Workshop

Oracle to Postgres Migration

Part 2 - Running Postgres

2016-06-22 @IDM

Chris Mair

http://www.pgtraining.com

The Workshop

very quick walk through for Postgres-DBAs to-be

- installation, getting support, the configuration files, psql, understanding transactions, the query-planner and locking, backups, system tables, streaming replication, hot standbys, connection pooling, load balancing and even automatic failover all with life-demos and condensed into just three hours - will we finish on time?

Getting Support

- **very** good community support through mailing lists: psql.it list / Italian and official list (English) and many others

- commercial support - in Italy for example from us at PGtraining (three free lancers) or 2ndQuadrant (SRL), in Austria from Cypertec (GmbH) et al

- don't forget managed hosting offerings from Amazon Web Services (PostgreSQL RDS), Heroku and others

Installing Postgres

- from your distro (note that the **second digit** is the **major** version 9.0 and 9.5 are five years apart and some distros carry outdated versions)

- from the official repos at www.postgresql.org/download/ - all major package formats supported

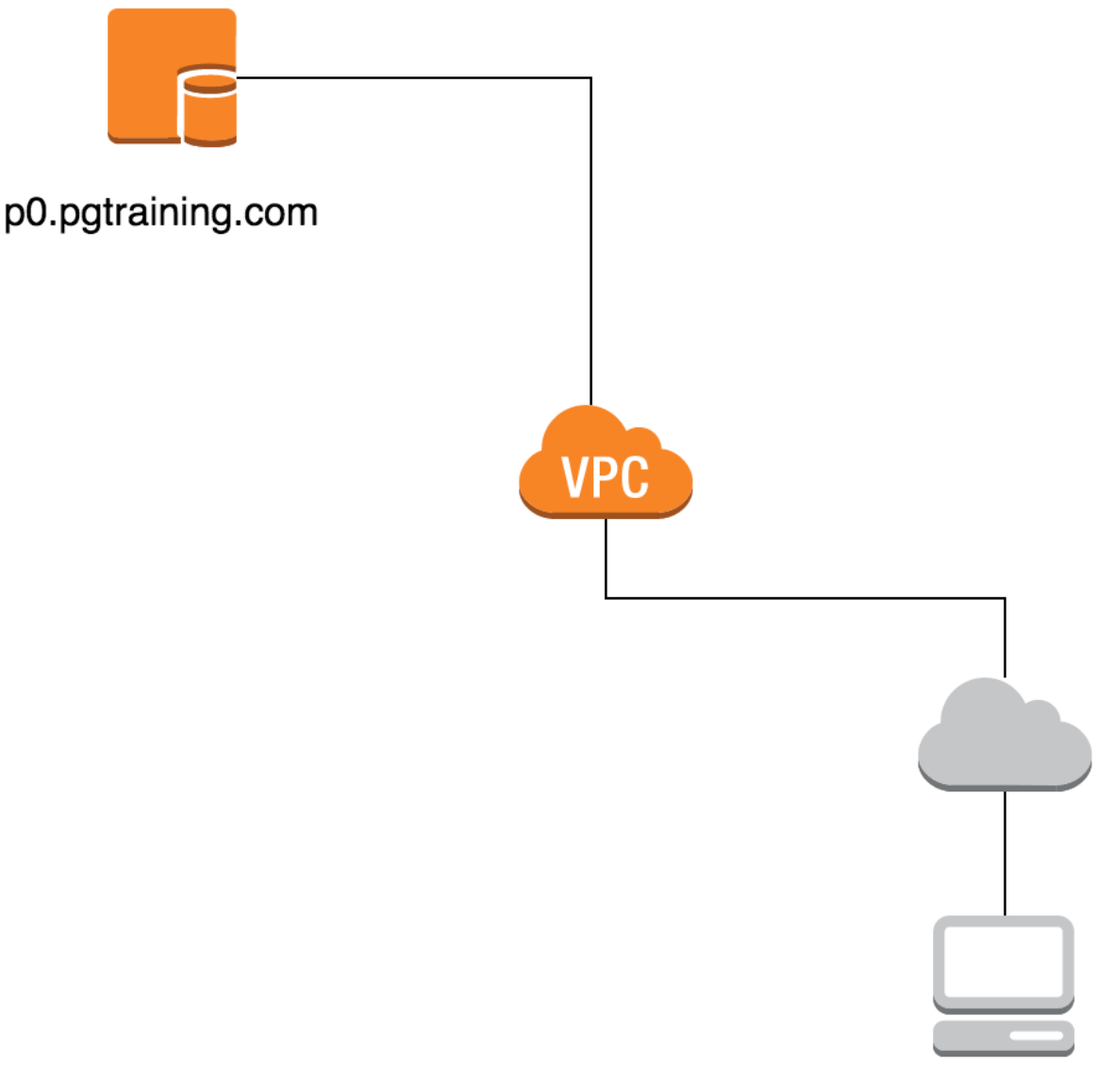
- from source (it is easier than you think: everything can be compiled in a minute or two)

