

A guide to correct PostgreSQL setup and management

Gavin Sherry

`gavin@alcove.com.au`

Alcove Systems Engineering

January 16, 2007

Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration
- 4 Hardware
- 5 Routine Maintenance
- 6 Replication/high availability

Latest version

- Version **8.2.1**
- PostgreSQL has good backward compatibility
- Most applications should use the latest and greatest

Installation of PostgreSQL

- In production, use a package manager (unless unsupported)
 - If your OS doesn't have one, why not create it?
 - And, tell us about it
- Other people need to know what's going on and packages are The Right Way

Packages

- RHEL/CentOS: <http://www.postgresql.org/download>
- Debian/Ubuntu: a deb package is available via the Usual Method
- Gentoo: the Usual Method
- Solaris: <http://pgfoundry.org/projects/solarispackages/>
- And even Windows – gasp!
<http://www.postgresql.org/download>

Problems with packages

- Upgrades are a head ache across major versions
- If you're doing a dump/restore upgrade, both versions of Postgres must be installed



Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration
- 4 Hardware
- 5 Routine Maintenance
- 6 Replication/high availability

Data upgrade concepts

- Generally, minor version upgrades (**8.2.0** → **8.2.1**) require only installation of binaries
 - Check the release notes though!
 - We won't corrupt your data if you make a mistake
- So, if you're doing a major version upgrade:
 - Do a test run data upgrade first
 - Ideally, thoroughly test the impact of this on your application(s) in a testing/staging environment
 - Prior to 8.1, we had a bug which allowed non-unicode characters into unicode databases - repair data before upgrades

Fixing utf-8 issues

- Firstly, it's important to actually understand Unicode
 - <http://www.joelonsoftware.com/articles/Unicode.html>

Fixing utf-8 issues

- Firstly, it's important to actually understand Unicode
 - <http://www.joelonsoftware.com/articles/Unicode.html>
- Next, dump every database as text with `pg_dump`

Fixing utf-8 issues

- Firstly, it's important to actually understand Unicode
 - <http://www.joelonsoftware.com/articles/Unicode.html>
- Next, dump every database as text with `pg_dump`
- Run the dump through `iconv`
 - If there's no output, there's no problem

Fixing utf-8 issues, cont.

- Usually, the problem is LATIN-1 or WINDOWS-1250 that have gotten in to your database
- Look at your app, what client encodings are your users likely to have
 - Also, the app is broken anyway

Example

```
$ iconv -f utf-8 -t utf-8 dump.sql > /dev/null
iconv: illegal input sequence at position 1000
$ head -c 1010 dump.sql | tail -c 20 | od -c
0000000  h .   B u t   i s n 222 t   i t
0000020  h i s ?
0000024
```

Dealing with major versions

- The professional way:
 - 1 Install the new package in a temporary location; or

Dealing with major versions

- The professional way:
 - 1 Install the new package in a temporary location; or
 - 2 Make a local source build, install in temporary location; or

Dealing with major versions

- The professional way:
 - 1 Install the new package in a temporary location; or
 - 2 Make a local source build, install in temporary location; or
 - 3 Use `s1ony` to do the upgrade

Dealing with major versions

- The professional way:
 - ① Install the new package in a temporary location; or
 - ② Make a local source build, install in temporary location; or
 - ③ Use `s1ony` to do the upgrade
- Once the upgrade is complete, install the new package correctly

Dealing with major versions

- The professional way:
 - 1 Install the new package in a temporary location; or
 - 2 Make a local source build, install in temporary location; or
 - 3 Use `s1ony` to do the upgrade
- Once the upgrade is complete, install the new package correctly



Data upgrade procedures: dump/restore

- You must dump with the `pg_dump` binary of the version being upgraded to
- This assumes you're upgrading to 8.1 or later

Steps

- 1 `newpg/pg_dumpall > dump.sql`
- 2 Start the new version of PostgreSQL on a different port
- 3 `newpg/psql -p $NEWPORT -f dump.sql > output.txt 2>&1`
- 4 Verify system manually, check `output.txt`, test application
- 5 Stop old version of PostgreSQL
- 6 Stop new version of PostgreSQL
- 7 **Move old configuration to new installation**
- 8 Upgrade software via package manager
- 9 Start new version of PostgreSQL via `init` or service manager for your OS

dump/restore considerations

- Make a backup of your old data directory
- To reduce restore time, modify `postgresql.conf`
- Remember to reset these to production values after the upgrade!
- To do the upgrade, you need at least 2 times the size of your current data directory in free space

Parameters

```
fsync = off
shared_buffers = [1/3 of memory]
wal_buffers = 1MB
checkout_timeout = 1h
checkpoint_segments = 300
maintenance_work_mem = [1/3 of memory]
```

