicant with login replication password 'mypassword';
--
-- enable streaming replication
alter system set wal_level = 'replica';
```

```
# as root; restart the service on PRIMARY pg1
systemctl restart postgresql-15
```

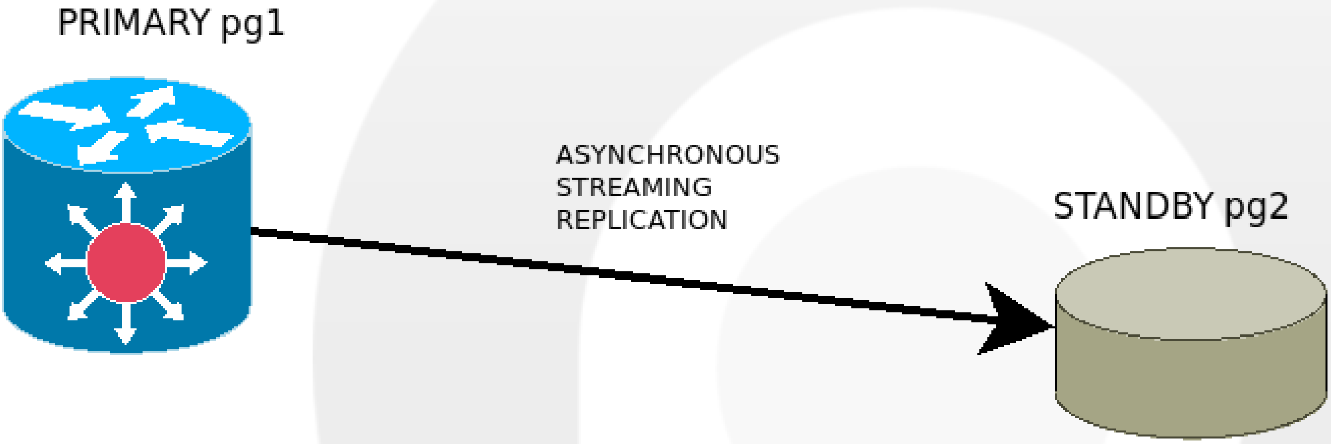
Execute on REPLICA, host pg2

```
-- point REPLICA to PRIMARY
alter system set primary_conninfo = 'host=pg1 user=replicant password=mypassword';
```

```
# as root; restart the service on REPLICA pg2
systemctl restart postgresql-15
```

Streaming Replication With Slots

Async Streaming Replication with slots



Replication Via Streaming, With Slots

Method 1: update existing configuration

Execute on PRIMARY, host pg1

```
select pg_create_physical_replication_slot('pg2');
```

Execute on REPLICA, host pg2

```
-- point REPLICA to PRIMARY using slot
alter system set primary_slot_name = 'pg2'

# as root; restart the service on REPLICA pg2
systemctl restart postgresql-15
```

Execute on PRIMARY, host pg1

```
-- confirm replication slot is active
postgres=# select slot_name, slot_type, active, active_pid from pg_replication_slots;
 slot_name | slot_type | active | active_pid
-----------+-----------+--------+------------
 pg2       | physical  | t      |       2313
```

Replication Via Streaming, With Slots

Method 2: generate new basebackup

As root: execute the following on REPLICA pg2

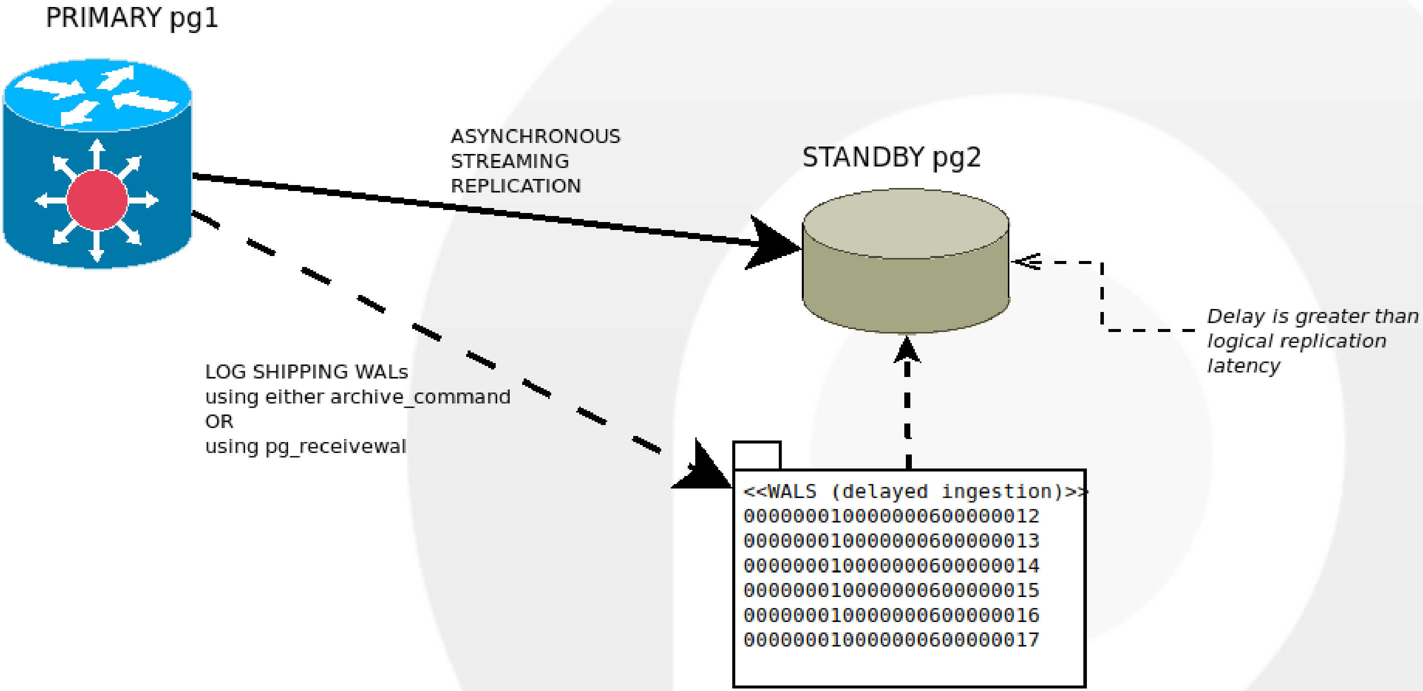
- Stop the service
- Delete the data cluster
- Create a new basebackup using slots, it's assumed no physical slot has already been created on PRIMARY

```
rm -rf /var/lib/pgsql/15/data/
```

```
# PGDATA environment variable assumes CENTOS
systemctl stop postgresql-15
su - postgres
rm -rf /var/lib/pgsql/15/data

/usr/pgsql-15/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432' \
                                --wal-method=stream \
                                -l basebackup \
                                -D /var/lib/pgsql/15/data \
                                -R -P -v \
                                --create-slot \
                                --slot=pg2


touch $PGDATA/standby.signal
```

PostgreSQL Replication, Variations

Async Streaming Replication with slots and Log Shipping Hybrid



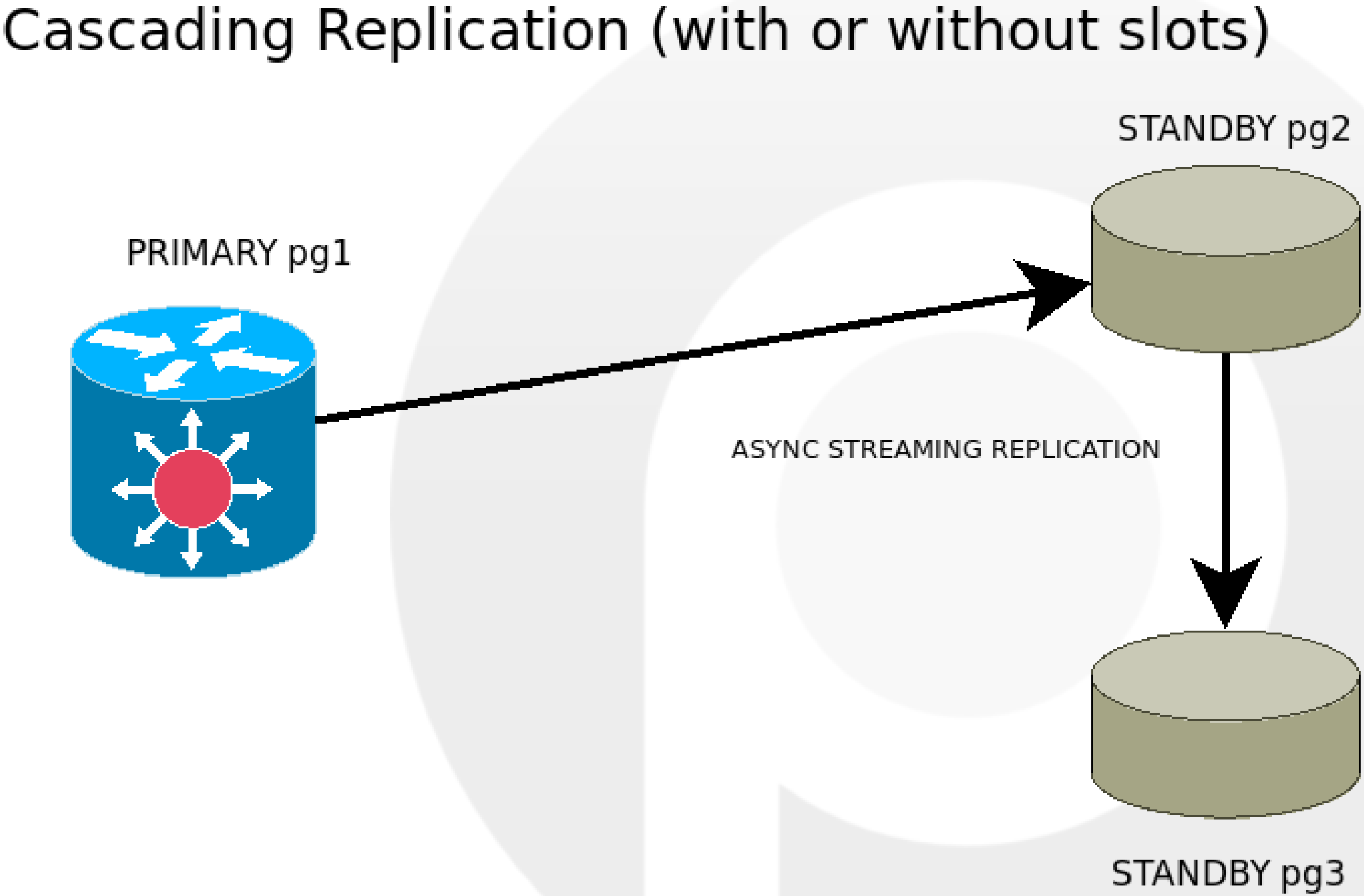
```
000000010000000600000012
000000010000000600000013
000000010000000600000014
000000010000000600000015
000000010000000600000016
000000010000000600000017
```

PostgreSQL Replication, Variations

Standby Mode: *(requires a server restart)*

