Patroni Deployment Patterns

Michael Banck <michael.banck@netapp.com>
Tech Lead Databases, NetApp Deutschland GmbH

SCaLE 21x, 2024-03-14

© Instaclustr Pty Limited, 2024
Introduction

• High-Availability is complicated
  • False positive failovers can reduce availability

• In-Core Postgres only provides replication primitives
  • Streaming replication, asynchronous or synchronous
  • Standby cloning
  • Hot standby
  • Promoting standbys to primaries
  • Rewinding outdated former primaries to new standbys

• Automatic failover requires external support/implementations
  • Failover/switchover orchestration
  • Leader election
  • Split-brain avoidance
  • Quorum enforcement
  • Fencing of faulty nodes
  • Service failover
Introduction

Prior Art

• Pacemaker / Postgres Automatic Failover (PAF)
  • Uses corosync for node inter-communication
  • Relies on STONITH for split-brain avoidance
  • PAF provides improved resource agent
  • Switchovers non-trivial, timeouts difficult to tune
  • Storage switchover possible

• repmgr
  • Good solution for manual management of replication
  • repmgrd provides automatic failover
  • Originally not intended for this, somewhat brittle
  • Upstream commitment unclear

• pg_auto_failover
  • Relatively new project, by Citus/Microsoft (now independent)
  • Thorough state-machine, monitor node
  • Future roadmap unclear, feature-complete?
Patroni Overview
Patroni Overview

- Cloud-Native project/template for PostgreSQL high availability
- Based on Compose Governor (https://github.com/compose/governor)
- Written in Python
- MIT Licence (similar to Postgres licence)
- Project run by Zalando (https://github.com/zalando/patroni)
  - Current project co-maintainer works at Zalando
  - Project founder / project co-maintainer now works at Microsoft
  - Project co-founder / former co-maintainer now works at Timescale
Patroni Overview

Major Features

• Agent, configures instances and replication, enables switchover (Bot-Pattern)
• Uses a distributed configuration store (DCS) for leader election and split-brain avoidance
• Offers a REST-API for status, health checks and configuration changes
• Prometheus metrics
• Citus support
• Optional HAProxy integration for master/replica service endpoints
  • HTTP check REST-API on /master and /replica, respectively
• Optional vip-manager for VIP service endpoint management
  • Checks status of local node in DCS and (de-)configures VIP accordingly
Patroni Overview

Deployment Options

• Containers
  • Spilo, https://github.com/zalando/spilo
  • Crunchy Container Suite, https://github.com/CrunchyData/crunchy-containers
  • CYBERTEC-pg-container, https://github.com/cybertec-postgresql/CYBERTEC-pg-container

• Kubernetes Operators
  • Zalando Operator, https://github.com/zalando/postgres-operator
  • Cybertec Operator, https://github.com/cybertec-postgresql/CYBERTEC-pg-operator
  • Crunchy PGO Postgres Operator, https://github.com/CrunchyData/postgres-operator
  • Percona Operator for PostgreSQL, https://github.com/percona/percona-postgresql-operator
  • OnGres StackGres, https://github.com/ongres/stackgres

• Bare-Metal
  • Python pip3 install patroni[etcd]

• Linux distribution packages
  • <apt.postgresql.org>, <yum.postgresql.org>
  • https://www.credativ.de/en/blog/howtos/integrating-patroni-into-debian/
Patroni Architecture Overview

- Distributed Configuration Store (DCS)
  - Key-value store of cluster configuration
  - Single source of truth

- Patroni Service
  - Manages local PostgreSQL node
  - REST API (monitoring/management)

- PostgreSQL
  - Streaming replication between nodes

- Routing
  - Via middleware (proxy/pooler)
  - Via virtual IP
  - Via client-based failover
Distributed Configuration Store

- RAFT consensus algorithm
- Distributed key-value store
- Key changes done via atomic CAS (compare and swap) operations
- Automatically expiring keys (TTL, watches)
- Implementations:
  - etcd (v2/v3)
  - Consul
  - Zookeeper
  - Kubernetes API
  - PySyncObj (deprecated)
Split-Brain Avoidance

