PostgreSQL 9.4 up and running..

Vasilis Ventirozos
Database Administrator
OmniTI

Who am I ?

DBA@OmniTI for the past 2.5 years

PostgreSQL DBA since 7.x

Working as DBA since 2001
(Informix, Postgres, MySQL, Oracle)

Doesn’t like Oracle !!!

Blog: http://evol-monkey.blogspot.com

Twitter: @vventirozos

Email: vventirozos@omniti.com

Why PostgreSQL ?

- Reliable
- Extensible
- Cross Platform
- Performance
- High Availability
- Development Pace
- $$ COST $$

Use the source Luke !

Why ?

- Customize your own installation, no matter the distribution.
- Install only the necessary.

How ?

Download source

```
wget -c https://ftp.postgresql.org/pub/source/v9.4.5/postgresql-9.4.5.tar.gz
```

Install dependencies

```
sudo apt-get install build-essential
sudo apt-get install libreadline6-dev

sudo apt-get install zlib1g-dev
```

Compile

```
./configure --prefix=/opt/pgsql/9.4.5 --enable-debug

make world

make install-world
```

Make your life easier

```
export PGDATA=/data/pgdata

export PATH=$PATH:/opt/pgsql/9.4.5/bin
```

init script can be found in :

```
postgresql-9.4.5/contrib/start-scripts/linux
```

Contrib packages are very useful:

```
psql -c "create extension pageinspect;" template1
psql -c "create extension pg_stat_statements;" template1
psql -c "create extension pg_buffercache;" template1
```

Getting there..

initdb -D $PGDATA
initializes a new database cluster
(also supports data checksums)

Configuring..part 1..

$PGDATA/pg_hba.conf (host based authentication)

```
local    all                 all                                      trust
host          all                 all                 127.0.0.1/32          trust
host          all                 user            192.168.1.0/16          md5
hostssl       all                 user-ssl        192.168.1.0/16          md5
host          replication   repuser      192.168.1.1/32     md5
hostssl       replication   repuser      192.168.1.2/32     md5
```
host all all 0.0.0.0/0
reject

*(it's last for a reason)

$PGDATA/postgresql.conf, the basics

listen_addresses = '*'
shared_buffers = 25% of your RAM or up to 8GB
maintenance_work_mem = ~1GB
work_mem = depends !
checkpoint_segments = ~100
archive_mode = on
archive_command = '/bin/true'
effective_cache_size = 75% of the ram, not allocated.

protip: work_mem is per operation, not per statement !

Getting ready for replication

```
wal_level = hot_standby
max_wal_senders = 5
max_replication_slots = 5
hot_standby = on
```

Logging is important !

```
log_destination = 'stderr'
logging_collector = on
log_directory = 'pg_log'
log_filename = 'postgresql-%Y-%m-%d_%H%M%S.log'
log_min_duration_statement = 0 (too much spam!!)
log_duration = on
log_line_prefix = '%t [%r] [%p]: [%l-1] user=%u,db=%d,e=%e '
log_statement = 'all'
log_temp_files = 0
```

Ready to start !!

```
pg_ctl -D $PGDATA start
```

(always check logs under $PGDATA/pg_log)

Adding a Streaming Replica

On Master :

```
psql -c "SELECT * FROM pg_create_physical_replication_slot('slave01_slot');" postgres
psql -c "CREATE role repuser with replication login password 'password';" postgres
```

Remember that pg_hba.conf entry ?

```
host replication   repuser        192.168.1.2/32        md5
```

On Slave :

```
pg_receivexlog -D . -S slave01_slot -v -h 192.168.1.1 -U repuser -W
Password:
pg_receivexlog: starting log streaming at 0/3E000000 (timeline 1)
pg_receivexlog: finished segment at 0/3F000000 (timeline 1)
^Cpg_receivexlog: received interrupt signal, exiting
pg_receivexlog: received interrupt signal, exiting
```

PROTIP: This will activate the replica slot ;-)

WTF is a replica slot ?

Introduced in 9.4

remember this? :
requested WAL segment 00000001000000C000000AE has already been removed.