```
-- Warm Standby (No Read)
alter system set hot_standby='off';

-- Hot Standby (DEFAULT: Read-Only)
alter system set hot_standby='on';
```

Cascading Replication


Cascading Replication (with or without slots)
PRIMARY pg1
STANDBY pg2
ASYNC STREAMING REPLICATION
STANDBY pg3

Cascading Replication, Cont'd

Perform BaseBackup

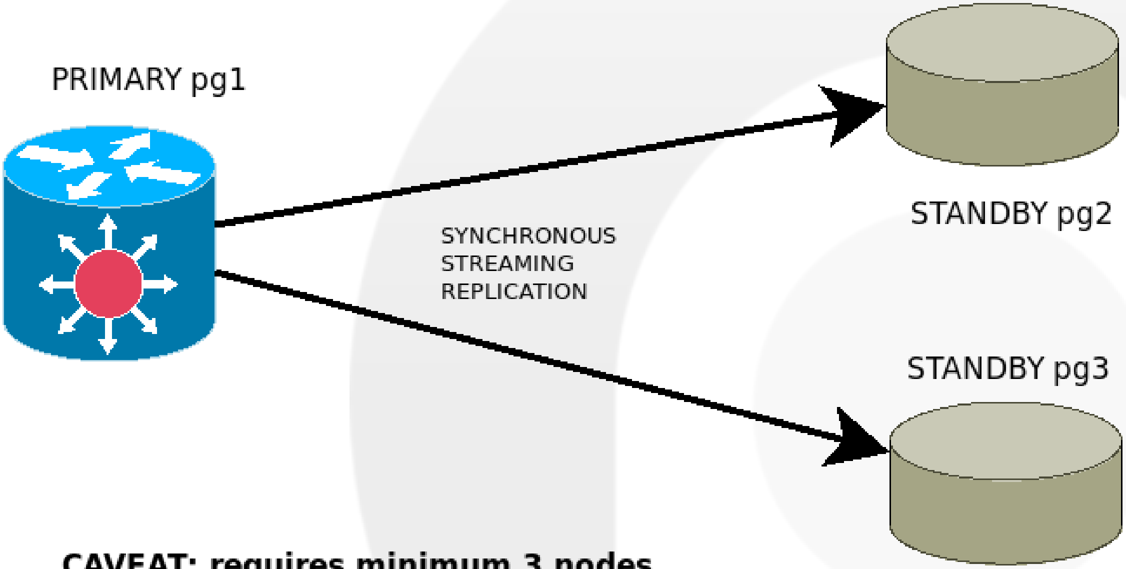
```
systemctl stop postgresql-15
su - postgres
rm -rf /var/lib/pgsql/15/data

# host is changed from pg1 to pg2
/usr/bin/pg_basebackup -d 'host=pg2 user=postgres password=postgres port=5432' \
                       --wal-method=stream \
                       -l basebackup \
                       -D /var/lib/pgsql/15/data \
                       -R -P -v \
                       --create-slot \
                       --slot=pg3
```

Configure CASCADED REPLICA, valid for pg ver 12+

```
touch $PGDATA/standby.signal
```

Synchronous Replication

Synchronous Streaming Replication (with slots of course!)



CAVEAT: requires minimum 3 nodes

Asynchronous VS Synchronous Replication:

- **Asynchronous** replication is non-blocking and returns as soon as the transaction is committed on the PRIMARY.
- **Synchronous** replication commits a transaction only when the operation completes on the slave(s).

```
#synchronous_standby_names = '' # SYNC ORDER
                                #   num_sync is the number of synchronous standbys
                                #   that transactions need to wait for replies from
                                #
                                # '*' OR 'all'
                                # [FIRST] num_sync ( standby_name [, ...] )
                                # ANY num_sync ( standby_name [, ...] )
                                # standby_name [, ...]
                                #
```

```
#synchronous_commit = on        # synchronization level;
                                #   off            # synchronous replication is ignored
                                #
                                #   local          # guaranteed data flush only on the primary node
                                #
                                #   remote_write   # commit waits for confirmation from standby
                                #                       writing the record (not yet flushed)
                                #
                                #   remote_apply   # standby replies when the commit record is replayed
                                #
                                #   on             # waits until data is flushed
                                #                       to the transaction log on all hosts
```

```
cluster_name (string)           # Sets a name that identifies this database cluster.
```

Synchronous Replication, Cont'd

HOST=PG1

```
-- ensure slots are present on the PRIMARY
select pg_create_physical_replication_slot('pg2');
select pg_create_physical_replication_slot('pg3');
```

HOST=PG2

```
###########
# as root:
#
systemctl stop postgresql-15
rm -rf /var/lib/pgsql/15/data
#
###########
# as postgres
#
/usr/pgsql-15/bin/pg_basebackup \
    -d 'host=pg1 user=replicant password=mypassword port=5432 application_name=pg2' \
    --wal-method=stream \
    -l basebackup \
    -D /var/lib/pgsql/15/data \
    -R -P -v \
    --slot=pg2

touch $PGDATA/standby.signal

echo "cluster_name = pg2" >> $PGDATA/postgresql.auto.conf
#
###########
# as root
systemctl start postgresql-15
```

Synchronous Replication, Cont'd

HOST=PG3

```
###########
# as root:
#
systemctl stop postgresql-15
rm -rf /var/lib/pgsql/15/data
#
###########
# as postgres
#
/usr/pgsql-15/bin/pg_basebackup \
    -d 'host=pg1 user=replicant password=mypassword port=5432 application_name=pg3' \
    --wal-method=stream \
    -l basebackup \
    -D /var/lib/pgsql/15/data \
    -R -P -v \
    --slot=pg3

touch $PGDATA/standby.signal

echo "cluster_name = pg3" >> $PGDATA/postgresql.auto.conf
#
###########
# as root
systemctl start postgresql-15
```

Synchronous Replication, Cont'd

HOST=PG1

```
# default = on, requires restart
show synchronous_commit;

# requires reload
alter system set synchronous_standby_names='pg2,pg3';
select pg_reload_conf();

# validate
select application_name,state,sync_state from pg_stat_replication order by 1;

 application_name |   state   | sync_state
------------------+-----------+------------
 pg2              | streaming | sync
 pg3              | streaming | potential
```

Example Alternative Replication Wait Modes

