High Availability and Automatic Failover in PostgreSQL using Open Source Solutions

PostgreSQL High Availability

Avinash Vallarapu (Avi)

PostgreSQL Support Tech Lead
PGCONF India, 2019
15th Jan, 2019

PERCONA

What is High Availability for database servers ?

- **High Availability in our routine database life.**
  - An always-on mechanism
  - Avoid data loss during disasters
  - Higher uptime for business
  - An immediate action upon a detection of failure (but not minutes or days)
  - Avoiding a single point of failure
  - Decrease or minimize the Unscheduled downtime
  - Seamless database fail-overs for Application and Business
  - Ability to perform both manual and automatic failover
  - Faster Point-in-time-recovery (PITR)

PERCONA

PostgreSQL Replication

- **Streaming Replication in PostgreSQL**
  - WAL Segments are streamed to Standby/Slave and replayed on Slave.
  - Not a Statement/Row/Mixed Replication like MySQL.
  - This can be referred to as a byte-by-byte or Storage Level Replication
  - Slaves are always Open for Read-Only SQLs but not Writes
  - You cannot have different Schema or data in a Master and a Slave in Streaming Replication.
  - Allows Cascading Replication
  - Supports both Synchronous and Asynchronous Replication
  - Supports a Delayed Standby for faster PITR

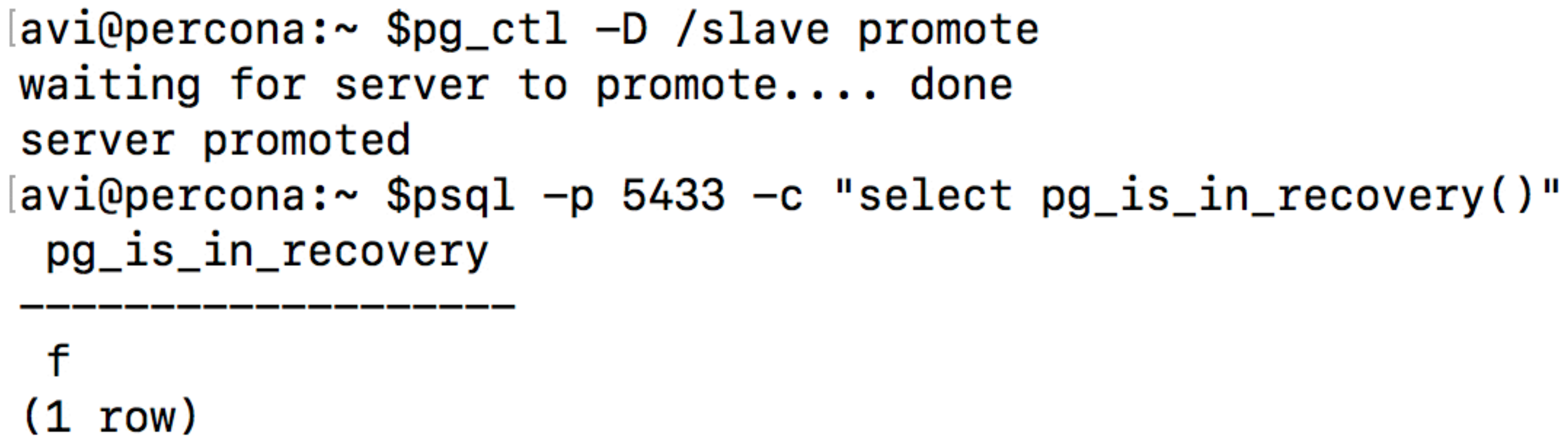
PostgreSQL Replication

- **Logical Replication and Logical Decoding for PostgreSQL 10 and above**
  - Allows for Replication of selected Tables using Publisher and Subscriber Model.
  - Similar to binlog_do_db in MySQL, but no DDL Changes are replicated.
  - Subscribers are also open for Writes automatically
  - Used in Data Warehouse environments that stores Data fetched from multiple OLTP Databases for Reporting, etc.
  - A friendly solution for Database Upgrades

PostgreSQL features and extensions for HA and Automatic failover

- Minimize data loss using **Synchronous Replication** in PostgreSQL.
- May reduce data loss on failover during huge replication lag using the **Archiving** feature in PostgreSQL.
- Faster and easy failover using **promote** or **trigger_file.**
- Faster catch-up of old Master using the extension **pg_rewind**.
- Re-direct READS and REPORTING jobs to a Slave using **hot_standby**.
- Allow long running reporting jobs on Slave to succeed upon changes on Master, using **hot_standby_feedback**, **max_standby_streaming_delay** and **max_standby_archive_delay.**
- Achieve flashback like Oracle features using **recovery_min_apply_delay** on Slave.