From Source, You Say?

- yeah, why not?

```
# Centos 7

yum -y install wget

yum -y install gcc make zlib zlib-devel libxml2 libxml2-devel \
               readline readline-devel openssl openssl-libs openssl-devel

useradd -m -s /bin/bash pg95

chmod 755 /home/pg95

su - pg95 -c 'wget https://ftp.postgresql.org/pub/source/v9.5.3/postgresql-9.5.3.tar.gz'

su - pg95 -c 'tar xf postgresql-9.5.3.tar.gz'

su - pg95 -c 'cd postgresql-9.5.3; ./configure --prefix=/home/pg95 --with-libxml \
                                               --with-openssl'
su - pg95 -c 'cd postgresql-9.5.3; make -j 2 && make install'
```

Sample Setup (v.1)



Configuration

- use initdb to create the "**cluster**" (as in "instance of postgres serving a set of databases", not as in a set of machines)

```
su - pg95 -c 'bin/initdb -D data'
# instance is fully contained in PGDATA=/home/pg95/data now
```

- configuration is in **`$PGDATA/postgresql.conf`** (at the very least check out listen_addresses, max_connections, shared_buffers and work_mem)

- ACLs are in **`$PGDATA/pg_hba.conf`**

Starting and Connecting

- pg_ctl is your friend (put this line in /etc/rc.local and make it executable):

```
su - pg95 -c 'bin/pg_ctl -D data -l log start'
```

- psql is the universal client:

```
[root@p0-primary ~]# su - pg95
Last login: Wed Jun 22 08:47:36 UTC 2016 on pts/0

[pg95@p0-primary ~]$ bin/psql postgres
psql (9.5.3)
Type "help" for help.

postgres=# \q
[pg95@p0-primary ~]$
```

Psql Sample Session

databases

schemas

tables et.al

\?

```
[root@p0-primary ~]# su - pg95
Last login: Wed Jun 22 08:47:36 UTC 2016 on pts/0
[pg95@p0-primary ~]$ bin/psql postgres
psql (9.5.3)
Type "help" for help.

postgres=# \l
                                  List of databases
   Name    | Owner | Encoding |   Collate   |    Ctype    | Access privileges
-----------+-------+----------+-------------+-------------+-------------------
 postgres  | pg95  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | pg95  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | [...]
 template1 | pg95  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | [...]
(3 rows)

postgres=# \dn
List of schemas
  Name   | Owner
--------+-------
 public | pg95
(1 row)

postgres=# \d
           List of relations
 Schema |    Name     |   Type   | Owner
--------+-------------+----------+-------
 public | tab         | table    | pg95
 public | tab_id_seq | sequence | pg95
(2 rows)
```

One Elephant at Work - understanding transactions

- let's generate a file with single inserts:

```
for (( i=0; i < 50000; i++ )) do
  echo insert into big values \( $RANDOM \) \;
done
```

- and load it into the database:

```
psql postgres -c "drop table big; create table big (x int);"
time psql postgres --quiet < inserts.sql
```

- experiments - what happens if:
  - you add a begin/commit around the inserts?
  - you create an unlogged table?
  - you set synchronous_commit to off?