```
alter system set synchronous_standby_names='FIRST 1 (pg3,pg2)';
alter system set synchronous_standby_names='FIRST 2 (pg3,pg2)';
alter system set synchronous_standby_names='ANY 2 (pg3,pg2)';
```

Replication Caveat

- Basebackup behaviour: log shipping versus streaming (with and without slots)
- Monitor:
  - PRIMARY:

    ```
    select * from pg_stat_replication;
    select * from pg_replication_slots;
    select * from pg_get_replication_slots();
    ```

  - REPLICA: postgres logs

    ```
    cat <postgres log> | grep -E 'ERROR|FATAL'
    ```

- Slot administration:
  - CLI:

    ```
    pg_receivewal
    ```

  - FUNCTIONS:

    ```
    pg_drop_replication_slot
    pg_copy_physical_replication_slot
    pg_create_physical_replication_slot
    pg_replication_slot_advance
    ```

Logical Replication

About

A method of replicating data objects and their changes, based upon their replication identity (usually a primary key). While streaming replication uses exact block addresses, logical replication is byte-by-byte.

Logical replication uses a publish and subscribe model with one or more subscribers subscribing to one or more publications on a publisher node. Subscribers pull data from the publications they subscribe to and may subsequently re-publish data to allow cascading replication or more complex configurations.

Logical Replication

Capabilities

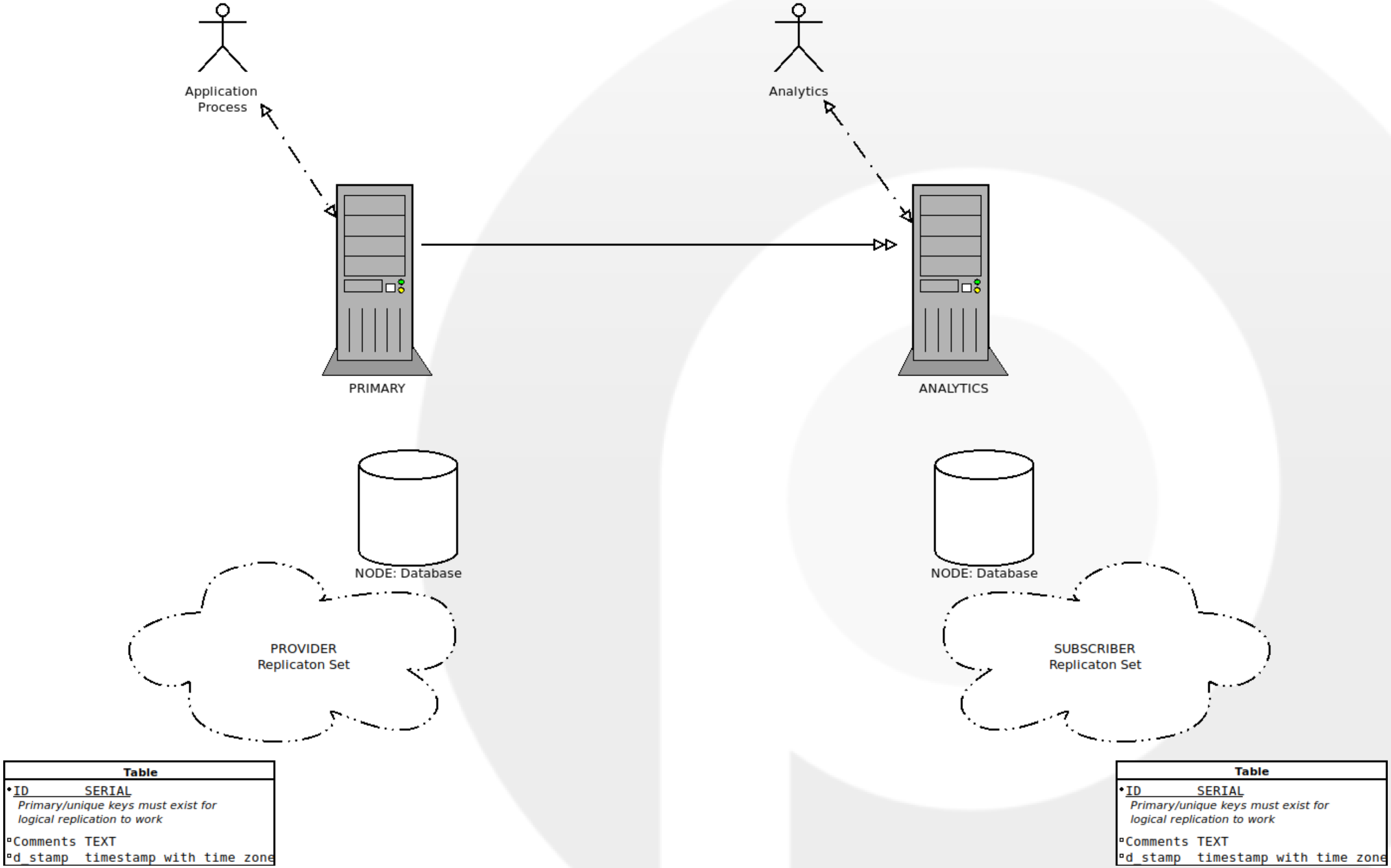
- UPGRADE: Upgrade PostgreSQL from 9.4 to 9.5, without downtime
- SCALE OUT: Copy all or a selection of database tables to other nodes in a cluster
- AGGREGATE: Accumulate changes from sharded database servers into a Data Warehouse
- INTEGRATE: Feed database changes in real-time to other systems
- PROTECT: Provide backup or high availability for clusters, replacing earlier technologies

Method

Logically replicating a table starts with snapshot of the data from the **publisher** database and copying that to the **subscriber** database.

Changes on the publisher are sent to the subscriber real-time and is applied in the same order.

Logical Replication


Application Process
Analytics
PRIMARY
ANALYTICS
NODE: Database
NODE: Database
PROVIDER Replicaton Set
SUBSCRIBER Replicaton Set
Table
ID SERIAL
Primary/unique keys must exist for logical replication to work
Comments TEXT
d_stamp timestamp with time zone
Table
ID SERIAL
Primary/unique keys must exist for logical replication to work
Comments TEXT
d_stamp timestamp with time zone

Logical Replication

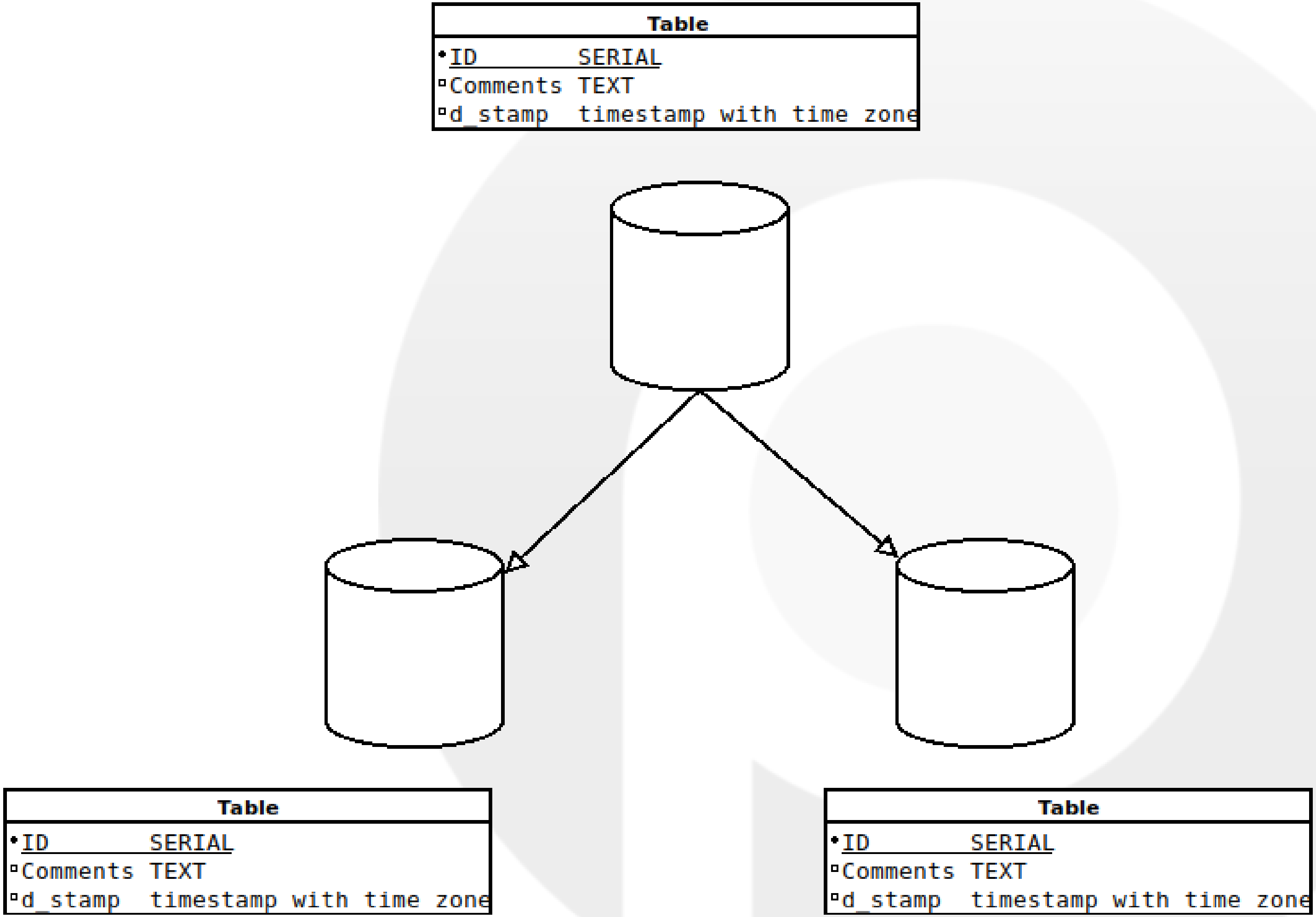


Table	
• ID	SERIAL
▫ Comments	TEXT
▫ d_stamp	timestamp with time zone

Table	
• ID	SERIAL
▫ Comments	TEXT
▫ d_stamp	timestamp with time zone

Table	
• ID	SERIAL
▫ Comments	TEXT
▫ d_stamp	timestamp with time zone

Logical Replication

Method

- create the database(s)
- create two schemas
- create and populate table(s)
- create the logical node(s)
- create replication set(s)
- subsribe to provider(s)
- peform DML

```
CREATE PUBLICATION name
    [ FOR TABLE [ ONLY ] table_name [ * ] [, ...]
      | FOR ALL TABLES ]
    [ WITH ( publication_parameter [= value] [, ... ] ) ]

CREATE SUBSCRIPTION subscription_name
    CONNECTION 'conninfo'
    PUBLICATION publication_name [, ...]
    [ WITH ( subscription_parameter [= value] [, ... ] ) ]
```

Logical Replication, **Example 1**

-- host:pg1

```
drop database if exists db01;
create database db01;
\c db01
alter role postgres with password 'postgres';
create schema a;
create table a.t1(i serial primary key,comments text);
create publication provider1 FOR TABLE a.t1;
```

-- host:pg3