Manual Failover using promote



```
[avi@percona:~ $pg_ctl -D /slave promote
waiting for server to promote.... done
server promoted
[avi@percona:~ $psql -p 5433 -c "select pg_is_in_recovery()"
 pg_is_in_recovery
-------------------
 f
(1 row)
```

Manual Failover using trigger_file

- **Using trigger_file**

```
avi@percona:~ $grep "trigger_file" /slave/recovery.conf
trigger_file = '/tmp/failover'
avi@percona:~ $
avi@percona:~ $touch /tmp/failover
avi@percona:~ $psql -p 5433 -c "select pg_is_in_recovery()"
 pg_is_in_recovery
-------------------
 f
(1 row)
```

```
2018-10-31 15:55:42.313 EDT [7926] LOG:  started streaming WAL from primary at 0/E000000 on timeline 1
2018-10-31 15:57:32.498 EDT [7922] LOG:  trigger file found: /tmp/failover
2018-10-31 15:57:32.498 EDT [7926] FATAL:  terminating walreceiver process due to administrator command
2018-10-31 15:57:32.500 EDT [7922] LOG:  invalid record length at 0/E001FA8: wanted 24, got 0
2018-10-31 15:57:32.500 EDT [7922] LOG:  redo done at 0/E001F70
2018-10-31 15:57:32.508 EDT [7922] LOG:  selected new timeline ID: 2
2018-10-31 15:57:32.558 EDT [7922] LOG:  archive recovery complete
2018-10-31 15:57:32.565 EDT [7920] LOG:  database system is ready to accept connections
```

PERCONA

Open Source Solutions for Automatic Failover in PostgreSQL

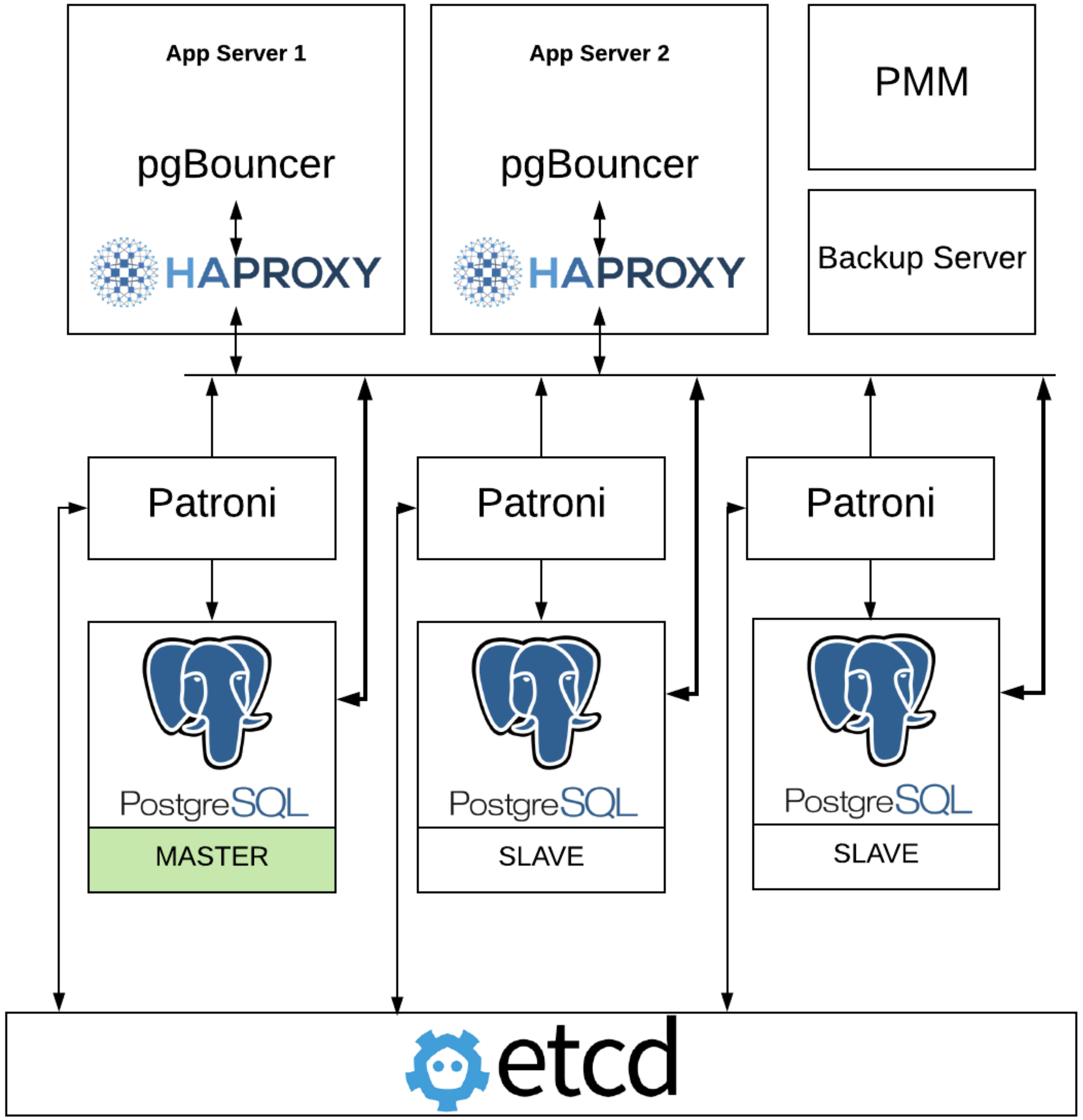
- **List of few Open Source projects for HA and Automatic Failover**
  - Patroni
  - Stolon
  - repmgr
  - PostgreSQL Automatic Failover (PAF)
  - pglookout
  - pgPool-II