} outcome will pretty much depend on disk type...

One Elephant at Work - understanding the planner

- let's generate a large table with an index:

```
select random() as x into big from generate_series(1, 1000000);
create index ix on big(x);
```

- and look at the plans for queries such as:

```
explain select count(*) from big where x < 0.00001;
```

- experiment - what happens if:
  - you switch off auto-analyze (parameter autovacuum = off in postgresql.conf), restart the server, drop and recreate the table and repeat the experiment?

One Elephant at Work - understanding MVCC and locking

- thanks to MVCC, "normal" operations such as update/delete/insert do not need to lock a table, you can do a:

```
begin;
update person set name = 'Chris' where id = 1;
-- wait
```

  in one session while the table is fully usable on another session. only if you try to update/delete THE SAME row, will the second session be blocked.

- there are, however, operations that need locks on whole tables, typically I've seen:
  - truncate
  - DDL statements such as ALTER TABLE
- I've seen situations were postgres instances were very "laggy", while the system load was low due to lock contention

Useful System Tables

- pg_stat_activity - list of sessions and what they're doing:

```
select pid, usename, state, query from pg_stat_activity;
```

- pg_locks (beware for example of AccessExclusiveLock locks on user tables):

```
select locktype, database, relation, (select relname from pg_class where
oid = relation), pid, mode from pg_locks;
```

- pg_stat_all_tables - to check among other things auto-analyze is good:

```
select relname, last_analyze, last_autoanalyze  from pg_stat_user_tables;
```

- and many more

Backups

- **cold backups** - just shut the server down and archive the $PGDATA directory
- **online backups** - `pg_dump` or `pg_dumpall`:
  - **`pg_dump`** is per database (or table) with options, for example binary output
  - **`pg_dumpall`** is needed to backup the cluster-wide info such as users
  - **`psql`** and possibly **`pg_restore`** (to read the binary format) are needed to restore the DBs
- demo as time permits

No More Elephants

- Have a look at Josh Berkus' 7 ways to crash Postgres:
  - no updates
  - out of disk space
  - deleting stuff
  - out of RAM
  - bad hardware
  - too many connections
  - zombie locks

More Than One Elephant

- the other meaning of the word "cluster" is somewhat vague - here are some Postgres features that I currently like to use:

- **streaming replication**: stream database operations to other nodes in real time (optionally as 2-safe replication - i.e. at least one slave must have ack'ed a transaction), this can be cascading too

