Administration of PostgreSQL

Stephen Frost
stephen@crunchydata.com

Crunchy Data, Inc.

September 16, 2015

Stephen Frost

- Chief Technology Officer @ Crunchy Data, Inc.
- Committer
- Major Contributor
- Row-Level Security in 9.5 (coming this fall)
- Column-level privileges in 8.4
- Implemented the roles system in 8.3
- Contributions to PL/pgSQL, PostGIS

Terms: Cluster/Instance

A single, complete, running PostgreSQL system.

- One PostgreSQL Server
- Listening on one port (may be multiple addresses)
- One set of data files (including tablespaces)
- One stream of Write Ahead Logs

Operations done on a cluster:

- Initialization (initdb)
- Start / Stop the cluster
- File-level Backup and Restore
- Streaming Replication

Objects defined at a Cluster level:

- Users/Roles
- Tablespaces
- Databases

Terms: Database

Container for schemas and database-level objects.
Database-level permissions include:

- CONNECT - allowed to connect, default allow to all
- CREATE - allowed to create schemas
- TEMPORARY - allowed to create temporary objects

Terms: Schema

Container for regular objects.
Schema-level permissions include:

- CREATE - allowed to objects in schema
- USAGE - allowed to use objects in schema

Individual objects have various permissions which can be granted, depending on the specific type of object.

Terms: Tablespace

Alternative directory to store PostgreSQL data files for:

- Tables
- Indexes

Cluster-level and therefore can contain objects from multiple databases.

Terms: Write Ahead Log/WAL

Data stream where changes are written to initially. Also know as the "transaction log" or XLOG, lives in "pg_xlog"

- Only committed once written to WAL and synced to disk
- WAL changes are CRC'd
- Changes written to data files in background
- On crash, replay of WAL ensures consistency
- Potential contention point with high write volume
- Contains Full Page changes and Incremental changes
- First change after checkpoint is a full page change

Terms: Checkpoint

Periodic process to ensure data has been written out to the main database files.

- Happens at least every 5 minutes by default
- Logging of checkpoints enabled via log_checkpoints
- May be forced due to running out of space for WAL

PostgreSQL Global Development Group Packages

- Provided by the PostgreSQL community
- Up-to-date packages for major distributions
- Concurrent installation of multiple major versions
- Smooth major version upgrades
- Well maintained by the same developers as PostgreSQL
- Supported through the community mailing lists
- Updates released in coordination with PostgreSQL

Debian/Ubuntu/etc Installation

- Use apt.postgresql.org
- Add PGDG sources.list.d
- 'lsb_release' -c to determine codename

```
/etc/apt/sources.list.d/pgdg.list:

deb http://apt.postgresql.org/pub/repos/apt/ wheezy-pgdg main

wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | \
  sudo apt-key add -
apt-get update
apt-get upgrade
apt-get install postgresql-9.4
```

Debian/Ubuntu/etc Configuration

- Config files in /etc/postgresql/X.Y/main/
- Database files in /var/lib/postgresql/X.Y/main/
- Wrappers for most binaries
- Actual binaries in /usr/lib/postgresql/X.Y/bin
- Logs in /var/log/postgresql
- Startup logs also in /var/log/postgresql
- Single init script to start all major versions

Debian/Ubuntu/etc "Clusters"

- Debian-provided wrappers and helper scripts
- Allows multiple concurrent clusters, same or different versions
- pg_lsclusters - lists all PG clusters
- pg_ctlcluster - pg_ctl for clusters
- –cluster option - Specify which cluster to work on

```
postgres@beorn:~$ pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
9.4 main    5435 online postgres /var/lib/postgresql/9.4/main   \
                          /var/log/postgresql/postgresql-9.4-main.log
9.4 testudr 5433 online postgres /var/lib/postgresql/9.4/testudr \
                          /var/log/postgresql/postgresql-9.4-testudr.log
postgres@beorn:~$ psql --cluster 9.4/main -l
                                     List of databases
Name      |  Owner   | Encoding |   Collate   |    Ctype    | Privs
----------+----------+----------+-------------+-------------+-------
postgres | postgres | UTF8      | en_US.UTF-8 | en_US.UTF-8 |
...
```

RedHat Installation

- Use yum.postgresql.org
- Install PGDG repo packages
- Initialize the cluster
- Similar steps for other versions
- Multiple versions can run in parallel

```
yum install \
  http://yum.postgresql.org/9.4/redhat/rhel-7-x86_64/
  pgdg-redhat94-9.4-1.noarch.rpm

# Install PostgreSQL packages
yum groupinstall "PostgreSQL Database Server 9.4 PGDG"
# Create initial database
/usr/pgsql-9.4/bin/postgresql94-setup initdb
# Start PG on boot
systemctl enable postgresql-9.4.service
```