```
create database db01;
alter role postgres with password 'postgres';
\c db01
create schema a;
create table a.t1(i serial primary key,comments text);

create subscription mysub
         connection 'host=pg1 port=5432 user=postgres dbname=db01 password=postgres'
        publication provider1
               with (enabled = true);
```

Logical Replication, **Example 2**

```
export PGUSER=postgres PGPASSWORD=postgres
```

```
createdb -h pg1 db02
createdb -h pg3 db02

/usr/pgsql-15/bin/pgbench -h pg1 -i db02

psql 'host=pg1 dbname=db02' -c 'alter table pgbench_history add primary key(tid,bid,aid,delta,mtime);'

pg_dump -s -h pg1 db02 | psql -h pg3 db02
```

pg1: create PUBLICATION, execute on host pg1, database db02

```
set search_path=public;
create publication publication1 for all tables;
```

pg3: create SUBSCRIPTION, execute on host pg3, database db02

```
create subscription subscript_set1
    connection 'host=pg1 dbname=db02 user=postgres password=postgres'
    publication publication1
        with (copy_data = true, create_slot = true, enabled = true, slot_name = myslot1);
```

Execute benchmarking on host pg1, database db02

```
/usr/pgsql-15/bin/pgbench -h pg1 -c 4 -j 2 -T 100 -b tpcb-like db02
```

Logical Replication

Caveat

- DDLs not supported
- No Replication Queue Flush (Failover is problematic)
- No Cascaded Replication
- One unique index/constraint/pk per table
- Permissions (remote access by subscriber)
- Primary key must exist
- Sequences
- Triggers
- Truncate command is not propagated
- Unlogged/temporary tables not supported

Connection Pooling

About Connection Pooling

A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database.

Reference: https://en.wikipedia.org/wiki/Connection_pool

About pgbouncer

A lightweight connection pooler for PostgreSQL

Reference: https://www.pgbouncer.org/

pgbouncer: Features

- Supports pooling policies:
  - Session pooling: Most polite method. When a client connects, a server connection will be assigned to it for the whole duration it stays connected. The server connection is put back into pool when the connection closes.
  - Transaction pooling: A server connection is assigned to a client only during a transaction. When PgBouncer notices that the transaction is over, the connection is put back into the pool.
  - Statement pooling: The connection is closed and is put back into the pool upon the completion of a SQL statement.
- Low memory requirements, 2 kB per connection
- Can alias connection parameters and target hosts and databases. For example, the destination database can reside on a different host.
- Supports online reconfiguration administration capabilities
- Supports online restart/upgrade without dropping client connections
- Supports multiple authentication protocols and encryption methods

pgbouncer: Installation

```
dnf install -y pgbouncer    # CENTOS 8
apt install -y pgbouncer    # Ubuntu
#
systemctl start pgbouncer
```

```
[root@pg1 ~]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      299/sshd
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN      1179/postmaster
tcp        0      0 127.0.0.1:6432          0.0.0.0:*               LISTEN      1192/pgbouncer
tcp6       0      0 :::22                   :::*                    LISTEN      299/sshd
tcp6       0      0 ::1:5432                :::*                    LISTEN      1179/postmaster
tcp6       0      0 ::1:6432                :::*                    LISTEN      1192/pgbouncer
```

ATTENTION:

- Centos/Redhat: pgbouncer process owned by **pgbouncer**
- Ubuntu/Debian: pgbouncer process owned by **postgres**

pgbouncer: Configuration

```
    mv /etc/pgbouncer/pgbouncer.ini /etc/pgbouncer/pgbouncer.ini_backup
```

Minimal Configuration

```
echo "
[databases]
* = host=127.0.0.1 port=5432

[users]

[pgbouncer]
logfile = /var/log/pgbouncer/pgbouncer.log
pidfile = /var/run/pgbouncer/pgbouncer.pid
listen_addr = 0.0.0.0
listen_port = 6432

auth_type = plain
auth_file = /etc/pgbouncer/userlist.txt
" > /etc/pgbouncer/pgbouncer.ini
```

Administrative LOGIN (Redhat/Centos)

```
[root@pg1 pgbouncer]# su - pgbouncer -c "psql 'host=/tmp port=6432' "
psql (12.6, server 1.15.0/bouncer)
Type "help" for help.
```

pgbouncer: Configuration Cont'd

```
pgbouncer=# show help;
NOTICE:  Console usage
DETAIL:
        SHOW HELP|CONFIG|DATABASES|POOLS|CLIENTS|SERVERS|USERS|VERSION
        SHOW FDS|SOCKETS|ACTIVE_SOCKETS|LISTS|MEM
        SHOW DNS_HOSTS|DNS_ZONES
        SHOW STATS|STATS_TOTALS|STATS_AVERAGES|TOTALS
        SET key = arg
        RELOAD
        PAUSE [<db>]
        RESUME [<db>]
        DISABLE <db>
        ENABLE <db>
        RECONNECT [<db>]
        KILL <db>
        SUSPEND
        SHUTDOWN
```

pgbouncer: Configuration Cont'd

ATTENTION:(there's more than one way to setup user login authentication)

- authentication settings: any, trust, plain, crypt, md5, cert, hba, pam, auth_query
- Ubuntu is easy to configure because pgbouncer is owned by "postgres" while CENTOS uses Unix user "pgbouncer" therefore pg_hba.conf must be edited for localhost connections ex: use METHOD "md5".
- always set ROLE passwords

pgbouncer: Configuration Cont'd

EXAMPLE 1: pgbouncer authenticates using "userlist.txt"

```
;; RECALL
[databases]
* = host=127.0.0.1
auth_type = plain
auth_file = /etc/pgbouncer/userlist.txt
```

```
echo '
"postgres" "postgres"
' > /etc/pgbouncer/userlist.txt
```

```
[root@pg1 ~]# psql 'host=127.0.0.1 port=6432 user=postgres password=postgres'
psql (12.6)
Type "help" for help.

postgres=#
```

pgbouncer: Configuration Cont'd

EXAMPLE 2: pgbouncer host based authentication

/etc/pgbouncer/pgbouncer.ini

```
[databases]
* = host=127.0.0.1
auth_type = hba
auth_file = /etc/pgbouncer/userlist.txt
auth_hba_file =/etc/pgbouncer/pg_hba.conf
```

/etc/pgbouncer/userlist.txt

```
#copy passwords from pg_shadow
echo '
;"postgres" "mypassword"
"postgres" "md50da9ad9e72f4a215ede570b27a736c4a"
"usr1" "md5b1d2240d1ca66a849768e63daae33e05"

; pgbouncer admin login: does not require a pg ROLE
"stats" "mypassword"
' > /etc/pgbouncer/userlist.txt
```

pgbouncer: Administration

CENTOS & Ubuntu

```
# execute as root
systemctl start pgbouncer
```

CENTOS:

```
systemctl start pgbouncer
su - pgbouncer -c "psql -U pgbouncer -p 6432"
```

Ubuntu:

```
systemctl start pgbouncer
su - postgres -c "psql -U pgbouncer -p 6432"
```

PostgreSQL Operations And Troubleshooting

Backups, Redundancy And Availability

Backups, Redundancy And Availability

PostgreSQL Command Line Utilities

Best Practices

Operations and Definitions

Failover/Switchover

Point In Time Recovery (PITR) ... pg_rewind

PostgreSQL Command Line Utilities

- pg_dumpall
- pg_dump
- pg_restore
- pg_basebackup
- pg_upgrade
- psql

pg_dumpall

- Can dump/backup all the databases of a cluster into a script file (uses text format).

```
$ pg_dumpall > /tmp/all_databases_dump.sql
```

- Dump **globals** using pg_dumpall.

```
$ pg_dumpall -g > /tmp/globals.sql
```

- Use psql to restore the backup taken using pg_dumpall.

```
$ psql -f /tmp/all_databases_dump.sql
```

pg_dump

- Use pg_dump to backup a table (with data)
    - using custom format

      ```
      $ pg_dump -Fc -t public.pgbench_history -d percona -f /tmp/pgbench_history
      ```

    - using plain text format

      ```
      $ pg_dump -t public.pgbench_branches -d percona -f /tmp/pgbench_branches
      ```

    - When you open both the dump files, you see that the dumpfile (/tmp/pgbench_branches) generated using plain text format is human-readable.

- Use pg_dump to backup a database

```
-- Works in both custom and plain text format
$ pg_dump percona -f /tmp/percona.sql
-- Use -s to take DDL only backup of Percona database
$ pg_dump -s percona -f /tmp/percona_ddl.sql
```

- You cannot take multiple databases backup using pg_dump

pg_restore

- When do we use **pg_restore** to restore a dumpfile ?
  - When a dumpfile is generated in **custom format** using **pg_dump**, we use **pg_restore**.
- When do we use **psql** to restore a dumpfile ?
  - When a dumpfile is generated in **plain text format** using **pg_dump**, we use **psql**.
- Use pg_restore to restore the table dump (/tmp/pgbench_history) taken using custom format

```
-- Create a test database
$ psql -c "CREATE DATABASE testdb"
-- Use pg_restore to restore the dump
$ pg_restore -t pgbench_history -d testdb /tmp/pgbench_history
```

pg_basebackup

- Used for taking a Consistent File System level backup that can be used for Point-in-time-recovery or setting up Slaves.
- At this point of time, pg_basebackup cannot run in parallel.
- pg_basebackup takes a snapshot of the entire Data Directory of PostgreSQL along with the WALs generated during the backup.
- pg_basebackup can be taken remotely from another Instance costing a network bandwidth at a controlled rate.
- You cannot take Incremental and Differential Backups using pg_basebackup.
- pg_basebackup uses a Replication Protocol. Which means, you have to allow replication connections from the user,host combination in the pg_hba.conf to run a pg_basebackup.

pg_basebackup Cont'd

```
Usage:
  pg_basebackup [OPTION]...