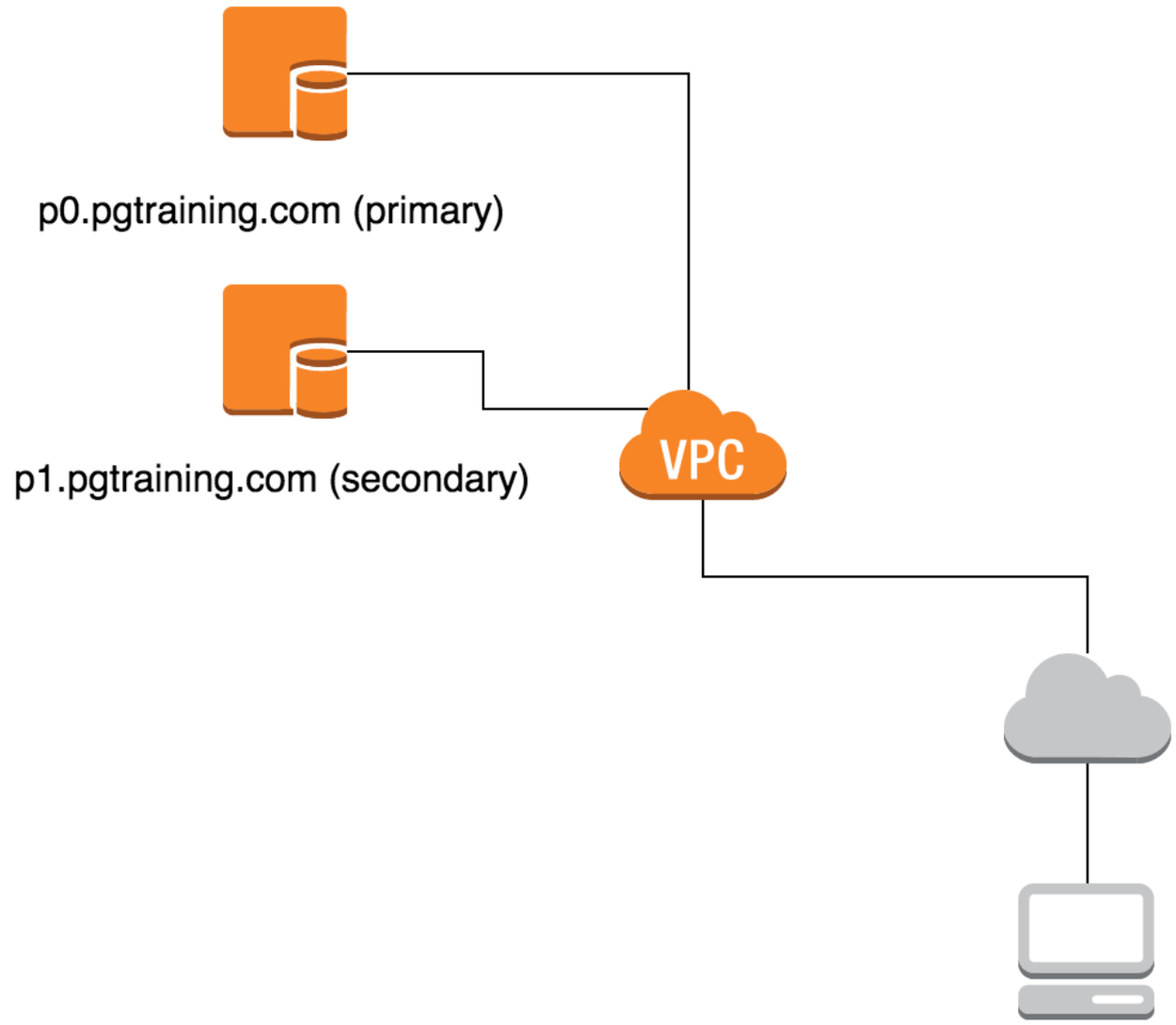
- **hot standby**: issue queries on any secondary node (this includes doing online backups on a secondary to save load from the primary)

- **instant failover**: promote a hot standby node to primary node instantly with a single operation for high availability setups

- third party software allows much more, including master-master setups

- recent developments have much enhanced the streaming capabilities, for example pglogical and BDR - eventually these will be merged into Postgres (see for example my presentation on BDR)

We've Been Doing it the Whole Time ;)



