# PostgreSQL (System) Administration

PGCon - 2014
Ottawa, Canada

Stephen Frost
*sfrost@snowman.net*

# Stephen Frost

- PostgreSQL
  - Major Contributor, Committer
  - Implemented Roles in 8.3
  - Column-Level Privileges in 8.4
  - Contributions to PL/pgSQL, PostGIS
- Resonate, Inc.
  - Principal Database Engineer
  - Online Digital Media Company
  - We're Hiring! - techjobs@resonateinsights.com

resonate

# Do you read...

- planet.postgresql.org

# Agenda

- Terms
- Installation
- Initial configuration
- Getting connected
- Users / Roles
- Permissions
- Tuning
- Backups
- Monitoring
- Extensions

resonate

# Terms

- "Cluster" ; aka "Instance"
  - One PG server
  - one "postmaster" - listens on one port
  - One set of data files (including tablespaces)
  - Users/Roles and tablespaces at cluster level
  - Replication at cluster level
  - One stream of Write-Ahead-Log (WAL)

# Terms (continued)

- WAL
  - Data stream where changes go first
  - Written to WAL is considered 'committed'
  - WAL is always CRC'd
  - On crash, WAL is replayed
  - Contention point with high write volume

# Terms (continued)

- Table
  - "Fixed" set of columns (can add/remove)
  - Variable number of rows
- Column
  - Named field inside of a table
  - Fixed data type (can be complex)
- Row
  - Single instance of all fields of a table
  - Fields for a row are stored together

# Terms (continued)

- Tablespace
  - Alternate directory/filesystem for PG to store data
  - Can contain objects from any/multiple databases
- Database
  - Lives inside a cluster
  - Schemas at the database level
- Schema
  - Lives inside a single database
  - Do not belong to any tablespace
  - Tables, views, functions at the schema level

# Terms (continued)

- Inheiritance
  - Parent/Child tables
  - Querying parent returns rows from children also
  - Children can add columns to those parent has
  - Differs from SQL:1999 inheiritance
- Partition
  - Implemented using inheiritance in PG
  - CHECK constraints can be used to filter
- Shard
  - One Cluster among many, data spread-out

# Installation

- Debian/Ubuntu/etc
  - apt.postgresql.org
  - Add PGDG sources.list.d
- RedHat/CentOS/etc
  - yum.postgresql.org
  - Download & Install PGDG RPM
- Multiple Major Versions

# Debian Install

- Configs in /etc/postgresql/X.Y/main/
- Initial DB in /var/lib/postgresql/X.Y/main
- Binaries into /usr/lib/postgresql/X.Y/bin
- Logs into /var/log/postgresql/
- Startup logs in /var/log/postgresql also
- One init script starts all major versions

# Debian "Clusters"

- Debian provides wrappers and helper scripts
- pg_lsclusters - lists all PG clusters
- pg_ctlcluster - Control specific clusters
- --cluster option - Specify specific cluster
  - psql --cluster 9.2/main
  - pg_dump --cluster 9.2/main, etc ...

# RedHat Install

- Configs in data directory
- Default DB in /var/lib/pgsql/X.Y/data
- Create DB with 'service postgresql-9.2 initdb'
- Binaries into /usr/pgsql-X.Y/bin
- Logs into /var/lib/pgsql-X.Y/data/pg_log
- Startup logs in /var/lib/pgsql-X.Y/pgstartup.log
- Init script per major version

# PostgreSQL Data Directory

- "Some thing in here do not react well to bullets."
- On Debian, just stay out of it
- On RedHat, be careful to only modify
  - postgresql.conf
  - pg_hba.conf
  - pg_ident.conf
  - pg_log/
- Do NOT touch files in pg_xlog or other dirs
- pg_xlog is PG's WAL- *not* just normal log files

# Initial postgresql.conf

- listen_addresses = '*' (for external access)
- checkpoint_segments = 30+
  - Uses more disk space in pg_xlog
  - Never let that partition run out of space!
- checkpoint_completion_target = 0.9
  - Targets finishing in 90% of time given
- effective_cache_size = half the RAM
  - Never allocated, just for planning
- max_wal_senders = 3
- More later...