Options controlling the output:
  -D, --pgdata=DIRECTORY receive base backup into directory
  -F, --format=p|t       output format (plain (default), tar)
  -r, --max-rate=RATE    maximum transfer rate to transfer data directory
                         (in kB/s, or use suffix "k" or "M")
  -R, --write-recovery-conf
                         write configuration for replication
  -T, --tablespace-mapping=OLDDIR=NEWDIR
                         relocate tablespace in OLDDIR to NEWDIR
      --waldir=WALDIR    location for the write-ahead log directory
  -X, --wal-method=none|fetch|stream
                         include required WAL files with specified method
  -z, --gzip             compress tar output
  -Z, --compress=0-9     compress tar output with given compression level

General options:
  -c, --checkpoint=fast|spread
                         set fast or spread checkpointing
  -C, --create-slot      create replication slot
  -l, --label=LABEL      set backup label
  -n, --no-clean         do not clean up after errors
  -N, --no-sync          do not wait for changes to be written safely to disk
  -P, --progress         show progress information
  -S, --slot=SLOTNAME    replication slot to use
  -v, --verbose          output verbose messages
  -V, --version          output version information, then exit
      --no-slot          prevent creation of temporary replication slot
      --no-verify-checksums
                         do not verify checksums
  -?, --help             show this help, then exit
```

pg_basebackup Cont'd

- Run the following command that helps you take a backup to /tmp/first_backup directory.

```
$ pg_basebackup -U postgres -p 5432 -h 127.0.0.1 -D /tmp/first_backup \
-Ft -z -Xs -P -R -l backup_label
```

- Once finished with the backup, see what is stored in the directory specified for backup.

```
$ ls -l /tmp/first_backup
total 8060
-rw-------. 1 postgres postgres 8231974 Dec 14 00:22 base.tar.gz
-rw-------. 1 postgres postgres   17647 Dec 14 00:22 pg_wal.tar.gz
```

- Now, extract the backup and see what is recorded in the file : backup_label

```
$ mkdir -p /tmp/data
$ tar -xzf /tmp/first_backup/base.tar.gz -C /tmp/data
$ cat /tmp/data/backup_label
START WAL LOCATION: 0/8000028 (file 000000010000000000000008)
CHECKPOINT LOCATION: 0/8000060
BACKUP METHOD: streamed
BACKUP FROM: master
START TIME: 2018-12-14 00:21:58 UTC
LABEL: backup_label
START TIMELINE: 1
```

pg_upgrade

- pg_upgrade allows data stored in PostgreSQL data files to be upgraded to a later PostgreSQL major version without the data dump/restore typically required for major version upgrades.
- Major PostgreSQL releases regularly add new features that often change the layout of the system tables, but the internal data storage format rarely changes. The community will attempt to avoid such situations.
- pg_upgrade supports upgrades from 9.2.X and later to the current major release of PostgreSQL, including snapshot and beta releases.

pg_upgrade Cont'd

```
Usage:
  pg_upgrade [OPTION]...

Options:
  -b, --old-bindir=BINDIR       old cluster executable directory
  -B, --new-bindir=BINDIR       new cluster executable directory (default
                                same directory as pg_upgrade)
  -c, --check                   check clusters only, don't change any data
  -d, --old-datadir=DATADIR     old cluster data directory
  -D, --new-datadir=DATADIR     new cluster data directory
  -j, --jobs=NUM                number of simultaneous processes or threads to use
  -k, --link                    link instead of copying files to new cluster
  -N, --no-sync                 do not wait for changes to be written safely to disk
  -o, --old-options=OPTIONS     old cluster options to pass to the server
  -O, --new-options=OPTIONS     new cluster options to pass to the server
  -p, --old-port=PORT           old cluster port number (default 50432)
  -P, --new-port=PORT           new cluster port number (default 50432)
  -r, --retain                  retain SQL and log files after success
  -s, --socketdir=DIR           socket directory to use (default current dir.)
  -U, --username=NAME           cluster superuser (default "postgres")
  -v, --verbose                 enable verbose internal logging
  -V, --version                 display version information, then exit
  --clone                       clone instead of copying files to new cluster
  -?, --help                    show this help, then exit

Before running pg_upgrade you must:
  create a new database cluster (using the new version of initdb)
  shutdown the postmaster servicing the old cluster
  shutdown the postmaster servicing the new cluster


EXAMPLE:
  pg_upgrade -d oldCluster/data -D newCluster/data -b oldCluster/bin -B newCluster/bin
```

Best Practices

- dump & restore
    - try to always use superuser (less hassle)
    - copy&paste/scripting
    - version redundancy (copies)
    - in regards to dumps
        - compressed vs uncompressed (speed,size)
        - sections vs complete
        - assigning role ownership & granting permissions
        - recreating
            - relations
            - database
    - in regards to restorations
        - compression required
        - multi-threaded (speed)
        - incrementally controlled restorations (manifests)

Best Practices Cont'd

- dumping and archiving
  - pg_dumpall -g
  - pg_dump
    - cycling across datacluster
    - diffs
- complex restorations
  - via schema or sections
  - manifests
  - advantages of piping & paging
    - building out gradually
    - double check before committing
- dump & restore across different versions
- examples
  - ex 1: pg_dump -Fp | psql
  - ex 2: pg_dump -Fc | pg_restore
  - ex 3: pg_dump -Fc | pg_restore -l > manifest.ini
  - ex 4: pg_dump -Fc | pg_restore -j 3 -L manifest.ini

Operations And Definitions

- **Failover**: the ability to seamlessly and automatically switch from a failed PRIMARY node to a reliable STANDBY by means of a system promotion
- **Switchover**: similar to a Failover except that the PRIMARY has not failed, rather it is a controlled promotion. Mean while the PRIMARY can be brought down under control and optionally reprovisioned as a new REPLICA node.
- **Failback**: A controlled Switchover returning read-write duties to a REPLICA that was formerly the PRIMARY node in the cluster.

Service Failover/SwitchOver

- **CENTOS**: pg_ctl -D [PGDATA] (action)
  - pg_ctl -D $PGDATA promote
- **Debian/Ubuntu**: pg_ctlcluster (version) (cluster) (action)
  - pg_ctlcluster 12 main promote

Caveat

- **pg_ctl**: run as postgres
- **pg_ctlcluster**: run as either postgres or root (but it should be root)

Point-In-Time-Recovery (PITR)

About PITR

- Archive Recovery Settings

```
restore_command
archive_cleanup_command
recovery_end_command
```

- Recovery Target Settings

```
recovery_target
recovery_target_name
recovery_target_time
recovery_target_xid
recovery_target_lsn
recovery_target_inclusive
recovery_target_timeline
recovery_target_action
```

Point-In-Time-Recovery (PITR) Cont'd

pgbackrest

Install packages

```
#
# CENTOS
#
dnf update -y
dnf install -y epel-release
dnf install -y pgbackrest
updatedb

mkdir -p /var/log/pgbackrest/
chown postgres.postgres /var/log/pgbackrest/
```

Edit pgbackrest.conf

```
#
# vi /etc/pgbackrest.conf
#
[global]
repo1-path=/var/lib/pgsql/15/backups

[main]
pg1-path=/var/lib/pgsql/15/data
```

Point-In-Time-Recovery (PITR) Cont'd

Update archive_command: (localhost --> push)

```
#
# no restart necesary if archive_mode is already "on"
#   archive_command='/bin/true'
#
system set archive_mode='on'
alter system set archive_command = 'pgbackrest --stanza=main archive-push %p';
select pg_reload_conf();
```

Commands

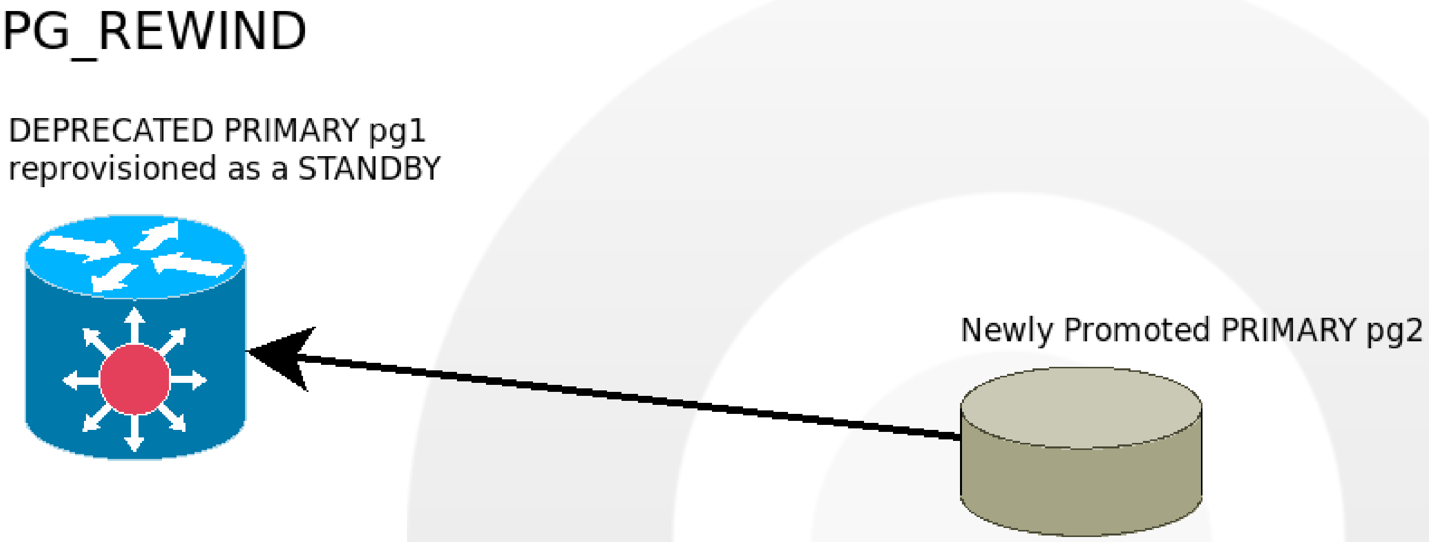
```
# create a stanza called "main"
pgbackrest --stanza=main --log-level-console=info stanza-create