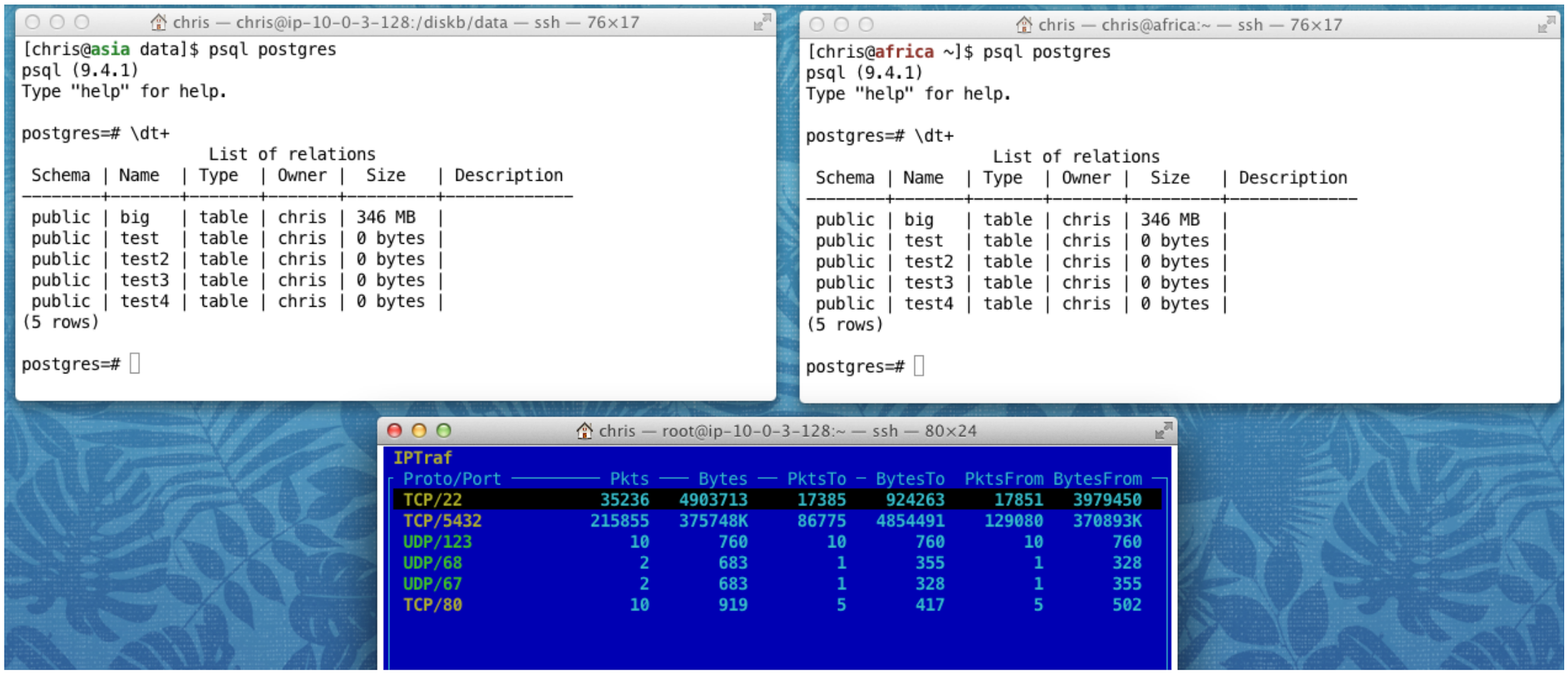
Setting up Streaming Replication with a Hot Standby

- 5 minutes instruction by Cybertec
- our setup scripted for reference:

```
PRIMARY_IP=10.0.1.123
SECONDARY_IP=10.0.1.124

# primary setup
su - pg95 -c 'bin/initdb -D data'
sed -i "s/#listen_addresses = 'localhost'/listen_addresses = '*'/"          /home/pg95/data/postgresql.conf
sed -i "s/#wal_level = minimal/wal_level = hot_standby/"                    /home/pg95/data/postgresql.conf
sed -i "s/#max_wal_senders = 0/max_wal_senders = 3/"                        /home/pg95/data/postgresql.conf
sed -i "s/#wal_keep_segments = 0/wal_keep_segments = 1024/"                 /home/pg95/data/postgresql.conf
sed -i "s/#hot_standby = off/hot_standby = on/"                             /home/pg95/data/postgresql.conf
echo "host replication all $SECONDARY_IP/32 trust" >>                       /home/pg95/data/pg_hba.conf
su - pg95 -c 'bin/pg_ctl -D data -l log start'
# note: use ssl and don't use trust auth in production, also have a look at the feature "replication slots"
# and if you're doing online backups on the standby see 25.5.2. Handling Query Conflicts in the manual

# secondary setup
su - pg95 -c 'mkdir data && chmod 700 data'
su - pg95 -c "bin/pg_basebackup -h $PRIMARY_IP -D /home/pg95/data --xlog-method=stream"
su - pg95 -c "echo 'standby_mode = on'                          > data/recovery.conf"
su - pg95 -c "echo \"primary_conninfo = 'host=$PRIMARY_IP'\" >> data/recovery.conf"
su - pg95 -c "echo \"trigger_file = '/tmp/promoteme'\"       >> data/recovery.conf"
```