# Logging

- postgresql.conf
  - log_connections = on
  - log_disconnections = on
  - line_prefix= '%m [%p]: %q [%l-1] %d %u@%r %a '
  - log_lock_waits = on
  - log_statement = 'ddl'
  - log_min_duration_statement = 100
  - log_temp_files = 0
  - log_autovacuum_min_duration = 0

# pg_hba.conf

- Controls *how* users are authenticated

```
local      DATABASE  USER  METHOD [OPTIONS]
host       DATABASE  USER  ADDRESS  METHOD [OPTIONS]
hostssl    DATABASE  USER  ADDRESS  METHOD [OPTIONS]
hostnossl  DATABASE  USER  ADDRESS  METHOD [OPTIONS]
```

- Read in order, top-to-bottom, first match is used
- 'hostssl' requires SSL connection, no is not SSL
- Special DBs - 'all', 'sameuser', 'replication'
- Special Users - 'all', '+' prefix for role membership
- Address can be IPv4 or IPv6, can include CIDR mask
- Special 'reject' method

# Authentication Methods

- The ones you *should* use ...
- peer
  - Secure, unix-socket-based auth
  - Checks the Unix username of the user
- gss (Kerberos)
  - Integreates w/ MIT/Heimdal Kerberos and AD
  - Recommended for Enterprise deployments
- cert (SSL Certificate)
  - Client-side certificate based authentication
  - Use pg_ident to map CNs to PG usernames

# Authentication Methods

- Acceptable, but not ideal...
- md5
  - Stock username/password
  - Use SSL if you're worried about security
- pam
  - Modules run as postgres user
  - Can't be used directly w/ pam_unix
  - saslauthd can make it work (pam_sasl, saslauthd)
- radius
  - Use SSL if you're worried about security

# Auth Method Don'ts

- trust - Never use this- *no* auth done
- password - Password sent in cleartext
- sspi
  - Windows-specific
  - Uses Kerberos/GSSAPI underneath
- ident
  - Insecure, don't trust it- use 'peer' for local
- ldap
  - Auths against an LDAP server
  - Use Kerberos/GSSAPI if you can

# pg_ident.conf

- Defines mappings which are used in pg_hba

```
map-name   auth-user            pg-user
kerbnames  sfrost@SNOWMAN.NET   sfrost
certname   stephen.frost        sfrost
```

- External-user to PG-user mappings
- Unix user 'joe' can be PG user 'bob'
- Regexps can be used- but be careful
- Also works for Kerberos, client certs, etc.

# Debian configs

- Extra config files in Debian/Ubuntu
- start.conf
  - Controls start of this cluster
  - Can be 'auto', 'manual', 'disabled'
- pg_ctl.conf
  - Options to pass to pg_ctl
  - Generally don't need to modify it
- environment
  - Controlls environment PG starts in
  - Generally don't need to modify it

# RedHat configs

- Basically just the init.d scripts.

# Connecting

- sudo su - postgres
- psql
- \? to see backslash-commands
- \h to get help on SQL queries/commands
- Exit with \q or ctrl-d
- psql -h localhost

# Looking around

- table pg_stat_activity; - aka 'w'
- \l - list databases

```
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
```

- \dn - list schemas

```
  Name  |  Owner
--------+----------
 public | postgres
```

- \db - list tablespaces

```
    Name    |  Owner   | Location
------------+----------+----------
 pg_default | postgres |
 pg_global  | postgres |
```

# User setups

- createuser / CREATE USER
- \password to set passwords
- Privileges
  - Superuser- Do not give this out
  - CreateRole- Creation *and* modification of roles
  - CreateDatabase- Allows database creation
  - Login- Allows user to connect to DB
  - Replication- Only for replication/system user
  - Admin- Allows changing role memberships
  - Inherit- Automatically get GRANTED privileges