Lets discuss a few of the most widely discussed tools

Patroni

- **Patroni**
  - Fork of Governor
  - PostgreSQL cluster management template/framework
  - Talks to a distributed consensus key-value store to decide the state of the Cluster
  - Distributed consensus can be obtained using etcd, ZooKeeper, Consul, etc for electing a leader.
  - Continuous monitoring and automatic failover
  - Built-in automation for bringing back a failed node to cluster.
  - REST APIs for cluster configuration and further tooling.
  - Provides infrastructure for transparent application failover
  - Distributed consensus for every action and configuration
  - Integration with Linux watchdog for avoiding split-brain syndrome.
  - Supports both manual and automatic failover

PERCONA

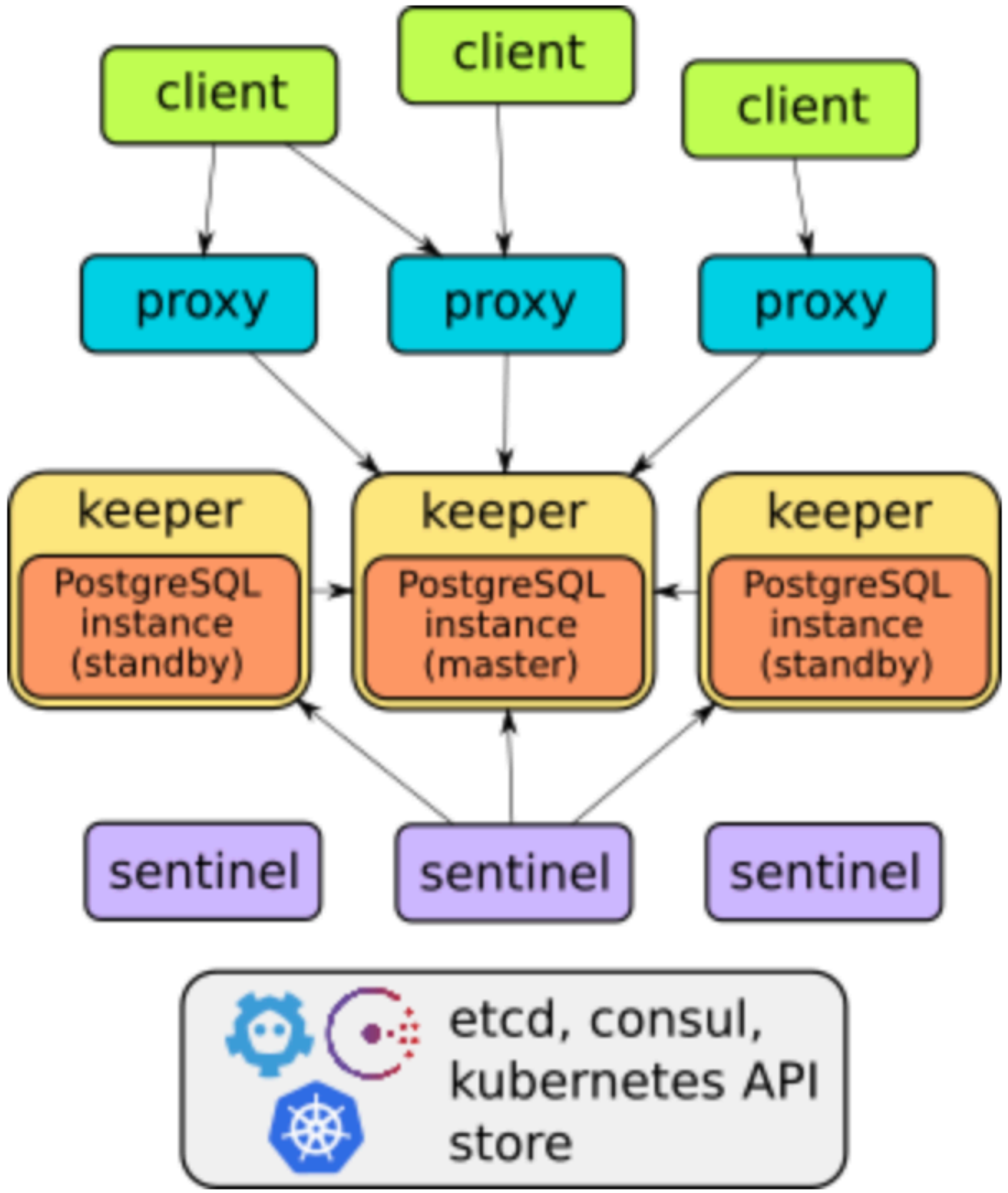

App Server 1
pgBouncer
HAPROXY
App Server 2
pgBouncer
HAPROXY
PMM
Backup Server
Patroni
Patroni
Patroni
PostgreSQL
MASTER
PostgreSQL
SLAVE
PostgreSQL
SLAVE
etcd

REPMGR

- **REPMGR**
  - Uses repmgrd installed in each node for management and monitoring
  - Supports both manual and automatic failover
  - Supports configuring a Witness server to avoid split brain scenario.
  - Provides a view : **replication_status** for monitoring and history of replication lag and node status.
  - Supports over 18 user-friendly commands to perform actions such as :
    - Cloning a Master/Primary
    - Switchover to promote a standby and demote the master
    - Rejoining a node to cluster
    - Promote to promote a standby
    - check node status
    - primary/standby register and unregister
  - Supports executing custom scripts upon automatic failover using **promote_command** and **follow_command**.

Stolon

- **Stolon**
  - Cloud-native HA solution that supports PostgreSQL cluster inside Kubernetes, IaaS and VMs.
  - Uses etcd, consul or Kubernetes API server for distributed consensus.