Streaming Experiments

- screenshot from another demo (with machines africa and asia):



L'Appetito vien mangiando

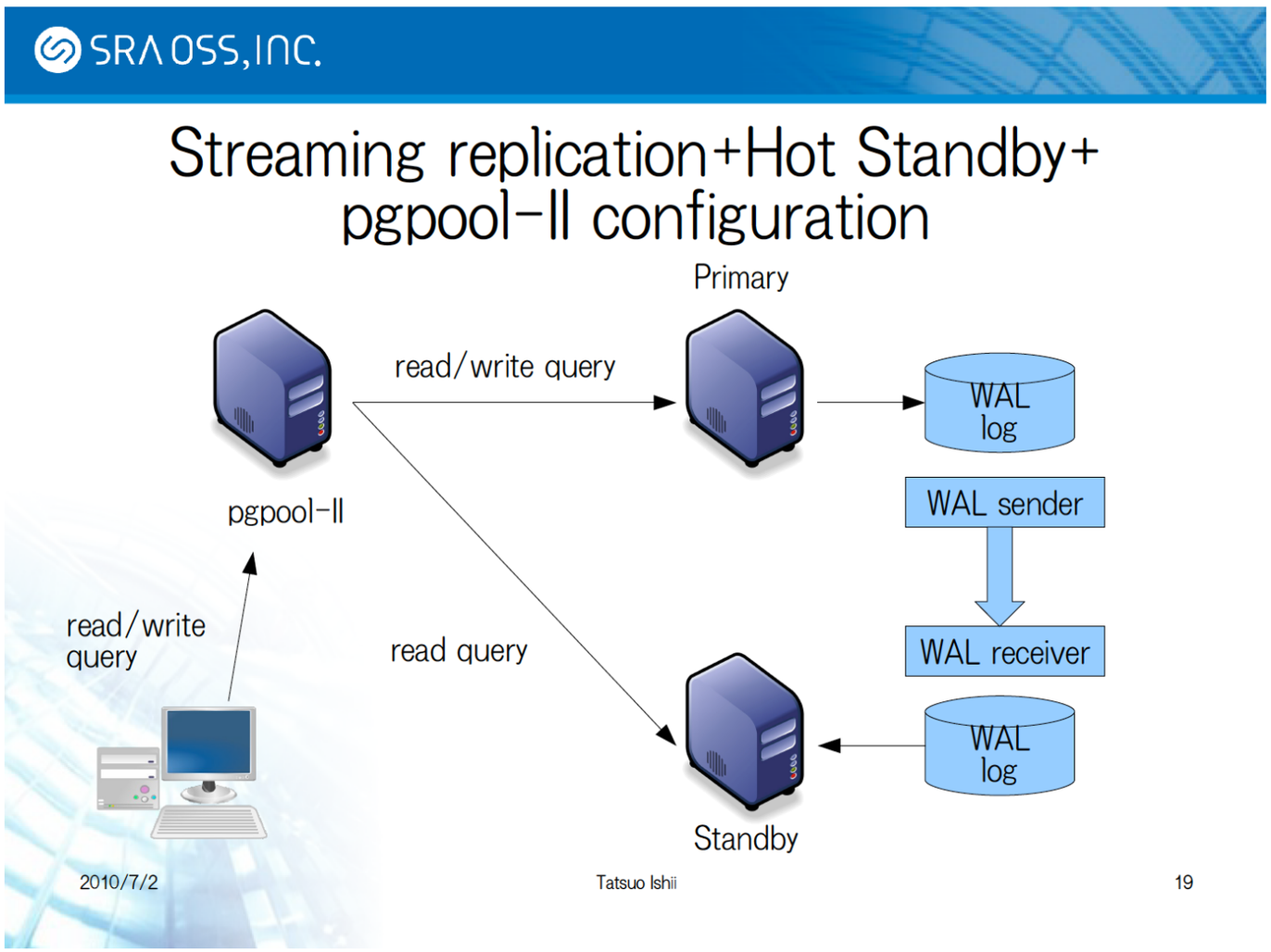
- from the point of view of the application:
  - hey, a connection pool would be handy!
  - mmm.... in case of failover to the standby, how am I notified that I need to change my JDBC URL?
  - come to think of it, it would be cool to off-load read-only queries to the secondary server(s), but I don't want to handle that logic by myself...