Upgrading old version

- Some times you encounter old (or very old) versions
 - 7.4, 7.3, 7.2, 7.1, 7.0
 - 6.5 – get out of here! That's 8 years old!
- Systems running 7.2 and earlier **should be upgraded immediately**
- `pg_dump` 8.2 will have issues with 7.2 and earlier
 - Upgrade to 7.3 first, then to 8.2
 - This actually works!
- Use `adddepend` on the 7.3 intermediate step
 - <http://pgfoundry.org/projects/adddepends/>

Data upgrade procedures: zero downtime

- Use Slony to replicate between the old and new version
- Once synchronisation is finished, cut your application over to the new version
- Zero downtime: use a connection pool, reconfigure it and restart

dump/restore vs. slony

Slony

- More complex
- Upgrade will probably take longer, require more testing
- Not practical for very large systems ($> 1/2$ TB)
- Using it already or want to use it
- Zero/minimal downtime

Dump/restore

- Simpler, less things to go wrong
- May have extended down time

Upgrade gotchas

- Data upgrade procedure must be performed for major version upgrades
- Point in time recovery cannot be used for upgrades
- Be very suspicious of upgrade tools provided by Linux distributions or other sources

The future of upgrades

- Ideally, the project needs to look at in place upgrades

Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration**
- 4 Hardware
- 5 Routine Maintenance
- 6 Replication/high availability

postgresql.conf first pass

- Lots of `postgresql.conf` remain untouched
- For an 8.2 system, check:
- http://www.powerpostgresql.com/Downloads/annotated_conf_80.html

Recommended changes for performance

- `fsync` - make sure it's on!
- `shared_buffers` - up to 1/3 of physical memory
- `work_mem`
- `maintenance_work_mem`
- `wal_buffers` - 1/2 MB to 4 MB

postgresql.conf second pass

- There are going to be errors and the like in production – lets catch them

Recommended changes for logging

- `log_min_error_statement = ERROR`
- `log_min_duration_statement = 1000ms`

Authentication

- There are a lot of options: trust, pam, pam, kerberos, ...
- For applications (say, web server), don't bother with password – use SSL
- PostgreSQL authentication should be your last line of defense – not your first

Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration
- 4 Hardware**
- 5 Routine Maintenance
- 6 Replication/high availability

Hardware configuration: basics

- If you're running an important database, get good hardware!
- Disk redundancy is critical (disks fail more often than anything else)
- These days, software RAID tends to be much better than hardware RAID
- SATA is fine, but the disk failure rate seems to be higher

Hardware configuration: storage

- Buy battery backed controllers, enable write caching
- Put WAL on its own RAID 1 partition
- Put the data directory(s) on RAID 10
- Maximise spindles (and controllers)
- Buy as much memory as you can - ideally more than the size of your data directory

Hardware configuration: storage caveats

- Do NOT mount PostgreSQL storage via NFS
- PostgreSQL can run against a SAN - but connect to it via FC
- Beware of IDE/SATA storage which uses write back caching without battery backing

Hardware configuration: CPUs

- Keep systems to ≤ 8 CPUs (for now)
- Opteron, Xeon and core 2 duo exhibit the best performance (for now)

Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration
- 4 Hardware
- 5 Routine Maintenance**
- 6 Replication/high availability

Vacuuming

- Ensure that a vacuuming procedure is in place
- Examine 8.1's auto vacuum daemon – especially for systems which aren't under heavy load
- Vacuum all databases regularly

Backups

- Disks and other components do fail
- When looking at a backup procedure, consider:
 - How much data can you afford to lose?
 - How long can you afford to be down?

Backing up with pg_dump

Pros

- Proven, well tested, well supported
- But you have all the data in a human friendly format

Cons

- Can take hours to perform
- Can take hours to restore

Backing up with PITR

Pros

- Backup is continuous
- Replay can be made continuous
- Ideal for low down time

Cons

- Takes 1 to 10 seconds per file to process
- Cannot be transferred between architectures (32 bit vs. 64 bit)
- More complex, conceptually

Backup caveats

- Try and get backups off site
- **TEST** your backup/recovery strategy
- Test it regularly

Outline

- 1 Installation
- 2 Upgrades
- 3 Configuration
- 4 Hardware
- 5 Routine Maintenance
- 6 Replication/high availability**

High availability as a backup option

- Handles most disaster scenarios
- Various HA options:
 - Slony
 - Shared SAN + RHCS/Linux HA
 - DRBD
 - Proprietary solutions
- Doesn't address the complete data centre destruction
- Doesn't address recovery of application-level data corruption

Disaster recovery

- Think through failure scenarios
 - Data corruption by a rogue application
 - Hacker
 - Incompetent user/admin
 - Corruption by hardware (faulty CPU, cache, disks, memory)
 - Complete system failure
 - Fire
 - ...
- `pg_filedump` - <http://sources.redhat.com/rhdb/utilities.html>
- Work with the community if you encounter a disaster out of your depth
- Organise commercial support