- Realised through quorum via DCS
- Primary periodically updates the leader key in DCS with a TTL
- Replicas watch validity of leader key
- Leader race initiated when leader key expires
- Fencing of problematic nodes
  - Primary demotes itself to standby when DCS is not reachable, failsafe mode is not active and leader key expires
  - Watchdog device can shutdown nodes in case they no longer react
Patroni Loops/Timeouts

- **TTL**
  - Time in seconds a DCS key is valid after update

- **Loop Wait**
  - Time in seconds between DCS updates

- **Retry Timeout**
  - Time in seconds that is waited (twice) in case DCS is not reachable

```plaintext
ttl: 30
loop_wait: 10
retry_timeout: 10
```
Leader Race

Leader Updates Leader Key
Leader Race

Leader Goes Down
Leader Race

Leader Key Expires
Leader Race

Followers Query Old Leader and Other Nodes
Leader Race

Followers Race to Acquire Leader Key
Leader Race

New Leader Elected
Patroni Operation Concepts
**patronictl**

- Command-Line tool to manage Patroni / Patroni clusters
- Talks to Patroni’s REST API
- Can start/stop nodes, initiate switchovers, planned restarts
- Cluster configuration changes (show-config, edit-config)
- Maintenance Mode (pause, resume)

```
root@pg1:~# patronictl list
+ Cluster: pg-test (7346325357177231574) --------------------------+
 | Member | Host   | Role    | State   | TL | Lag in MB |
+--------+--------+---------+---------+----+-----------+
 | pg1    | 10.0.3.236 | Leader  | running | 1  |           |
 | pg2    | 10.0.3.31  | Replica | streaming | 1 | 0         |
 | pg3    | 10.0.3.238 | Replica | streaming | 1 | 0         |
+--------+--------+---------+---------+----+-----------+
```
Client Failover

• If a switch/failover occurs, clients need to connect to new leader

• HAPerxy
  • Can use REST API http response codes for health checks

• vip-manager
  • Polls DCS for state of local node and (de)configures VIP
  • Supports only etcd and consul
  • vip-manager-2.x supports etcd V3 API (etcd3), vip-manager-1.x supports etcd V2 API (etcd)

• Client-based failover
  • libpq host=pg1,pg2,pg3 target_session_attrs=primary
  • pgJDBC jdbc:postgresql://pg1,pg2,pg3?targetServerType=primary
Configuration Tags

Tags determine behaviour of individual nodes

- `nofailover`: true/false - failover/switchover is disabled, node is never promoted to leader
- `noloadbalance`: true/false - /replica endpoint always returns 503
- `clonefrom`: true/false - node is used for cloning of new standbys in favor of cluster leader
- `nosync`: true/false - node never becomes a synchronous standby
- `replicatetfrom`: node_name - node to (cascadingly) replicate from
Replication Modes

• Patroni uses asynchronous physical replication by default
• Synchronous replication is possible
• Two different synchronous replication modes
  • synchronous_mode: true
  • synchronous_mode_strict: true
  • synchronous_node_count: n
• Cascading replication possible via replicatefrom tag
• Logical replication / change data capture including switch/failover support
  • Logical replication slots need to be registered with Patroni
  • Slot management and replication position advancement
  • Integration with pg_failover_slots pending
Replica Creation Options

- Default replica creation via `pg_basebackup`
- Other replica creation methods can be specified via `create_replica_methods`
- Allows replica creation from backups

```yaml
create_replica_methods:
- pgbackrest
  pgbackrest:
    command: /usr/bin/pgbackrest --stanza={{ stanza }} --delta restore
    keep_data: true
    no_params: true
  recovery_conf:
    restore_command: /usr/bin/pgbackrest --stanza={{ stanza }} archive-get %f %p
```
Deployment Topologies
Deployment Topologies