Now forget it :-)

Ready for pg_basebackup

pg_basebackup

- works over the replication stream (no FW changes needed)
- it’s fast !
- can be used for all kinds of backups.
- supports more than one formats and it can use compression.

To create the replica you want to run something like :

```
pg_basebackup -h <master ip> -D $PGDATA -X stream -P -U repuser
```

Last step .. I promise

$PGDATA/recovery.conf

```
standby_mode = 'on'
primary_conninfo = 'host=<MASTER IP> port=5432 user=repuser password=password
application_name=slave01'
trigger_file = '<a path>/failover.trigger'
recovery_target_timeline = 'latest'
primary_slot_name = 'slave01_slot'
```

And finally….

Startup of slave

pg_ctl -D $PGDATA start

Slave will accept read only connections, a nice way to scale reads.

(Always check logs)

Monitoring replication

Monitoring is important amirite ?

=# SELECT application_name,sync_state,pg_xlog_location_diff(pg_current_xlog_insert_location(), flush_location) AS lag_byte FROM ***pg_stat_replication*** order by 1;

```
application_name | sync_state | lag_byte
-----------------+------------+----------
slave01          | async      |        0
```

=# SELECT slot_name, active, pg_xlog_location_diff(pg_current_xlog_insert_location(), restart_lsn) AS retained_bytes FROM ***pg_replication_slots*** order by 1;

```
 slot_name   | active | retained_bytes
-------------+--------+----------------
slave01_slot | t      |              0
```

Synchronous VS Asynchronous

Synchronous means that master has to get a confirmation from one server that transaction made it there before it commits.

```
synchronous_commit = on
synchronous_standby_names = 'server1,server2,server3 ….'
```

Remember..
being paranoid equals delays...

omg noes, Master db died !!

Pre-Failover check list (if you can..)

- Active transactions
- Making sure no one will connect from now on.
- Replication Lag

Failover is easy, failback is not !

From slave : pg_ctl promote, or touch the trigger file
yeah, that’s it...

So, ex-master is now ok ?

A Demoted server is USELESS! (sorry)

- Add as a new slave (rebuild)
- Promote
- Rebuild the new slave again

Backups

But, i have a replica, i should be ok … NOPE!

- User mistakes
- Application errors
- Bugz
- Corruptions

To dump..

pg_dumps

pros

- Small
- Allows partial restore
- Fast (or not)
- can be done from slave

cons

- Frozen in time, no PITR
- Slow (size matters!)
- Has to be repeated

Or not to dump ..

Continuous backups with archiving

pros

- PITR
- It’s incremental
- It’s easy
- ..PITR..

cons

- Needs disk space.. sometimes, A LOT..
- Doesn’t allow partial restore

Other backup tools

pg_basebackup
http://www.postgresql.org/docs/9.4/static/app-pgbasebackup.html
OmniPITR
https://github.com/omniti-labs/omnipitr

protip: backups can run from your slave

Restoring is more important..