  - Composed of 3 components :
    - **keeper** : Maintains a cluster view as provided by sentinel(s).
    - **sentinel** : Monitors keepers and builds the cluster view
    - **proxy** : Re-directs connects to Master always for a seamless Application failover.
  - Built on top of PostgreSQL Streaming replication - Synchronous and Asynchronous
  - Supports command line client - **stolonctl** and **kubectl** to perform actions such as :
    - Initialize a cluster
    - Promoting a standby
    - check status


client
client
client
proxy
proxy
proxy
keeper
PostgreSQL instance (standby)
keeper
PostgreSQL instance (master)
keeper
PostgreSQL instance (standby)
sentinel
sentinel
sentinel
etcd, consul, kubernetes API store

pgPool-II

- **pgPool-II**
  - Supports Connection Pooling
  - Manages Replication
  - Load Balancing of Reads and Writes
  - Parses SQLs to determine if it is a read or write
  - Ability to configure weights to balance reads between master and slave
  - Supports Automatic Failover
  - Connections exceeding the max_connections are queued on pgPool-II without rejecting them.
  - Must use Active-Passive pgPool setup for high availability

Points to Remember

- Make sure to test the tool you use for automatic failover.
- Ensure to have a good backup strategy that helps you manage panic situations.
- Be prepared for a data loss and build the ability to manage it from the application.
- The architecture of your HA solution depends on your environment.
- Build the ability to distinguish reads and writes in the application layer for better scalability.
- Perform routine disaster recovery drills through a manual failover to ensure that the setup is reliable.
- Ensure to monitor for patches and perform updates of your PostgreSQL and the HA solution.

Champions of Unbiased
Open Source Database Solutions

Questions ??