High Availability for Postgres using OpenSource tools

By Jobin Augustine & HariKrishna

Introduction

Jobin Augustine

Designation : Architect

Email: jobin.augustine@openscg.com

Contact No: + 91 9989932600

HariKrishna

Designation : Associate Architect

Email: hari.krishna@openscg.com

Contact No: + 91 9701027722

What is High Availability(HA)

High availability is a characteristic of a system, which aims to ensure an agreed level of operational performance, usually uptime, for a higher than normal period.

Service Level Agreements(SLA):

Availability %	Downtime Per Year	Downtime Per Month	Downtime Per Week	Downtime Per Day
90% ("one nine")	36.5 days	72 hours	16.8 hours	2.4 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours	14.4 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes	1.44 minutes
99.99% ("four nines")	52.56 minutes	4.38 minutes	1.01 minutes	8.64 seconds
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds	864.3 ms

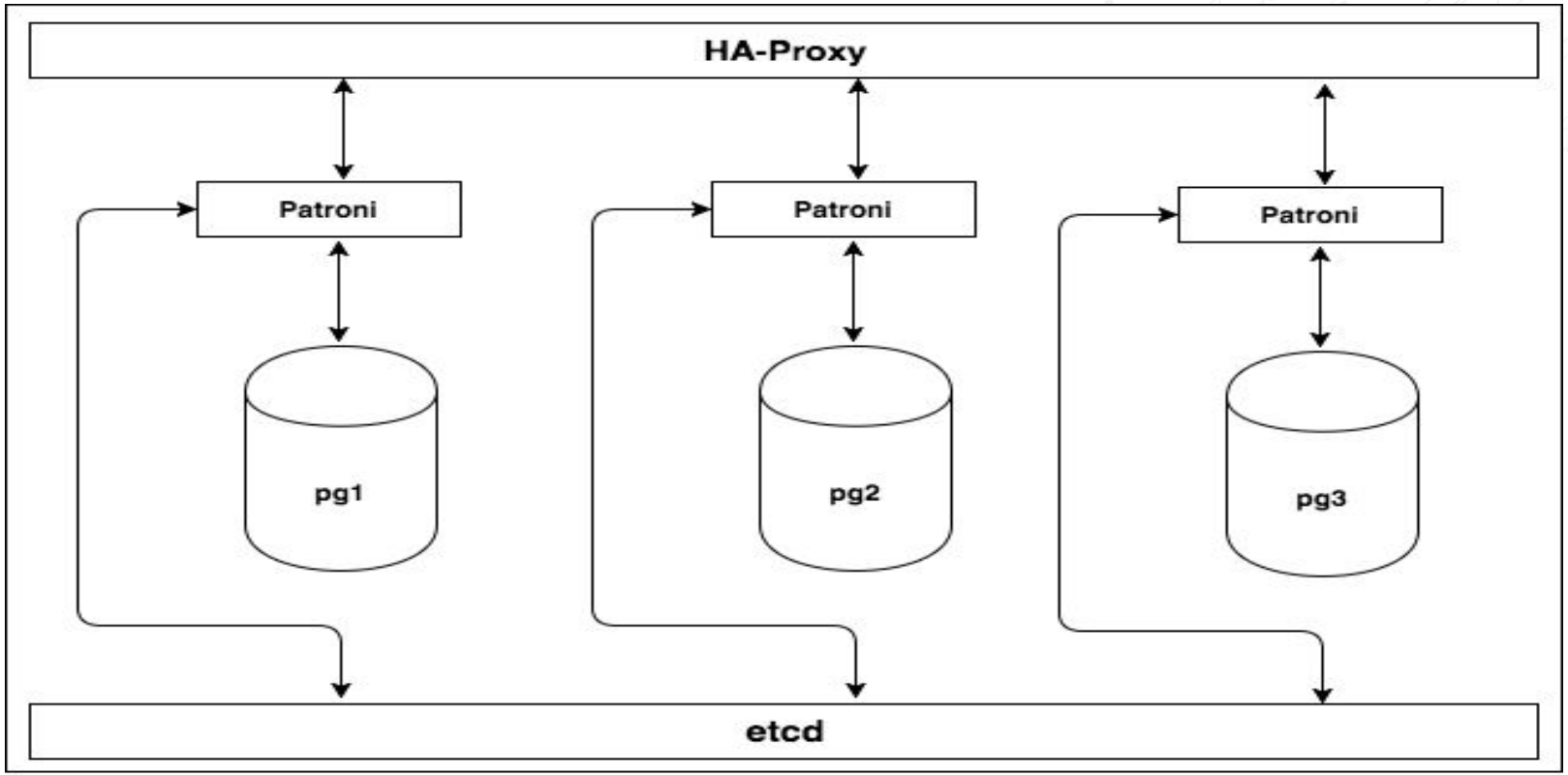
HA Challenges

- Master election and restructure of cluster
- Protection from Split-Brain
- Rejoining of a failed node to the cluster
- Add new nodes - Scale up
- Prevent false Failover
- Protection against Race Conditions and conflicts
- Automatic and Transparent Application connections failover
- Standard components and easy setup