Always validate backups by restoring them .-

Reporting

pgbadger

https://github.com/dalibo/pgbadger

tail n mail

https://bucardo.org/wiki/Tail_n_mail

system monitoring

https://github.com/omniti-labs/system_monitoring

Integration

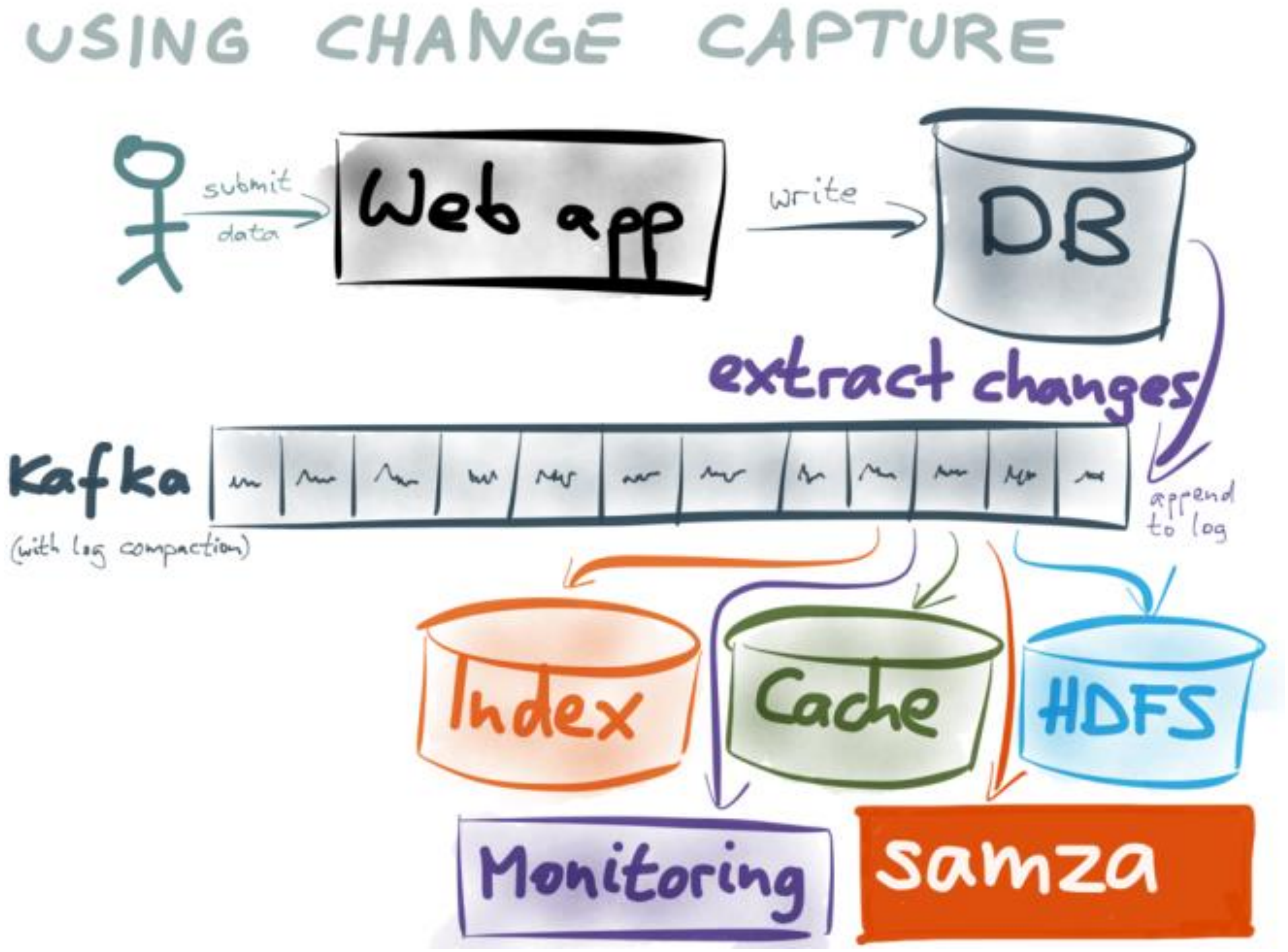
Logical Decoding, introduced in 9.4

- test_decoding – the default plugin
- wal2json – shows the changes in JSON format
- decoder_raw – reconstructs the query that has applied the change.

```
=# SELECT * FROM pg_create_logical_replication_slot('my_slot', 'test_decoding');
 slot_name | xlog_position
-----------+---------------
 my_slot   | 0/16CB0F8
(1 row)

=# UPDATE aa SET c = 3 WHERE (a, b) = (1, 1);
UPDATE 1
=# SELECT * FROM pg_logical_slot_get_changes('my_slot', NULL, NULL);
 location  | xid  |                              data
-----------+------+-----------------------------------------------------------------
 0/1728D50 | 1013 | BEGIN 1013
 0/1728D50 | 1013 | table public.aa: UPDATE: a[integer]:1 b[integer]:1 c[integer]:3
 0/1728E70 | 1013 | COMMIT 1013
(3 rows)
```

Integration example



Questions?