About me

@github/database-infrastructure

Author of orchestrator, gh-ost, freno, ccql and others.

Blog at http://openark.org

@ShlomiNoach
Agenda

Raft overview

Why orchestrator/raft

orchestrator/raft implementation and nuances

HA, fencing

Service discovery

Considerations
Raft

Consensus algorithm
Quorum based
In-order replication log
Delivery, lag
Snapshots
HashiCorp raft

golang raft implementation

Used by Consul

Recently hit 1.0.0

github.com/hashicorp/raft
orchestrator

MySQL high availability solution and replication topology manager

Developed at GitHub

Apache 2 license

github.com/github/orchestrator
Why orchestrator/raft

- Remove MySQL backend dependency
- DC fencing
- And then good things happened that were not planned:
  - Better cross-DC deployments
  - DC-local KV control
  - Kubernetes friendly
**orchestrator/raft**

- **n** orchestrator nodes form a raft cluster
- Each node has its own dedicated backend database (MySQL or SQLite)
- All nodes probe the topologies
- All nodes run failure detection
- Only the leader runs failure recoveries
Implementation & deployment @ GitHub

- 5 Nodes (2xDC1, 2xDC2, 1xDC3)
- 1 second raft polling interval
- step-down
- raft-yield
- SQLite-backed log store
- MySQL backend (SQLite backend use case in the works)
A high availability scenario

**o2** is leader of a 3-node orchestrator/raft setup
Injecting failure

master: `killall -9 mysqld`

**o2** detects failure. About to recover, but…
Injecting 2nd failure

DROP DATABASE orchestrator;

o2: DROP DATABASE orchestrator;

o2 freaks out. 5 seconds later it steps down
orchestrator recovery

o1 grabs leadership
MySQL recovery

\textbf{o1} detected failure even before stepping up as leader.

\textbf{o1}, now leader, kicks recovery, fails over MySQL master.
orchestrator self health tests

Meanwhile, o2 panics and bails out.
Some time later, puppet kicks orchestrator service back on o2.
orchestrator startup

**orchestrator** service on o2 bootstraps, creates **orchestrator** schema and tables.
Joining raft cluster

$o_2$ recovers from raft snapshot, acquires raft log from an active node, rejoins the group
Grabbing leadership

Some time later, o2 grabs leadership
Assume this 3 DC setup

One orchestrator node in each DC

Master and a few replicas in \textbf{DC2}

What happens if \textbf{DC2} gets network partitioned?

i.e. no network in or out \textbf{DC2}
DC fencing

From the point of view of DC2 servers, and in particular in the point of view of DC2’s orchestrator node:

- Master and replicas are fine.
- DC1 and DC3 servers are all dead.
- No need for fail over.

However, DC2’s orchestrator is not part of a quorum, hence not the leader. It doesn’t call the shots.
In the eyes of either DC1’s or DC3’s orchestrator:

All DC2 servers, including the master, are dead.

There is need for failover.

DC1’s and DC3’s orchestrator nodes form a quorum. One of them will become the leader.

The leader will initiate failover.
DC fencing

Depicted potential failover result. New master is from DC3.
orchestrator/raft & consul

**orchestrator** is Consul-aware

Upon failover **orchestrator** updates Consul KV with identity of promoted master

Consul @ GitHub is DC-local, no replication between Consul setups

**orchestrator** nodes, update Consul locally on each DC
Considerations, watch out for

Eventual consistency is not always your best friend

What happens if, upon replay of raft log, you hit two failovers for the same cluster?

\texttt{NOW()} and otherwise time-based assumptions

Reapplying snapshot/log upon startup
orchestrator/raft roadmap

Kubernetes

ClusterIP-based configuration in progress

Already container-friendly via auto-reprovisioning of nodes via Raft
Thank you!

Questions?

github.com/shlomi-noach
@ShlomiNoach