RedHat Configuration

- Default data directory is /var/lib/pgsql/X.Y/data
- Configs in data directory
- Binaries installed into /usr/pgsql-X.Y/bin
- Logs in /var/lib/pgsql-X.Y/data/pg_log
- Startup logs in /var/lib/pgsql-X.Y/pgstartup.log
- Indepedent init script needed for each version
- No helper scripts ala Debian/Ubuntu

PostgreSQL Config Files

- postgresql.conf - General server configuration
- pg_hba.conf - Configure Host-Based Authentication
- pg_ident.conf - User mapping tables
- pg_log - Log files (RedHat only)

Debian-based systems:

- Files live in /etc/postgresql/X.Y/main

RedHat-based systems:

- Files live in data directory
- Be careful to NOT modify other files in data directory!
- pg_xlog is the WAL- *not* normal log files!

Debian-specific Config Files

In the per-cluster directory (eg: /etc/postgresql/X.Y/main):

- start.conf
  - Controls start of the cluster
  - Options are 'auto', 'manual', 'disabled'
- pg_ctl.conf
  - Options to pass to pg_ctl
  - Generally should be left alone
- environment
  - Environment settings for starting PostgreSQL
  - Generally should be left alone

Debian-specific Config Files

In /etc/postgresql-common:

- createcluster.conf
    - Defaults for the pg_createcluster command
    - Allows alternative data and xlog directories
    - Options for initdb
- user_clusters
    - Controls default cluster for users to connect to
    - Can be user-specific
    - Can also specify alternative default database
- pg_upgradecluster.d/
    - Scripts to be run during pg_upgrade
    - Can be populated by extensions

RedHat-specific Config Files

- Init scripts
- Recent changes reduce need to modify them
- Port no longer specified in init scripts

Initial Configuration

- Defaults are decent for small instances
- listen_addresses = '*' (to allow external access)
- checkpoint_segments = 30+
  - Allows more space usage in pg_xlog
  - Never let pg_xlog location run out of space!
- checkpoint_completion_target = 0.9
  - Targets finishing in 90% of time allocated
  - Overall time deffined by checkpoint_timeout
- effective_cache_size = half of RAM
- max_wal_senders = 3

Logging Configuration

Logging defaults are terrible, in general. Recommendations:

- log_connections = on
- log_disconnections = on
- log_lock_waits = on
- log_statement = 'ddl'
- log_min_duration_statement = 100
- log_temp_files = 0
- log_autovacuum_min_duration = 0

Logging Configuration - log_line_prefix

- Prefix for each log line
- log_line_prefix = '%m [%p]:%q [%l-1] %d %u@%r %a '
  - %m - Timestamp, with milliseconds
  - %p - Process ID/PID
  - %q - Stopping point for non-session processes
  - %l - Per-session number of log line
  - %d - Database name
  - %u - Login user name
  - %r - Remote host and port
  - %a - Application name

Basic pg_hba.conf configuration

- 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 used
- 'hostssl' matches if SSL used
- Special DBs- 'all', 'sameuser', 'replication
- Special users- 'all', '+role' for membership
- Address can be IPv4 or IPv6, can include CIDR mask
- 'reject' method denies access on match

Authentication Methods

The ones you should use:

- peer
  - Secure, unix-socket-based auth
  - Passes through Unix user connected
- gss (aka Kerberos) / sspi (for Windows)
  - Integrates with MIT/Heimdal Kerberos
  - Integrates with Active Directory
  - Strongly recommended for Enterprise deployment
- cert (SSL Certificate-based)
  - Client-side certificate based authentication
  - Map CNs to PG usernames (pg_ident.conf)

Authentication Methods

Acceptable, but not ideal:

- md5
  - Stock username/password authentication
  - Fixed, relatively weak algorithm
- pam (Pluggable Authentication Methods)
  - Uses system PAM configuration
  - PAM modules run as postgres user, not root
  - saslauthd can be used to run as root with pam_sasl
  - Use with SSL for network security
- radius
  - Integrates with Enterprise RADIUS solutions
  - Use with SSL for network security
- password
  - Traditional password-based authentication
  - Use with SSL for network security

Authentication Methods

Methods to avoid:

- ldap
  - Uses simple bind/connect to LDAP server
  - Proxies credentials provided
  - GSSAPI/SSPI should be used instead, if possible
- ident
  - Network-based, intended to be like 'peer'
  - No real authentication of remote server
  - Extremely insecure
  - Should be removed eventually
- trust
  - Bypasses all authentication
  - Accepts any user

pg_ident Configuration

Defines mappings from system user to PostgreSQL user.

```
map-name   auth-user           pg-user
peermap    joe                 bob
certname   stephen.frost       sfrost
kerbnames  sfrost@SNOWMAN.NET  postgres
kerbnames  /^(.*)@SNOWMAN\.NET$ \1
```

- Regexps can be used- but use caution and anchor them
- Unix user 'joe' can connect as PG user 'bob'
- Client certificate Common Name 'stephen.frost' as 'sfrost'
- Kerberos principal 'sfrost@SNOWMAN.NET' as 'postgres'
- Kerberos principals '*@SNOWMAN.NET' as that user
- map specified in pg_hba.conf with 'map=peermap' as option

Is PostgreSQL up?

Standard tools work-

```
postgres@beorn:~$ service postgresql status
9.4/main (port 5435): online
9.4/testudr (port 5433): online
```

PostgreSQL also includes 'pg_isready':

```
postgres@beorn:~$ pg_isready --cluster 9.4/main
/var/run/postgresql:5435 - accepting connections
```

Connect with client tool 'psql':

```
postgres@beorn:~$ psql --cluster 9.4/main
psql (9.4.4)
Type "help" for help.

postgres=#
```

Note that –cluster is a Debian-specific option.

psql

psql is the user interface included with PostgreSQL and is extremely powerful.

- psql commands start with
- All other commands sent to server as queries
- `\?`
  to see list of psql backslash-commands
- `\h`
  to get syntax for SQL queries/commands
- Exit using
  `\q`
  or ctrl-d
- Queries return tables or command results
- Expanded output format can be toggled using
  `\x`

Who is Connected?

```
postgres=# \x -- Expanded display is on.
postgres=# table pg_stat_activity ;
-[ RECORD 1 ]----+------------------------------
datid            | 12173
datname          | postgres
pid              | 12742
usesysid         | 10
usename          | postgres
application_name | psql
client_addr      |
client_hostname  |
client_port      | -1
backend_start    | 2015-09-16 07:17:06.713886-04
... xact_start, query_start, state_change
waiting          | f
state            | active
backend_xid      |
backend_xmin     | 733
query            | table pg_stat_activity ;
```

What databases exist?

```
postgres=# \l
List of databases
-[ RECORD 1 ]-----+----------------------
Name              | postgres
Owner             | postgres
Encoding          | UTF8
Collate           | en_US.UTF-8
Ctype             | en_US.UTF-8
Access privileges |
-[ RECORD 2 ]-----+----------------------
Name              | template0
Owner             | postgres
Encoding          | UTF8
Collate           | en_US.UTF-8
Ctype             | en_US.UTF-8
Access privileges |
-[ RECORD 3 ]-----+----------------------
Name              | template1
[...]
```

What are templates?

- CREATE DATABASE copies an existing database
- Uses template1 by default
- Objects can be added to template1
- template0 contain only the standard objects
- Never modify template0

Creating Users

Two methods exist: command-line 'createuser' and SQL 'CREATE USER':

```
postgres@beorn:~$ createuser test1
postgres@beorn:~$
postgres@beorn:~$ psql
psql (9.4.4)
Type "help" for help.

postgres=# CREATE USER test2;
CREATE ROLE
postgres=# \password test2
Enter new password:
Enter it again:
postgres=#
```

User 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 'group' privileges

Roles

- Users are Roles, Roles are Users
- Groups are Roles too
- CREATE ROLE (or just createuser –nologin)
- Any role can be GRANT'd to any other role (no loops)
- Inherit is default, which acts like group privileges
- Noinherit forces user to run 'set role', ala 'sudo'

Admin Role

- For 'sudo'-like administration with SET ROLE

```
postgres=# CREATE ROLE admin WITH NOINHERIT;
CREATE ROLE
postgres=# GRANT postgres TO admin;
GRANT ROLE
postgres=# CREATE USER joe;
CREATE ROLE
postgres=# GRANT admin TO joe;
GRANT ROLE
```

- On login, joe has rights of joe AND admin, but NOT postgres
- joe can SET ROLE to postgres, also becomes superuser
- No way, currently, to require password for SET ROLE

Permissions

GRANT and REVOKE commands control privileges in PG

- Default 'public' schema allows any user to create objects
- Recommended to REVOKE CREATE on 'public' schema
- Use per-user schemas instead
- Nearly all objects have some set of permissions
- Type of privileges available varies by object

Common object types and their common privileges:

- databases - CONNECT, CREATE, TEMPORARY
- schemas - CREATE, USAGE
- tables - SELECT, INSERT, UPDATE, DELETE, TRUNCATE
- views - same as tables (including update!)
- columns - SELECT/INSERT/UPDATE
- functions - EXECUTE; can be SECURITY DEFINER aka setuid

Default Permissions

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
- Applied to a role, can't be applied to just a schema
- New objects will have default privileges specified

GRANT ... ON ALL ... IN SCHEMA

- Convenience command for lots of GRANTs (or REVOKEs)
- For tables, views, sequences, functions
- One-time operation, new tables will not have privs

Database Size Information

Database size seen with pg_database_size():

```
postgres=# select pg_size_pretty(pg_database_size('postgres'));
 pg_size_pretty
----------------
 6539 kB
(1 row)
```

Size of individual tables with pg_total_relation_size():

```
postgres=# select pg_size_pretty(pg_total_relation_size('pg_class'));
 pg_size_pretty
----------------
 232 kB
(1 row)
```

Database Size Information

Size of just the table data (no indexes, etc) with pg_relation_size():

```
postgres=# select pg_size_pretty(pg_relation_size('pg_class'));
 pg_size_pretty
----------------
 64 kB
(1 row)
```

Size of all tables in a schema:

```
postgres=# select
postgres-# pg_size_pretty(sum(
postgres-# pg_total_relation_size(schemaname || '.' || tablename)
postgres-# )) from pg_tables where schemaname = 'pg_catalog';
 pg_size_pretty
----------------
 6496 kB
(1 row)
```

Creating a Tablespace

- Identify directory on server to use
- Ensure directory is empty
- Directory should be owned by postgres user
- Permissions must be 0700 (u=rwx,g=,o=).
- Must specify full path to directory
- Tablespace belongs to specific cluster
- Do not use mount point, create directory under it

```
postgres=# CREATE TABLESPACE ts1 LOCATION '/volume1/ts1';
CREATE TABLESPACE
```

Tablespace Info

- pg_default contains objects not in other tablespaces
- pg_global is a special tablespace for shared catalogs
- Size information available with pg_tablespace_size()

```
postgres=# \db
             List of tablespaces
    Name    |  Owner   |   Location
------------+----------+--------------
 pg_default | postgres |
 pg_global  | postgres |
 ts1        | postgres | /volume1/ts1
(3 rows)

postgres=# select pg_size_pretty(pg_tablespace_size('ts1'));
 pg_size_pretty
----------------
 4096 bytes
(1 row)
```

Dropping a Tablespace

- Must be empty
- May have to connect to multiple databases to drop objects

```
postgres=# DROP TABLESPACE ts1;
DROP TABLESPACE
```

Simple File-Based Backups

Backups are critical to any production deployment!

- pg_basebackup with WAL receive
- One-time, consistent binary-based backup
- Requires full backup every time
- MUST have WAL files included via WAL receive or other means
- Connects as a replication user to the replication DB
- Includes indexes

Simple Logical-Based Backups

Backups are critical to any production deployment!

- pg_dump and pg_dumpall
- Logical, text-based backup (includes SQL statements)
- Indexes are NOT included- they have to be rebuilt
- User using pg_dump must have access to all objects (eg: superuser)
- Requires a lock on every object in the system

Restoring!

Backups must be tested or they don't work!

- Regular testing is critical to ensure they work when needed
- Consider multiple failure scenarios
    - Tape-based restore
    - Restore from off-site
    - Fail-over / fail-back
    - How much data loss is acceptable?
    - How much downtime is acceptable?

Simple restoring with pg_basebackup

Backups must be tested or they don't work!

- pg_basebackup creates a tar file
- Extract the tar file into a directory
  - All data files will be included
  - All WAL files necessary for restore included
  - All data since backup lost

PGBackRest

Available from https://github.com/pgmasters/backrest
Features include:

- All archive and backup data compressed by default
- Network traffic also compressed
- Extremely simple for the simple case- but able to be highly complex
- Local and remote backup support
- Full Point-in-time-Recovery; supports all PG has to offer
- Incremental, differential, and full backups supported
- Multi-threaded capability for large systems

PGBackRest - postgresql.conf

- Required setings in postgresql.conf:
- wal_level = archive
- archive_mode = on
- archive_command = 'pgbackrest –stanza=main archive-push %p'

PGBackRest - pgbackrest.conf

Configuration of pgBackRest:

```
/etc/pgbackrest.conf:
[global:general]
repo-path=/pgbackups

[main]
db-path=/data/db
```