Agenda

- Demo
- HA Architecture- Case study of Patroni
- Components of HA
- RAFT Algorithm
- Q & A

HA Architecture- Patroni

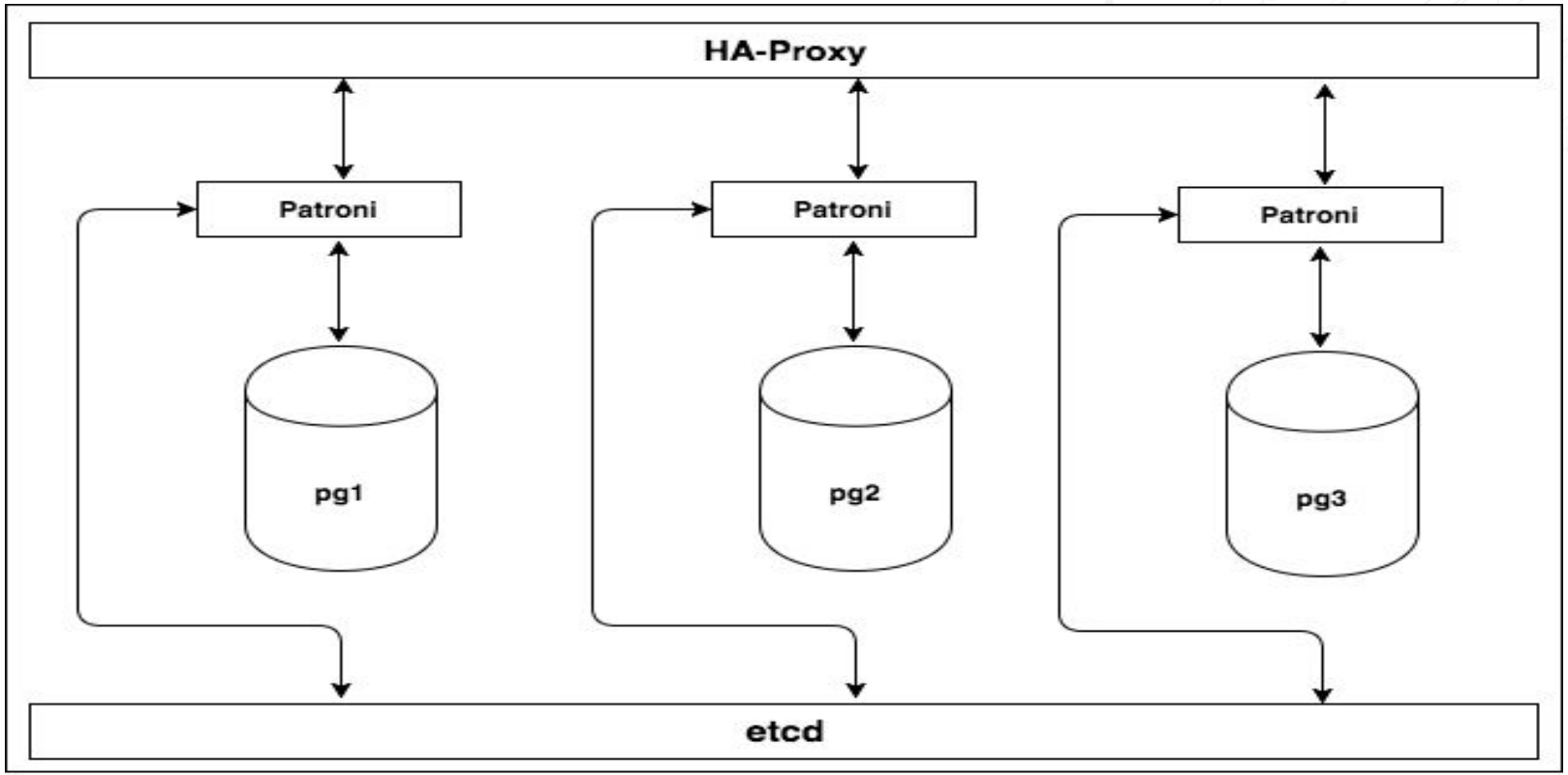


OpenSCG BIGSQL

Demo - Infrastructure details

Server Name	IP	Role	Components Installed
pg0	192.168.50.90	Primary	Etcd, Patroni, PostgreSQL 9.6
pg1	192.168.50.95	Replica	Etcd , Patroni, PostgreSQL 9.6
HA-Proxy	192.168.50.100	Replica/ Middle Tier	Etcd, Patroni, PostgreSQL 9.6 , HA-Proxy, Custom Scripts