# test the integrity of the setup
# note: if it doesn't work, you must also review the postgres configuration.
pgbackrest check --stanza=main --log-level-console=info

#
# BACKUP: useful for major backups i.e. execute a cron job once a week
#         default location is /var/lib/pgbackrest
#
pgbackrest backup --stanza=main --log-level-console=info --repo1-retention-full=2


#
# DIFF: useful for incremental, daily backups i.e. execute as a cron job every day
# an argument has been added controlling the number of backups retained
#
pgbackrest backup --stanza=main --log-level-console=info --repo1-retention-full=2 --type=diff
```

Point-In-Time-Recovery (PITR) Cont'd

References And Resources

- https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-ARCHIVE-RECOVERY
- https://www.postgresql.org/docs/current/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET
- https://pgbackrest.org/
- https://www.percona.com/blog/

About pg_rewind



About pg_rewind

```
pg_rewind resynchronizes a PostgreSQL cluster
          with another copy of the cluster.


Usage:
  pg_rewind [OPTION]...

Options:
  -D, --target-pgdata=DIRECTORY  existing data directory to modify
      --source-pgdata=DIRECTORY  source data directory to synchronize with
      --source-server=CONNSTR    source server to synchronize with
  -n, --dry-run                  stop before modifying anything
  -N, --no-sync                  do not wait for changes to be written
                                 safely to disk
  -P, --progress                 write progress messages
      --debug                    write a lot of debug messages
  -V, --version                  output version information, then exit
  -?, --help                     show this help, then exit
```

pg_rewind: Example

Steps

- Start with a 2 node Replication Cluster i.e. pg1, pg2
- Confirm postgres runtime parameters are set
- Promote STANDBY, pg2
- Shut down and deprecate PRIMARY, pg1
- Confirm pg1 is in condition to be provisioned as a new REPLICA
- Perform pg-rewind dry run
- Execute pg_rewind and reprovision pg1 as a new STANDBY
- Update datacluster on pg1
    - edit postgresql.auto.conf
    - touch standby.signal
- Start pg1 and validate

pg_rewind: Example Cont'd

```
-- PG1: execute as required on pg1 BEFORE creating REPLICA(s)
alter role postgres with password 'postgres';
alter system set listen_addresses = '*';
alter system set wal_log_hints = 'on'
alter system set wal_keep_size = 100;
```

```
# PG2:create REPLICA, can be with/without slots
/usr/bin/pg_basebackup -d 'host=pg1 user=postgres password=postgres port=5432' \
                      --wal-method=stream \
                       -l basebackup \
                       -D $PGDATA \
                       -R -P -v \
                      --slot=pg2
touch $PGDATA/standby.signal
```

```
-- PG2 promote new PRIMARY
select pg_promote();
-- create new slot on new PRIMARY, pg2, and validate
select * from pg_create_physical_replication_slot('pg1');
select * from pg_get_replication_slots();
```

```
# PG1 TIP: try adding or removing objects AFTER pg2 promotion to make it interesting
systemctl stop postgresql-15
```

pg_rewind: Example Cont'd

```
# PG1: TIP, test first by using '--dryrun'
/usr/pgsql-15/bin/pg_rewind \
    --target-pgdata /var/lib/pgsql/15/data \
    --source-server='user=postgres password=postgres host=pg2'
```

```
# PG1: edit/update postgresql.auto.conf
echo "
primary_conninfo = 'user=postgres password=postgres host=pg2 port=5432'
primary_slot_name = 'pg1'
recovery_target_timeline = 'latest'
wal_log_hints = 'on'
" >> $PGDATA/postgresql.auto.conf

# add standby file
touch $PGDATA/standby.signal
```

```
# PG1
systemctl start postgresql-15
```

PostgreSQL Operations And Troubleshooting

Patroni

Patroni

- About
- Scenario
- Installation
- Configuration
- Administration
- Callbacks

Patroni: About

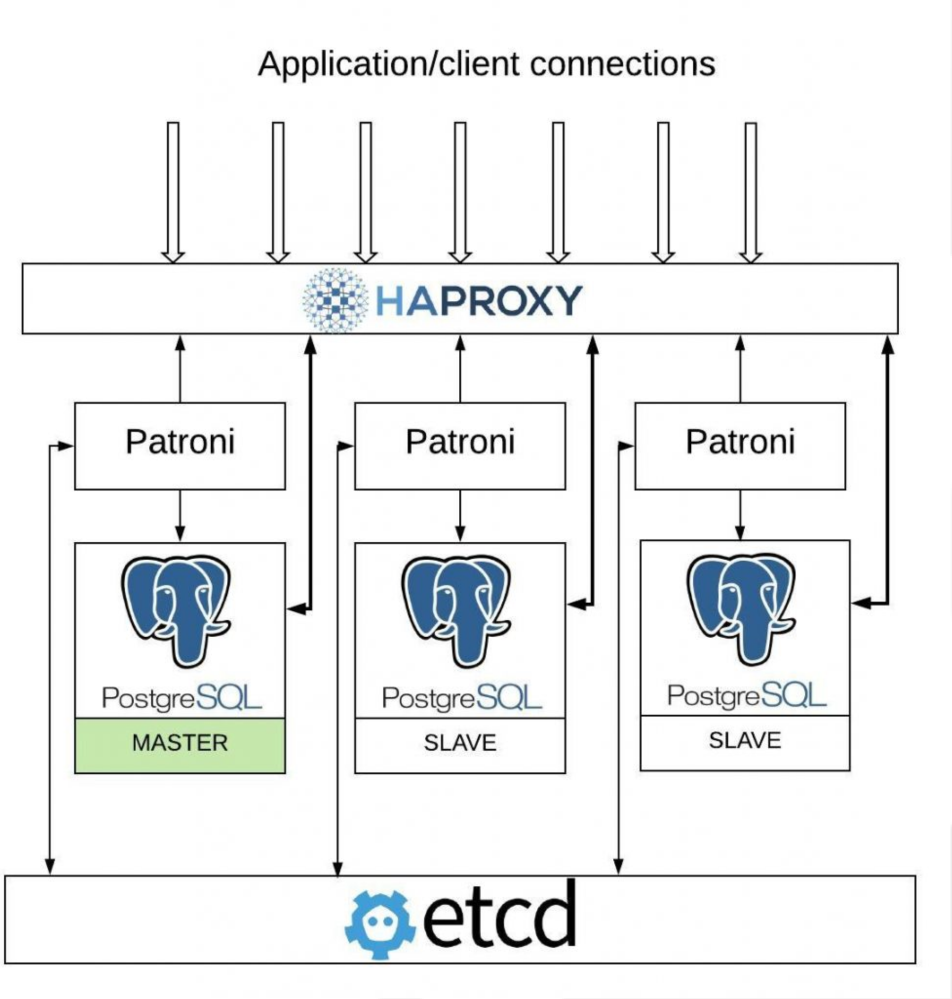
Patroni is a template for you to create your own customized, high-availability solution using Python and - for maximum accessibility - a distributed configuration store like ZooKeeper, etcd, Consul or Kubernetes. Database engineers, DBAs, DevOps engineers, and SREs who are looking to quickly deploy HA PostgreSQL in the datacenter-or anywhere else-will hopefully find it useful.

We call Patroni a “template” because it is far from being a one-size-fits-all or plug-and-play replication system. It will have its own caveats. Use wisely. There are many ways to run high availability with PostgreSQL; for a list, see the PostgreSQL Documentation.

REFERENCES:

```
https://patroni.readthedocs.io/en/latest/index.html
https://patroni.readthedocs.io/en/latest/SETTINGS.html#settings
https://github.com/zalando/patroni
```

Patroni: About



Patroni: Installation

Scenario

- Clean up and verify RPM packages are properly installed
- Initialize a datacluster in a non-standard location on a single postgres host, pg1. Keep in mind that the systemd unit files must be updated on all postgres servers taking into consideration the non-standard location of the datacluster.
- Install a single instance of the ETCD discovery service on HOST "etcd" (192.168.2.221)
- Install Patroni on each host i.e. pg1, pg2, pg3.
- Configure and test ETCD on 192.168.2.221
- Configure Patroni
- Create a patroni managed cluster with pg1 as a single member to this cluster.
- Add new nodes to the cluster i.e. pg2 and pg3 respectively.

Patroni: Installation Cont'd

CENTOS 8

```
# validate packages are installed i.e. pg1, pg2, pg3
dnf install -y epel-release
dnf install -y patroni-etcd
```