# Roles

- Users are really roles
- Groups are implemented with roles
- CREATE ROLE (or just createuser --nologin)
  - Same privilege options
  - Can start as nologin, then be granted login
  - Can cascade
- Any role can be GRANT'd to any other role
- Inherit is default, acts like group privs
- Noinherit means user must run 'set role', ala sudo

# Permissions

- 'public' means 'all users'
- GRANT / REVOKE to give/take away privs, roles, etc
- CONNECT privs on the database (public by default)
- schemas - CREATE, USAGE
  - recommend dropping 'public' or revoke CREATE
  - Use per-user or per-app schemas
- tables - SELECT/INSERT/UPDATE/DELETE/TRUNCATE
- view - same (incl update!); execute as view owner
- columns - SELECT/INSERT/UPDATE
- functions - 'SECURITY DEFINER' are akin to setuid

# Default perms

- Generally 'secure-by-default'
  - *Except* functions- EXECUTE granted by default
  - Owners have all rights on their objects
  - Membership in owning role == ownership
- ALTER DEFAULT PRIVILEGES - for roles
  - FOR ROLE ... IN SCHEMA ... GRANT
  - Can't be applied to just a schema
- GRANT ... ON ALL ... IN SCHEMA
  - For tables, views, sequences, functions
  - One-time operation, new tables will not have privs

# Tablespaces

- Permissions
  - Perms must be 0700, owned by postgres
  - Must explicitly GRANT create rights
- Implementation
  - Symlinks in pg_tblspc directory
  - Recommend against messing with them directly
  - Must be fully-qualified
- GUCs
  - default_tablespace
  - temp_tablespaces

# Tuning

- For a dedicated server
- shared_buffers
  - Will be dedicated to PG for cacheing
  - Up to half of main memory
  - Try 2G on larger servers, more may not help
  - Pre-9.3, need to bump sysctl params
  - Post-9.3, you don't!
  - Defaults to 128MB

# Tuning (continued)

- work_mem
  - Used for in-memory hashing, sorts, etc
  - Can be increased inside a given connection
  - Used many times over- *not* a hard limit
  - Per connection, so be careful
  - Defaults to 1MB (wayy too small..)
- maintenance_work_mem
  - Used for building indexes
  - Make it larger before building an index
  - Defaults to 16MB (that's a very small index)

# Tuning (continued)

- effective_cache_size
  - Tells PG how much of the DB is in memory
  - Half of main memory
  - Never allocated, only for planning purposes
  - Defaults to 128MB
- autovacuum
  - On a high-rate server, make it more aggressive
  - Increase max_workers
  - Decrease autovacuum_vacuum_cost_delay
  - Defaults are for lightly loaded systems

# Tuning (continued)

- pg_xlog
  - Sequential writes
  - Put on dedicated disks
  - Monitor very closely for space
- pg_stat_tmp
  - Consider tmpfs
  - Written to by stats collector constantly
  - File per-DB in 9.3+, helps a lot

# Slow Queries

- Logging all queries hurts
- log_min_duration_statement
  - Logs queries over time
  - Includes duration (no need for log_duration)
- pgfouine - Log Analyzer
  - Best with specific log_line_prefix
  - Generates very nice reports
  - Various sorts- total time, max length, etc

# Config Bump-Ups

- max_connections = 100
  - Consider using pg_bouncer
  - # connections == # of CPUs is ideal
- shared_buffers = couple gig
  - Probably not more than 3-4G (Test!)
- maintenance_work_mem = maybe a gig
  - Used for building indexes
- max_locks_per_transaction = 128
  - More if you have lots of objects
  - # locks available is actually this * max_conn

# Simple Backups

- Extremely important!
- pg_basebackup w/ WAL recieve
  - Binary-based backup
  - *MUST* have WAL files backed up also!
  - Needs to connect to 'replication' DB
- pg_dump
  - Logical, text-based backup
  - Does not back up indexes, must rebuild
  - Requires lightweight locks on everything
- Test restoring your data!