Enter pgpool-II

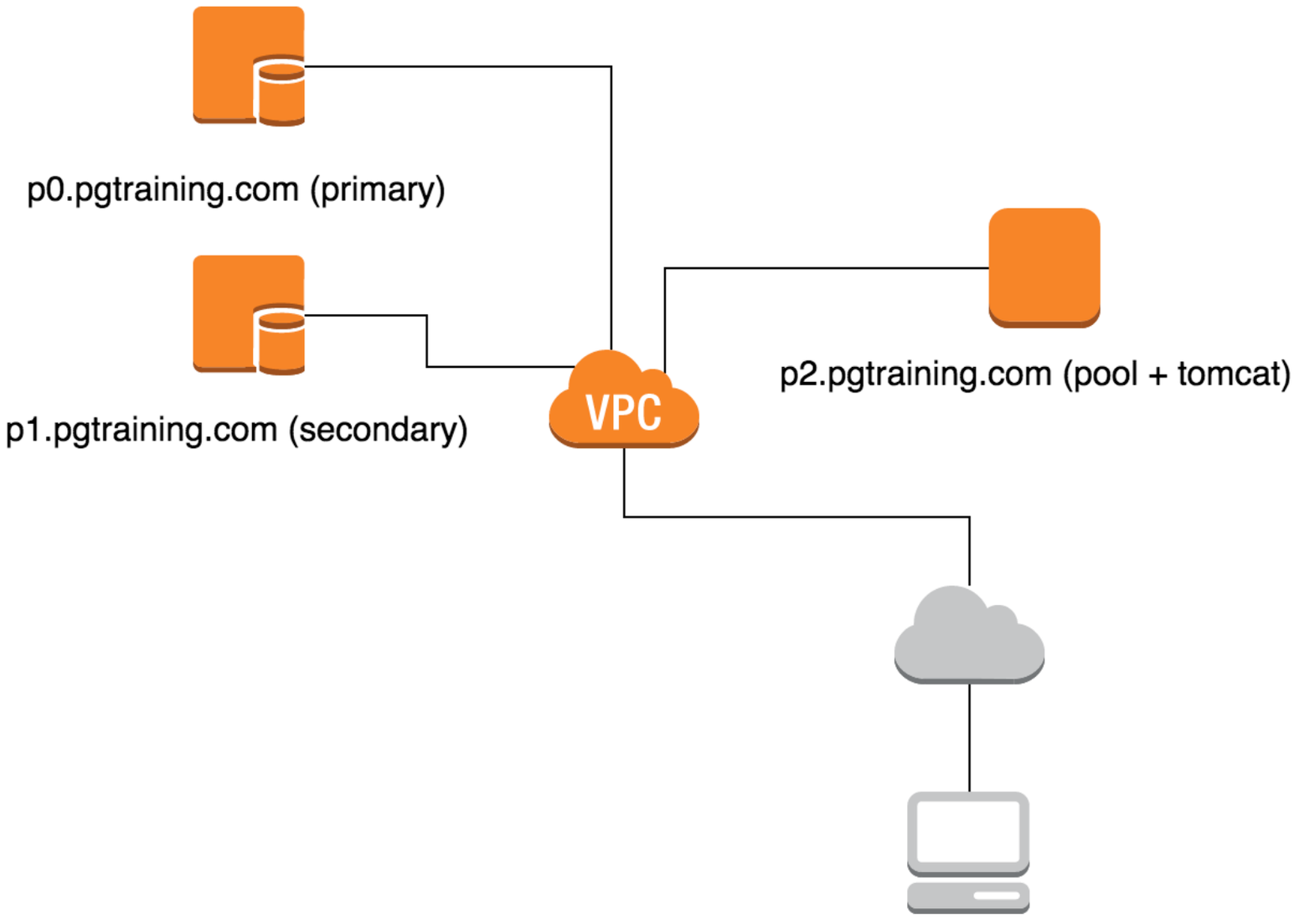
- pgpool-II is a middleware that does exactly this:
  - it hides Postgres servers behind one port 5432
  - it does **connection pooling**
  - it does **load balancing** with the ability to pre-parse queries and send read-only once to the standbys
- and much more:
  - it can do replication by sending the same queries to multiple servers (this is master-master replication even, but it is less efficient and more fragile than doing it with streaming replication)
  - it has a built-in watchdog for high availability setups with two pgool-II servers and virtual IPs
  - etc.