HA Architecture- Patroni



OpenSCG BIGSQL

Features of HA-Patroni

- Self healed and self monitored
- Manual & Scheduled Switchover
- Automatic Failover
- Rejoining of old Master with pg_rewind
- Customizable replica creation methods
- Dynamic Configurations
- Pause(Maintenance Mode)
- Transparent Application connection failover

Why do we need Distributed Consensus?

Consensus requires agreement among a number of Process(or agents) for a single data value.

- Avoid Single point of decisions and failures
- To have a Leader(Master) election
- Need to get consensus from members for any action.
- No leader - which nodes the master key
- Leader is present/disappeared -should be the same for all nodes

Components of HA

- etcd
- Patroni
- Watchdog
- HA-Proxy

etcd

- Is a distributed key value store
- It gracefully handles leader elections
- Consistency guarantees to implement database leader
- Implements RAFT
- Talks REST
- Fast : benchmarked 10,000 writes/sec
- Key expiration with TTL and test and set operations

RAFT

- Distributed Consensus Algorithm
- Achieves Consensus by directing all changes to the leader
- Only commit the change if its acknowledge by majority of nodes
- 2 stages
  - Leader election
  - Log Replication
- Implemented in etcd,consul

RAFT Algorithm visualizations ...

- Log Replication
- Leader Election

Patroni

- Manages a Single PostgreSQL Node
- Runs on the same host as PostgreSQL
- Uses etcd for data store
- Promotes/demotes the managed node depending on the leader key

Patroni Configuration Parameters

- YAML Configuration
- General Parameters
  - Ttl
  - Loop_wait
  - Retry_timeout

- PostgreSQL Section
  - Name :
  - Listen :
  - connect_addr :
  - Use_slots :
  - Pg_rewind :
  - Maximum_lag_on_failover:
  - Create_replica_methods
  - Recovery.conf

REST API

REST API endpoint for working with dynamic configuration.

- **GET/Config**- Get current version of dynamic configuration.
- **PATCH /Config**- Change existing configuration
- **PUT /Config** - To perform the full rewrite of an existing dynamic configuration unconditionally

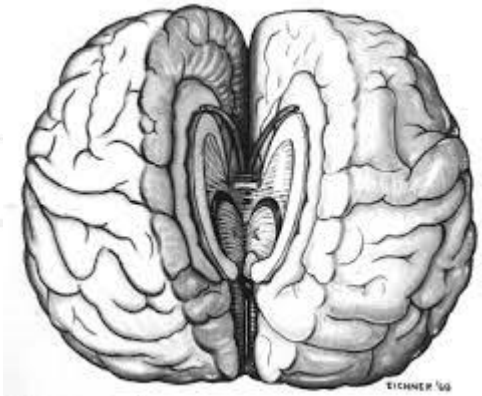
Watchdog

Split-Brain triggered by

- Temporary hangs.
- Temporary network disruptions
- Network partitioning

Ends up in more than 1 master node

Anything that can go wrong, will go wrong



Watchdog

- Hardware + Software mechanism
- Reset the whole system.
- Expects heartbeat within a specified timeframe.
- Standard Interface provided by Linux
- Oracle Node eviction (hangcheck timer)

- Very lightweight and fast
- load-balancer + Router
- TCP Layer 4 routing.
- Rest API of patroni - No custom scripts.
- Transparent redirection of application connection.
- No dynamic config changes.

Caveats

- Minimal Data loss for **async** replication
- verifies **maximum_lag_on_failover** before failover
- verifies watchdog before master promotion
- Non-superuser for Application connection is adviced.
  - avoid hitting on **superuser_reserved_connections**

References

Patroni :

https://github.com/zalando/patroni

Etcd :

https://github.com/coreos/etcd

RAFT :

https://raft.github.io/

HA-Proxy :

http://git.haproxy.org/?p=haproxy-1.8.git

Q & A

Thank You!