- 1-N Patroni/Postgres nodes
  - 1-node Patroni setups to leverage integrated configuration/administration
  - 2-node Patroni setups minimal requirement for high-availability
  - 3-node Patroni setups typical
- M>2 DCS nodes required to avoid single point of failure (SPOF)
- Number of DCS nodes should be odd (M=3,5,7…)
- DCS Deployment topologies:
  - DCS operated on same hosts as Patroni and PostgreSQL
  - DCS operated as stand-alone Cluster
  - DCS locally with PySyncObj RAFT (deprecated)
1 Node, Standalone DCS
2 Nodes, Standalone DCS
3 Nodes, Standalone DCS
2 Nodes + Quorum-Node, Local DCS
3 Nodes, Local DCS
3 Nodes, Internal DCS
Multi-Datacenter Deployments
Multi-Datacentre Deployments

- Multi-region, multi-datacenter (DC), multi-availability zone (AZ)
- Single Patroni cluster stretched over multiple DCs
  - Automatic failover and synchronous replication possible
- Separate Patroni clusters in separate DCs
  - Manual failover and asynchronous replication
Multi-Datacenter Deployments

Single Patroni Cluster

- Stretching a DCS cluster over multiple DCs is discouraged
  - Depending on distance, latency will be a problem
- If multi-DC automatic failover is a requirement, three DCs are needed
- At least one DCS node per DC, one DC can be witness (no Patroni instance)
Multi-Datacenter Deployments

Single Patroni Cluster
Multi-Datacenter Deployments

Multiple Patroni Clusters

- Patroni provides Standby-Cluster functionality
  - Secondary DC has own DCS cluster
  - Standby-Cluster leader replicates from primary cluster

```yaml
standby_cluster:
  host: dc1-1,dc1-2,dc1-3 # or VIP/service IP
  primary_slot_name: dc2
slots:
  dc1:
    type: physical
  dc2:
    type: physical
```

- Failover manually by removing `standby_cluster` section from config
- Former primary cluster needs to be fenced
Multi-Datacenter Deployments

Multiple Patroni Clusters
DCS Caveats and Considerations
DCS Caveats and Considerations

- Up until 2.x, Patroni was reliant on DCS being available
- If DCS is not reachable (down, or doing leader election)
  - Patroni assumes network partition
  - Leader is demoted to avoid split-brain

2024-02-18 09:59:18,301 ERROR: Request to server http://10.0.3.184:2379 failed:
ReadStreamTimeoutError("HTTPConnectionPool(host='10.0.3.184', port=2379): Read timed out. (r>
[...]
2024-02-18 09:59:24,971 ERROR: Error communicating with DCS
2024-02-18 09:59:24,972 INFO: demoting self because DCS is not accessible and I was a leader
2024-02-18 09:59:24,972 INFO: Demoting self (offline)

- etcd in particular requires strict network / I/O latency guarantees
  - Multi-DC latencies usually too high for defaults
  - I/O operations slower than 1s can lead to leader election
DCS Failsafe Mode

• Patroni 3.0 introduces DCS failsafe mode
• Leader maintains /failsafe DCS key, contains list of cluster members
• In case DCS is not available
  • Leader tries to contact all followers
  • In case all followers reply, cluster carries on
  • Otherwise, leader demotes itself
• Configuration: failsafe_mode: true
• https://patroni.readthedocs.io/en/latest/dcs_failsafe_mode.html

2024-02-18 10:18:44,450 ERROR: Error communicating with DCS
2024-02-18 10:18:44,461 INFO: Got response from pg2 http://10.0.3.31:8008/patroni: Accepted
2024-02-18 10:18:44,463 INFO: continue to run as a leader because failsafe mode is enabled and all members are accessible
etcd Considerations

- etcd is sensitive to resource starvation if run locally on Postgres nodes
  - Dedicated network interface advised
  - Dedicated storage device advised

- etcd can use up the default WAL space if used with Patroni
  - Set `ETCD_AUTO_COMPACTION_RETENTION` environment variable / `--auto-compaction-retention` option
THANK YOU!