- '[main]' is a stanza, represents a PostgreSQL cluster
- 'db-path' is the path to the data directory
- 'repo-path' is the path for pgBackRest to store WAL and backups

PGBackRest Info Command

Info command of pgBackRest:

```
postgres@server:~$ pgbackrest info
stanza main
    status: ok
    oldest backup label: 20150901-220418F
    oldest backup timestamp: 2015-09-01 22:03:25
    latest backup label: 20150901-220418F
    latest backup timestamp: 2015-09-01 22:03:25
postgres@server:~$
```

- Reports on all clusters/stanzas configured
- JSON output format also provided

check_postgres.pl

- Used with Nagios, Icinga, MRTG, etc
- Able to provide metrics as well
- Supports custom queries

Minimum recommended set of checks:

```
archive_ready (if doing WAL archiving)  --- Number of WAL .ready files
autovac_freeze                          --- How close to AV Max Freeze
backends (Metric)                       --- Number of Backends running
dbstats (Metrics)                       --- Lots of different stats
listener (If using LISTEN/NOTIFY)       --- Is anyone 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                                --- TXNs longer than X time
txn_wraparound                          --- How close to TXN wraparound
```

Monitoring Log Files

- PostgreSQL log entries can be multi-line
- tail_n_mail understands PG log files
- Most other solutions do not (syslog, logstash, logcheck..)
- Automatically processed CSV files also good

CSV log files configured using:

```
log_destination = 'csvlog'
logging_collector = 'on'
```

Options also available to control log rotation, filename, permissions, and location.

shared_buffers

shared_buffers is a pool of memory dedicated to PostgreSQL for cacheing.

- Reduces need to request data from the system
- Default changes over time, currently 128MB (quite small)
- Optimal value varies quite a bit depending on specific workload
- If the entire database can fit in memory, have a large value
- Otherwise, consider 2G or so, larger can be bad
- Pre-9.3, sysctl parameters have to be adjusted
- Post-9.3, no sysctl changes required!

work_mem

Amount of memory PG may use for individual operations.

- Uses include building hash tables, doing sorting, etc
- Can be changed inside a given session
- Each use may use up to this amount, complex queries use many times the value
- Also per connection- lots of connections with complex queries chews up memory
- Default is 1MB, which is quite small

maintenance_work_mem

Amount of memory PG may use for creating indexes, performing VACUUM, etc.

- Not double-counted like work_mem
- Can be changed inside a given session
- Defaults to only 16MB (which would be a very small index...)
- Larger can greatly improve index creation speed
- Probably bump up, but not too much for the default
- Increase in a session prior to building an index

effective_cache_size

Hints to PostgreSQL how much system memory is being used for filesystem cacheing.

- Never actually allocated, just used for planning purposes
- PostgreSQL uses this number to take a guess as to if data is in memory
- Defaults to 128MB, very small amount of cache
- Reasonable setting is half of main memory on most systems

Autovacuum

The autovacuum process is a cleanup routine which runs periodically in PostgreSQL to mark dead data as reusable space.

- Defaults are for small, low transaction rate, systems
- On a busy server, autovacuum needs to run more frequently, not less
- Increate the number of workers allowed to run (max_workers)
- Decrease the cost delay (or eliminate it)- autovacuum_vacuum_cost_delay

Managing connections

PostgreSQL performs best when the number of active backends is near the number of CPU cores in the system, and the number of idle connections is minimal.

- max_connections can be bumped to 100-200, but avoid going higher
- Use pgBouncer- very good connection pooler
- Use connection pooling in the application stack
- Another connection pooler is PGPool
- Monitor number of connections, especially idle ones
- Watch for idle-in-transaction connections, can cause bloat

Managing Locks

PostgreSQL has a single pool of locks, but the size is based on max_connections and max_locks_per_transaction.

- max_locks_per_transaction defaults to 128
- Heavy-weight locks (not per-row)
- A heavy-weight lock is required for each object accessed during a session
- Consider pg_dump, which locks all objects and if enough locks exist

Checkpoints

- Ensure checkpoints are happening due to time, not other causes
- If checkpoints due to XLOG, increase checkpoint_segments
- Consider changing checkpoint_timeout
- Longer the checkpoint_timeout, longer potential downtime due to crash
- checkpoint_timing should be increased to 0.9
- checkpoint_warning generally not helpful- use log_checkpoints

pgBadger

pgBadger is a log analyzer for PostgreSQL and produces reports about slow queries.

- Generates extremely useful reports
- Requires specific settings in postgresql.conf to parse log file
- log_min_duration_statement = 0 # May generate a lot of log
- log_line_prefix at least '%t [%p]: [%l-1] '

Thank You

- Questions?