2ndQuadrant
Professional PostgreSQL

Automate High Availability using repmgr 3

Gianni Ciolli

PGConf.EU
Vienna, 27-30 October 2015

repmgr Overview

- Clusterware for PostgreSQL replication
- Open source (GPL)
- Current version: 3.0.2
  - Released on 2 October 2015
- `http://www.repmgr.org/`

Some `repmgr` Features

- Monitoring
- Automatic Failover
- Base Backup with rsync or `pg_basebackup`
- Follow without restart
- Supports Cascading Replication
- Supports Replication Slots
- Event Logging and Commands

Initial Architecture

- We start from here:
  - One Database Server (PostgreSQL)
  - One Backup Server (Barman)
    - Why this one?
    - No Production Without Backup!

Initial Configuration

- barman.conf

```
[haclu]
ssh_command = ssh haclu-primary
conninfo = service=haclu-primary
description = Test HA cluster
```

.

Initial Configuration

- ~barman/.pg_service.conf

```
[haclu-primary]
host=vm1.haclu
user=postgres
```

- ~barman/.ssh/config

```
Host haclu-primary
     HostName 192.168.56.81
     User postgres
```

- Anything depending on state is placed in userspace
  - Our choice (good practice?)

Introducing `repmgr`

- Create `repmgr.conf`

```
cluster=haclu
node=1
node_name=vm1
conninfo=host=vm1 dbname=repmgr
```

repmgr Usage

```
repmgr master register

repmgr standby clone ...
repmgr standby register
repmgr standby unregister
repmgr standby promote
repmgr standby follow

repmgr witness create

repmgr cluster show
```

One Node

```
postgres@vm1:~$ repmgr master register

postgres@vm1:~$ repmgr cluster show
 Role      | Connection String
 * master  | host=vm1 dbname=repmgr
```

Another Node

```
postgres@vm2:~$ repmgr standby clone -h vm1

postgres@vm2:~$ repmgr standby register

postgres@vm2:~$ repmgr cluster show
 Role      | Connection String
 * master  | host=vm1 dbname=repmgr
   standby | host=vm2 dbname=repmgr
```

What about Barman?

- Barman only needs the primary
- Standbys are exact clones of the primary
- Several copies of one database server

Introducing PgBouncer

- Connection Pooling
- Connection Concentration
- Free Software
- .. and much more!

PgBouncer Databases

- PgBouncer defines one or more *databases*
- Each PgBouncer database is a *connection string*
  - Local or Remote
- Clients connect to PgBouncer and are rerouted

PgBouncer Database Conf

- Our choice: separate reads and writes
  - Good practice

- `pgbouncer.ini` on `vm1`

```
[databases]
postgres_rw = host=vm1 dbname=postgres
postgres_ro = host=vm1 dbname=postgres
```

- `pgbouncer.ini` on `vm2`

```
[databases]
postgres_rw = host=vm1 dbname=postgres
postgres_ro = host=vm2 dbname=postgres
```

`repmgr` Automation

- Daemon `repmgrd`
  - Automatic Failover
  - Monitoring
- Extra automation:
  - When the state changes:
    reconfigure what needs to be reconfigured

Automatic Failover

```
failover=automatic
master_response_timeout=20
reconnect_attempts=3
reconnect_interval=5
promote_command=repmgr standby promote
follow_command=repmgr standby follow -W
```

- Can define node priority
  - Promote only if positive

Cluster State?

- A *standby* can replace the *master*
  - That’s what "stand by" means. . .
- Two different terms:
  - Switchover: planned
  - Failover: unplanned
- Crucial difference!
- The state of the cluster:
  - List of nodes
  - Which node is the master

New Primary via Switchover

```
postgres@vm1:~$ pg_ctl shutdown

postgres@vm2:~$ repmgr standby promote



postgres@vm3:~$ repmgr standby follow
postgres@vm4:~$ repmgr standby follow
...
postgres@vm100:~$ repmgr standby follow
```

Switchover Wishlist

- `repmgr standby switchover`
- That would be all!

Cluster State Change

- When the state changes:
  - We must update part of the configuration
- All in userspace:
  - `~barman/.ssh/config`
  - `~barman/.pg_service.conf`
- Well, almost. . .
- Not in userspace:
  - `/etc/pgbouncer/pgbouncer.ini`

Event Notification Commands

- Add to `repmgr.conf` (only two lines):

```
event_notification_command =
  repmgr-agent.sh repmgr.conf
  barman-server %n %e %s

event_notifications =
  master_register, standby_register,
  standby_promote
```

- Run a custom script in occasion of cluster events
  - A bit like AFTER triggers
- Only those that *change the status*

`repmgr-agent.sh`

- Script that updates the configuration
- Idempotent
- Prototype, to be contributed to `repmgr`
- Reads the cluster state
  - From any node in the cluster
- Rewrites:
  - `~barman/.ssh/config`
  - `~barman/.pg_service.conf`
  - `/etc/pgbouncer/pgbouncer.ini`

Integrating repmgr

- Integrating with
  - Barman
  - PgBouncer
  - …

Integration with Barman

- Barman: Disaster Recovery
- Extent of Integration
  - Compatibility
    - `repmgr` does not break Barman **OK**
    - Barman does not break `repmgr` **OK**
  - Symbiosis
    - Barman helps `repmgr` **TODO**
    - `repmgr` helps Barman **???**

Barman helps `repmgr` #1 (TODO)

- `repmgr standby clone`
- Needs a backup
- Currently performs one
- Could reuse a Barman backup!
  - No overhead on primary
  - Cannot fail (backup has happened *already*)
- Barman as a Base Backup Provider

Barman helps `repmgr` #1 (TODO)

- UI Study / Hypothetical HOWTO
- Feature name: *Backup From Barman*
- Must tell `repmgr` where Barman is
- Add to `repmgr.conf`:
  - `barman_ssh_command = ssh barman@backup`
  - `barman_server_name = main`
- Deactivate with command line option `--without-barman`

Barman helps `repmgr` #2 (TODO)

- `restore_command` from Barman
- Uses `barman get-wal`
- Replaces:
  - `wal_keep_segments`
  - Replication slots
  - `archive_cleanup_command`
- Barman as an Archive
  - Very deep
  - Retention policies
  - File Compression

Barman helps `repmgr` #2 (TODO)

- UI Study / Hypothetical HOWTO
- Feature name: *Archive From Barman*
- Same interface as *Backup From Barman*
  - Configure in `repmgr.conf`
  - Disable with `--without-barman`
    - No need to disable it. . .
- IJW
  - **It Just Works!**

And now. . .

Questions?

And then…

Thank you!

`postgresql.eu/events/feedback/pgconfeu2015`

Licence