```
# shutdown all services i.e. pg1, pg2, pg3
systemctl stop postgresql-15
```

```
# delete all dataclusters i.e. pg1, pg2, pg3
rm -rf /var/lib/pgsql/15/data /mnt/pg/15
```

Patroni: Installation Cont'd

Initialize Primary Host

```
# perform on all three hosts: pg1, pg2, pg3
mkdir -p /mnt/pg
chown postgres.postgres /mnt/pg
```

```
# perform only on pg1
export PGSETUP_INITDB_OPTIONS="--auth-local=peer --auth-host=md5 --pgdata=/mnt/pg/15/data"

/usr/pgsql-15/bin/postgresql-15-setup initdb
```

Patroni: Configuration Cont'd

pg_hba.conf

```
# perform only on host pg1
echo "

# appended rules
host    all             all             0.0.0.0/0               md5
host    all             all             ::0/0                   md5
host    replication     all             0.0.0.0/0               md5
host    replication     all             ::0/0                   md5
" >> /mnt/pg/15/data/pg_hba.conf

echo "listen_addresses = '*' " >> /mnt/pg/15/data/postgresql.auto.conf
```

Patroni: Configuration Cont'd

Create systemd unit override file on pg1, pg2, pg3

```
systemctl edit postgresql-15
```

```
[Service]
# Location of database directory
Environment=PGDATA=/mnt/pg/15/data/
```

Update systemd environment

```
systemctl daemon-reload
```

Patroni: Configuration Cont'd

Create & Validate ROLES

```
# Service start
systemctl start postgresql-15
```

```
-- Update ROLES
alter role postgres with password 'postgres';
create role replicant with replication login password 'mypassword';
alter system set listen_addresses='*';
```

```
postgres=# select usename, passwd from pg_shadow;
  usename   |                 passwd
------------+-------------------------------------
 postgres   | md53175bce1d3201d16594cebf9d7eb3f9d
  replicant | md5361fea8a0a5cfbaf3341cc407b96dfcc

postgres=# \du
                                    List of roles
 Role name  |                         Attributes                          | Member of
------------+------------------------------------------------------------+-----------
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
  replicant | Replication                                                | {}
```

```
# Service stop
systemctl stop postgresql-15
```

Patroni: Configuration Cont'd

HOST etcd (192.168.2.221, installed on CENTOS 7!)

```
# HOST: ETCD (installed on CENTOS 7!)
yum install -y etcd
```

```
# HOST: ETCD
#
# vi /etc/etcd/etcd.conf
#
ETCD_NAME=etcd
ETCD_INITIAL_CLUSTER="etcd=http://192.168.2.221:2380"
ETCD_INITIAL_CLUSTER_STATE="new"
ETCD_INITIAL_CLUSTER_TOKEN="patroni-token"
ETCD_INITIAL_ADVERTISE_PEER_URLS="http://192.168.2.221:2380"
ETCD_DATA_DIR="/var/lib/etcd/postgres.etcd"
ETCD_LISTEN_PEER_URLS="http://192.168.2.221:2380"
ETCD_LISTEN_CLIENT_URLS="http://192.168.2.221:2379,http://localhost:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://192.168.2.221:2379"
ETCD_ENABLE_V2="true"
```

```
# ETCD
systemctl start etcd

etcdctl member list
8d6e29b24fd57235: name=patroni peerURLs=http://192.168.2.221:2380 clientURLs=http://192.168.2.221:2379 isLeader=true
```

Patroni: Installation

Install Patroni packages on all postgres servers

```
# CENTOS 8: PG1, PG2, PG3
dnf install -y patroni-etcd
```

Patroni: Configuration Cont'd

Configuration "patroni.yml": host pg1

```
mkdir -p /etc/patroni/
vi /etc/patroni/patroni.yml
chown postgres.postgres /etc/patroni/patroni.yml
chmod 600 /etc/patroni/patroni.yml
```

ATTENTION:

- ETCD has been provisioned only on host "etcd"
- patroni.yml: IP addresses are edited for each file on each host i.e. pg1, pg2, pg3
- SECURITY RISK: You should set permissions on "/etc/patroni/patroni.yml" to 600

Patroni: Configuration Cont'd

Configuration "patroni.yml": host pg1, cont'd

```
#
# mkdir -p /etc/patroni/
# vi /etc/patroni/patroni.yml
# chown postgres.postgres /etc/patroni/patroni.yml
#

scope: pgcluster
name: pg1
#name: pg2
#name: pg3

restapi:
  listen: 0.0.0.0:8008
  connect_address: 192.168.2.11:8008
# connect_address: 192.168.2.12:8008       #pg2
# connect_address: 192.168.2.13:8008       #pg3

etcd:
  host: 192.168.2.221:2379
# host: etcd:2379                          # pg2, pg3 points to the same ETCD service
```

Patroni: Configuration Cont'd

Configuration "patroni.yml": host pg1, cont'd

```
bootstrap:
  dcs:
    ttl: 30
    loop_wait: 10
    retry_timeout: 10
    maximum_lag_on_failover: 1048576
    synchronous_mode: false
    postgresql:
      use_pg_rewind: true
      use_slots: true
      parameters:
        wal_level: replica
        hot_standby: "on"
        wal_keep_size: 20
        max_wal_senders: 5
        max_replication_slots: 5
        wal_log_hints: "on"
        archive_mode: "on"
        archive_command: "/bin/true"
  initdb:
  - encoding: UTF8
  pg_hba:
  - host replication replicant 127.0.0.1/32 trust
  - host replication replicant 0.0.0.0/0 md5
  - host replication replicant ::0/0 md5
  - host all all 0.0.0.0/0 md5
  - host all all ::0/0 md5
  users:
    admin:
      password: admin
      options:
        - createrole
        - createdb
```

Patroni: Configuration Cont'd

Configuration "patroni.yml": host pg1, cont'd

```
postgresql:
  listen: 0.0.0.0:5432
  connect_address: 192.168.2.11:5432
# connect_address: 192.168.2.12:5432       # pg2
# connect_address: 192.168.2.13:5432       # pg3
  data_dir: /mnt/pg/15/data/
# data_dir: /var/lib/pgsql/15/data
  bin_dir: /usr/pgsql-15/bin
  pgpass: /tmp/pgpass0
  authentication:
    replication:
      username: replicant
      password: mypassword
    superuser:
      username: postgres
      password: postgres
    rewind:
      username: postgres
      password: postgres
  callbacks:
    on_reload: /etc/patroni/callback.sh
    on_restart: /etc/patroni/callback.sh
    on_role_change: /etc/patroni/callback.sh
    on_start: /etc/patroni/callback.sh
    on_stop: /etc/patroni/callback.sh
  parameters:
    unix_socket_directories: "/var/run/postgresql/"
tags:
  nofailover: false
  noloadbalance: false
  clonefrom: false
  nosync: false
```

Patroni: Configuration Cont'd

Configuration "callback.sh": host pg1

```
#!/bin/bash

#
# vi /etc/patroni/callback.sh
# chmod 777 /etc/patroni/callback.sh
#
echo "$(date): $1, $2, $3; callback invoked" > /tmp/patroni_callback.tmp
```

Patroni: Configuration Cont'd

Testing And Validation

```
# anything amiss will be reported immediately
su - postgres -c "/usr/bin/patroni --validate-config /etc/patroni/patroni.yml"

# this will execute a basebackup
su - postgres -c "/usr/bin/patroni /etc/patroni/patroni.yml"
```

Caveat

- inspect the datacluster and look for ...
    - file permissions
    - file ownerships ex: root
    - presence of udesired files ex: *standby.signal*

Patroni

Administration

- about "patronictl"
- configuration files
- status
- maintenance mode
- switchover
- failover
- decommision
- provisioning, add standby
- provisioning, reinitialize node

Patroni

```
patronictl -c /etc/patroni/patroni.yml --help

Usage: patronictl [OPTIONS] COMMAND [ARGS]...
Options:
  -c, --config-file TEXT  Configuration file
  -d, --dcs TEXT          Use this DCS
  -k, --insecure          Allow connections to SSL sites without certs
  --help                  Show this message and exit.