pgpool-II

- here is a pgpool-II presentation from the author of the software - this is what we want to do (from the linked presentation):



The pool is ready!



Experiments

- demo what we have on p2, enable query logging on p0 and p1 to see the load balancing in action, see what happens if p0 or p1 goes down!
- our setup for reference:

```
# note: make a db user nobdody for the monitoring and make a pg_hba.conf entry on p0 and 01 too...

useradd -m -s /bin/bash pgpool
su - pgpool -c 'wget -O pgpool-II-3.5.3.tar.gz http://www.pgpool.net/download.php?f=pgpool-II-3.5.3.tar.gz'
su - pgpool -c 'tar xf pgpool-II-3.5.3.tar.gz'
su - pgpool -c 'cd pgpool-II-3.5.3; ./configure --prefix=/home/pgpool --with-openssl --with-pgsql=/home/pg95'
su - pgpool -c 'cd pgpool-II-3.5.3; make -j 2 && make install'
su - pgpool -c 'cp etc/pgpool.conf.sample-stream etc/pgpool.conf'
su - pgpool -c 'cp etc/pool_hba.conf.sample etc/pool_hba.conf'
su - pgpool -c 'cp etc/pcp.conf.sample etc/pcp.conf'
sed -i "s/^backend_/#backend_/" /home/pgpool/etc/pgpool.conf
sed -i "s/^pid_file_name = '\/var\/run\/pgpool\/pgpool.pid'/pid_file_name = '\/home\/pgpool\/pgpool.pid'/" /home/pgpool/etc/pgpool.conf
sed -i "s/^logdir = '\/tmp'/logdir = '\/home\/pgpool'/" /home/pgpool/etc/pgpool.conf
sed -i "s/^health_check_period = 0/health_check_period = 1/" /home/pgpool/etc/pgpool.conf

echo "backend_hostname0 = '$PRIMARY_IP'"    >> /home/pgpool/etc/pgpool.conf
echo "backend_port0     = 5432"             >> /home/pgpool/etc/pgpool.conf
echo "backend_weight0   = 1"                >> /home/pgpool/etc/pgpool.conf

echo "backend_hostname1 = '$SECONDARY_IP'"  >> /home/pgpool/etc/pgpool.conf
echo "backend_port1     = 5432"             >> /home/pgpool/etc/pgpool.conf
echo "backend_weight1   = 1"                >> /home/pgpool/etc/pgpool.conf

echo "pgpool:d41d8cd98f00b204e9800998ecf8427e" >> /home/pgpool/etc/pcp.conf # empty password

su - pgpool -c 'nohup pgpool -n 2> log &'
```

Failover

- one of the cool features of pgpool-II is that events from nodes attaching/detaching can be scripted

- demo (if time permits) how to instruct pgpool-II to connect to the standby over SSH and touch the trigger file to trigger a promotion to primary

- however, always be aware that automatic failover can be tricky (test well!)

A Simpler Pool

- if you don't need load balancing and automatic failover, I recommend PgBouncer
- PgBouncer is "only" a connection pool, but it does that job **really** well
- you can also combine pgpool-II and PgBouncer

'k thx bye ;)