# Parallel Backups

- pg_dump support in recent versions
- pg_restore also supports- not transactional
- Binary backups
  - Use rsync
  - Parallelize by tablespace
  - No parallel option for pg_basebackup (yet)

# PITR Backups

- Point-in-time-Recovery w/ WAL
  - From base-backup, play forward WAL
  - Can stop at any point-in-time
- Requires a base/binary backup (pg_basebackup)
- Must archive all WAL
  - WAL archived with archive_command
  - Only WAL after a base backup is useful

# archive_command

- %f replaced with WAL filename
- %p replaced with full path to WAL
  - test -f /archive/%f &&
  - cp %p /archive/%f
- Be sure to test
- Monitor your postgres logs!
- Must return zero ONLY on success

# Restoring!

- Make sure to test your backups!
- Test by doing a *restore*!
- Test regularly! (at least once a year..)
- Consider multiple scenarios
  - Tape-based restore?
  - Restore from off-site?
  - Fail-over?
  - How much data lost?
  - How much downtime?

# recovery.conf

- restore_command
- %f is WAL file needed
- %p is where to put it
  - cp /archive/%f %p
- Only return zero when successful!
- Will be called for non-existant files

# Replication

- Read-only streaming slaves
- Set up WAL archiving
  - Not strictly required
  - Very recommended
- Initial copy with pg_basebackup
- Configure connection in recovery.conf
- recovery.conf must live in data dir
- Monitor lag- replica can fall behind

# Monitoring

- check_postgres.pl
- Useful with Nagios, Icinga, MRTG, etc.
- Provides metrics as well as monitoring
- Allows custom query for monitoring
- Minimum set of checks

```
archive_ready (if doing WAL archiving)  --- Number of WAL .ready files
autovac_freeze                          --- How close to Autovacuum Max Freeze
backends (Metric)                       --- Number of Backends running
dbstats (Metrics)                       --- Lots of different stats
listener (If using LISTEN/NOTIFY)       --- Checks if anyone is LISTEN'ing
locks (Metric)                          --- Number of locks held
pgbouncer options (if using pgbouncer)  --- Various pgbouncer checks
txn_idle                                --- Transactions idle for X time
txn_time                                --- Transactions longer than X time
txn_wraparound                          --- How close to transaction wraparound
```

# Log Monitoring

- PG logs are multi-line
- tail_n_mail works great
- Other solutions do not understand PG logs
  - syslog-based
  - logstash
  - logcheck
- Automatically-processed CSV log

# Extensions

- Install -contrib package
- Use PGXN - http://pgxn.org
- table pg_available_extensions;

```
       name        | default_version | installed_version |                              comment
-------------------+-----------------+-------------------+-----------------------------------------------------------------------
 file_fdw          | 1.0             |                   | foreign-data wrapper for flat file access
 dblink            | 1.0             |                   | connect to other PostgreSQL databases from within a database
 plpgsql           | 1.0             | 1.0               | PL/pgSQL procedural language
 pg_trgm           | 1.0             |                   | text similarity measurement and index searching based on trigrams
 adminpack         | 1.0             |                   | administrative functions for PostgreSQL
 ip4r              | 2.0             |                   |
 hstore            | 1.1             |                   | data type for storing sets of (key, value) pairs
```

- adminpack allows superuser to change anything..
- \dx lists installed extensions

# Extensions (cont'd)

- Requires superuser to install
- Often include compiled C code - .so's
  - C code can crash the backend, use caution
  - C code has access to everything
- PGXN is pretty 'open'
- Modules from -contrib maintained by PGDG

# Thank you!

Stephen Frost
*sfrost@snowman.net*
*@net_snow*

resonate