Commands:
  configure    Create configuration file
  dsn          Generate a dsn for the provided member,...
  edit-config  Edit cluster configuration
  failover     Failover to a replica
  flush        Discard scheduled events
  history      Show the history of failovers/switchovers
  list         List the Patroni members for a given Patroni
  pause        Disable auto failover
  query        Query a Patroni PostgreSQL member
  reinit       Reinitialize cluster member
  reload       Reload cluster member configuration
  remove       Remove cluster from DCS
  restart      Restart cluster member
  resume       Resume auto failover
  scaffold     Create a structure for the cluster in DCS
  show-config  Show cluster configuration
  switchover   Switchover to a replica
  topology     Prints ASCII topology for given cluster
  version      Output version of patronictl command or a...
```

Patroni

Configuration Files

INSTALLED

- Patroni:
  - /etc/patroni/patroni.yml
- PostgreSQL:
  - pg_hba.conf
  - postgresql.base.conf
  - postgresql.conf
  - postgresql.auto.conf

Patroni

Status Test, Return State Of Hosts

```
pg1: standby
pg2: standby
pg3: primary
.
systemctl status patroni                     # active
systemctl status postgresql@12-main          # dead, controlled by patroni
pg_lsclusters                                # active
.
```

```
patronictl -c /etc/patroni/patroni.yml list
.
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host      |  Role  |  State  | TL | Lag in MB |
+----------------+---------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | running |  4 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  4 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  4 |           |
+----------------+---------------+--------+---------+----+-----------+
```

TIP, in case you get garbage on your terminal: **export LC_ALL="en_US.UTF-8"**

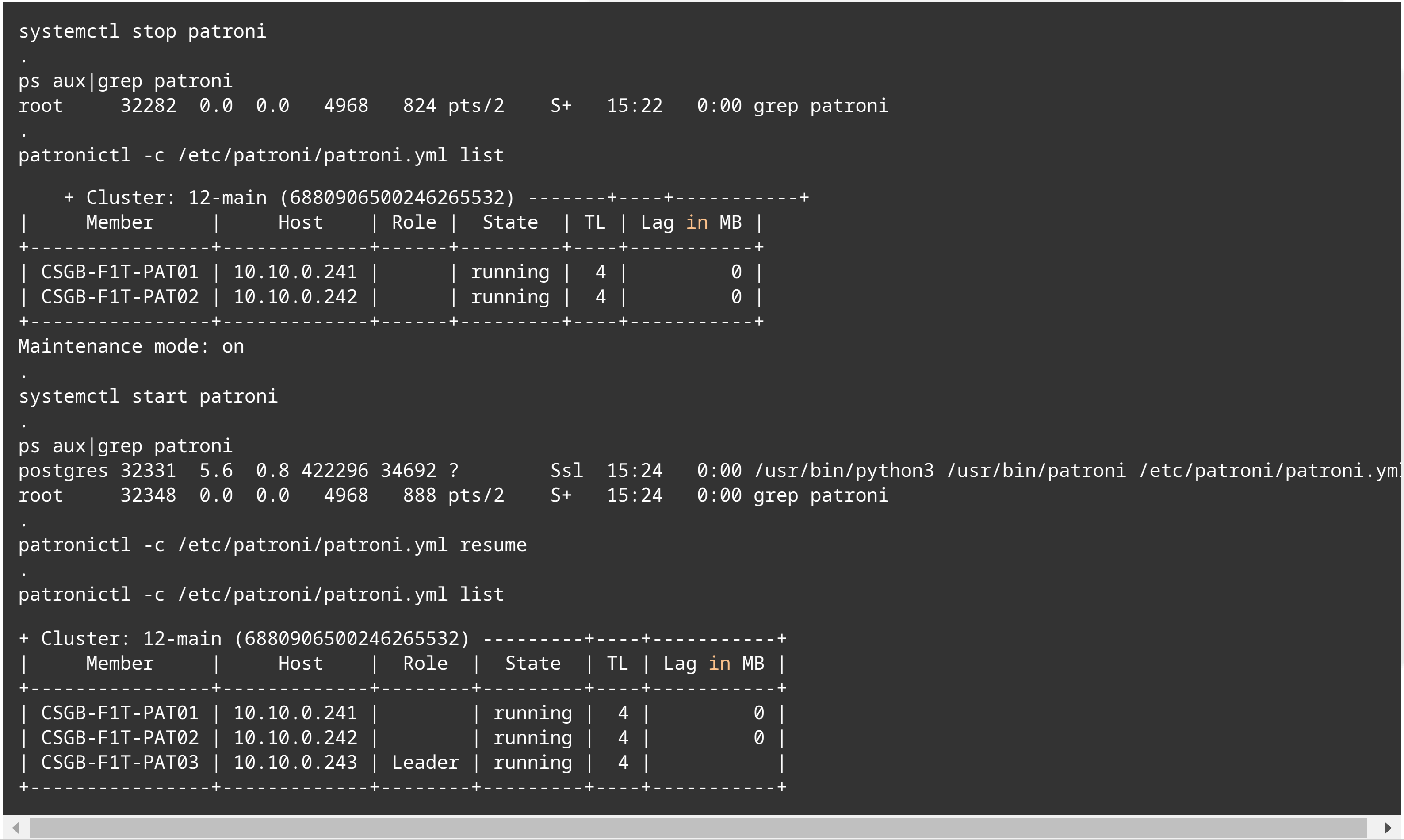
Patroni

Maintenance Mode Test, Turn On And Off A Pg Server

```
# PAT01: execute as postgres
patronictl -c /etc/patroni/patroni.yml pause
.
pg_ctlcluster 12 main stop
.
pg_lsclusters

Ver Cluster Port Status        Owner    Data directory              Log file
12  main    5432 down,recovery postgres /var/lib/postgresql/15/main /var/log/postgresql/postgresql-15-main.log

patronictl -c /etc/patroni/patroni.yml list

+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host    |  Role  |  State  | TL | Lag in MB |
+----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | stopped |    |   unknown |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  4 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  4 |           |

# PAT01: execute as postgres
pg_ctlcluster 12 main start
```

Patroni

Maintenance Mode Test Cont'd



```
systemctl stop patroni
.
ps aux|grep patroni
root     32282  0.0  0.0   4968   824 pts/2    S+   15:22   0:00 grep patroni
.
patronictl -c /etc/patroni/patroni.yml list

    + Cluster: 12-main (6880906500246265532) -------+----+-----------+
|     Member      |      Host     | Role |  State  | TL | Lag in MB |
+-----------------+-------------+------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |      | running |  4 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |      | running |  4 |         0 |
+-----------------+-------------+------+---------+----+-----------+
Maintenance mode: on
.
systemctl start patroni
.
ps aux|grep patroni
postgres 32331  5.6  0.8 422296 34692 ?        Ssl  15:24   0:00 /usr/bin/python3 /usr/bin/patroni /etc/patroni/patroni.yml
root     32348  0.0  0.0   4968   888 pts/2    S+   15:24   0:00 grep patroni
.
patronictl -c /etc/patroni/patroni.yml resume
.
patronictl -c /etc/patroni/patroni.yml list

+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member      |      Host     |  Role  |  State  | TL | Lag in MB |
+-----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | running |  4 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  4 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  4 |           |
+-----------------+-------------+--------+---------+----+-----------+
```

Patroni

Switchover Test, Promote Standby To Primary

```
patronictl -c /etc/patroni/patroni.yml switchover \
            --master CSGB-F1T-PAT03 --candidate CSGB-F1T-PAT01
.
When should the switchover take place (e.g. 2020-12-02T16:28 )  [now]:
Current cluster topology
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host    |  Role  |  State  | TL | Lag in MB |
+----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | running |  4 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  4 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  4 |           |
+----------------+-------------+--------+---------+----+-----------+
.
Are you sure you want to switchover cluster 12-main, demoting current master CSGB-F1T-PAT03? [y/N]: y
2020-12-02 15:28:33.10437 Successfully switched over to "CSGB-F1T-PAT01"
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host    |  Role  |  State  | TL | Lag in MB |
+----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 | Leader | running |  4 |           |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  4 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 |        | stopped |    |   unknown |
+----------------+-------------+--------+---------+----+-----------+
.
postgres@CSGB-F1T-PAT01:~$ patronictl -c /etc/patroni/patroni.yml list
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host    |  Role  |  State  | TL | Lag in MB |
+----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 | Leader | running |  5 |           |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  5 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 |        | running |  5 |         0 |
+----------------+-------------+--------+---------+----+-----------+
```

Patroni

Automated Failover Test, Shutdown Primary

```
# PAT01: execute as root
systemctl stop patroni
.
patronictl -c /etc/patroni/patroni.yml list
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member      |     Host     |  Role  |  State  | TL | Lag in MB |
+-----------------+--------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | stopped |    |   unknown |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  6 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  6 |           |
+-----------------+--------------+--------+---------+----+-----------+
```

Patroni

Decommision Test, Remove Failed Standby

```
patronictl -c /etc/patroni/patroni.yml remove CSGB-F1T-PAT01
.
patronictl -c /etc/patroni/patroni.yml list
.
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host    |  Role  |  State  | TL | Lag in MB |
+----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  6 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  6 |           |
+----------------+-------------+--------+---------+----+-----------+
```

Patroni

Provisioning Test, Add Standby

```
# PAT01: execute as root
systemctl start patroni
#
patronictl -c /etc/patroni/patroni.yml list
.
root@CSGB-F1T-PAT01:/var/log/postgresql# patronictl -c /etc/patroni/patroni.yml list
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member     |     Host     |  Role  |  State  | TL | Lag in MB |
+----------------+--------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | running |  6 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  6 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  6 |           |
+----------------+--------------+--------+---------+----+-----------+
```

Patroni

Provisioning Test, Reinitialize Node I.E. Pat02

```
# PAT02: execute from any node
patronictl -c /etc/patroni/patroni.yml reinit --help
#
patronictl -c /etc/patroni/patroni.yml reinit --wait --force 12-main CSGB-F1T-PAT02
#
Success: reinitialize for member CSGB-F1T-PAT02
Waiting for reinitialize to complete on: CSGB-F1T-PAT02
Reinitialize is completed on: CSGB-F1T-PAT02
.
patronictl -c /etc/patroni/patroni.yml list
+ Cluster: 12-main (6880906500246265532) ---------+----+-----------+
|     Member      |     Host     |  Role  |  State  | TL | Lag in MB |
+-----------------+-------------+--------+---------+----+-----------+
| CSGB-F1T-PAT01 | 10.10.0.241 |        | running |  6 |         0 |
| CSGB-F1T-PAT02 | 10.10.0.242 |        | running |  6 |         0 |
| CSGB-F1T-PAT03 | 10.10.0.243 | Leader | running |  6 |           |
+-----------------+-------------+--------+---------+----+-----------+
```

PERCONA
TRAINING

Questions?

v20210305