PERCONA

TRAINING

High Availability

http://www.percona.com/training/

How we achieve HA?

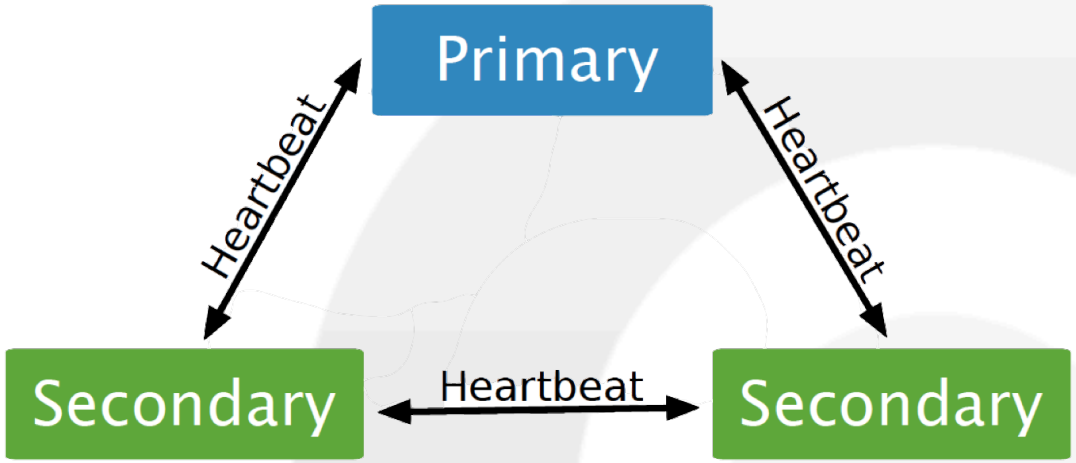
- Replica sets
- Node auto discovery (drivers)
- Auto failover
- Geo distribute the data

Node auto discovery

- Drivers run `isMaster` to determine the state of the replica set members and to discover additional members

```
> db.isMaster()
{
"topologyVersion" : {
"processId" : ObjectId("602be753469d04105300e1dd"),
"counter" : NumberLong(4)
},
"hosts" : [
"localhost:27017",
"localhost:27018"
],
"arbiters" : [
"localhost:27019"
],
"setName" : "myReplSet",
"setVersion" : 4,
"ismaster" : false,
"secondary" : true,
"primary" : "localhost:27018",
"me" : "localhost:27017",
...
```

Replication Heartbeat



Replication Heartbeat

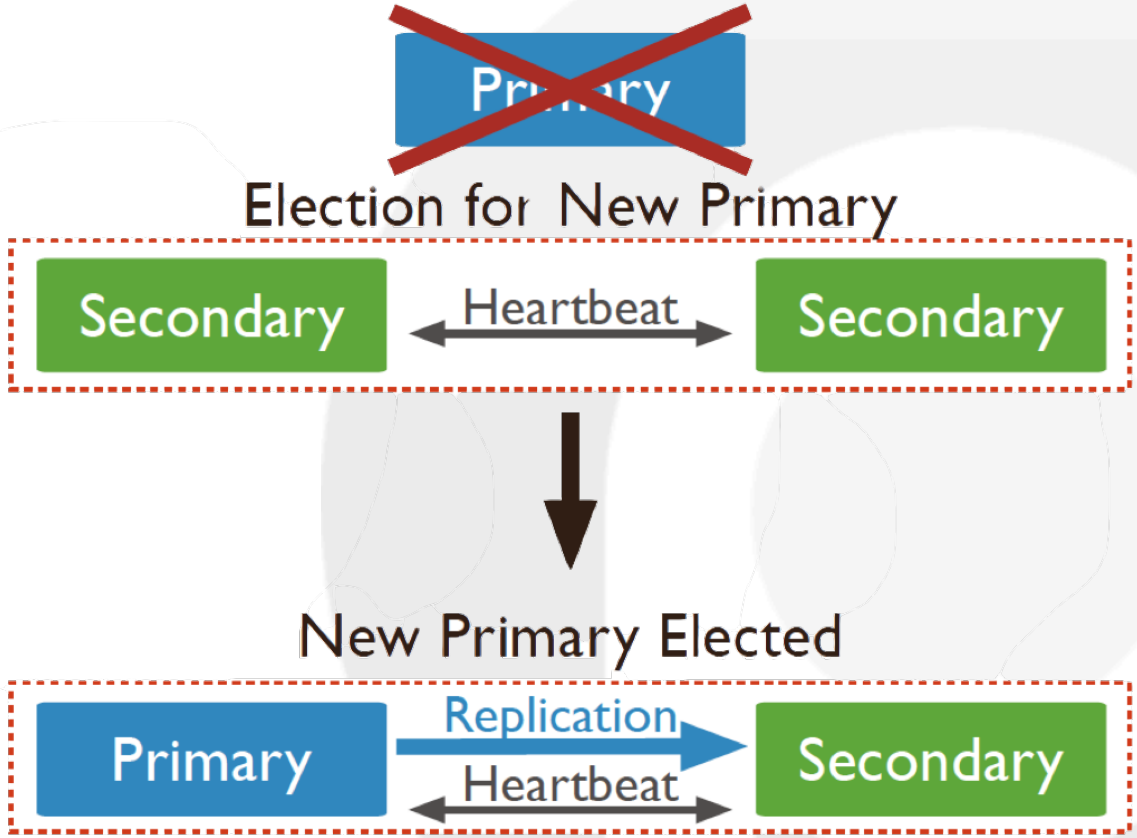
- Members ping each other every 2 sec
- If heartbeat doesn't return in 10 sec the member is marked innaccesible

Election of New Primary

What triggers an election?

- `rs.initiate()`
- `rs.stepDown()`
- Adding a new node to the replica set
- Primary unreachable by other members of the set for more than `electionTimeoutMillis` period (10 seconds by default)

Election of New Primary



Member priority

- Priority affects the outcome of elections
- Best-effort attempt to promote highest priority member to Primary
- Members continue to call elections until the highest priority member available becomes Primary
- Priority 0 members cannot become Primary

Votes

- Non-voting members must have priority 0
  - Can still serve reads
- Members with non-zero priority cannot have 0 votes
- Max of 7 voting members per replica set

Takeaway: Only alter the number of votes in exceptional cases - best to use priority to control election outcomes

Retryable writes

- MongoDB 4.2-compatible drivers enable retryable writes by default
- MongoDB 4.0 and 3.6-compatible drivers must explicitly enable retryable writes by including `retryWrites=true` in the connection string
- Blocks for up to `serverSelectionTimeoutMS` before throwing exception

Mirrored reads

- Starting in version 4.4, MongoDB pre-warms the cache of electable secondary members
- Mirror a subset of operations to a subset of electable secondaries
- Enabled by default with 1% rate
  - `db.adminCommand( { setParameter: 1, mirrorReads: { samplingRate: 0.01 } } )`

Tolerance and HA

Topics:

- Rolling upgrades and maintenance risks
- Secondary Reads - pros and cons
- Tags - replication use cases
- Tags - sharding use cases
- Resolving a rollback

Rolling Maintenance

1. Perform work on SECONDARY servers (one at a time)
2. Issue `rs.stepDown()`
3. Perform work on old primary

Rolling Maintenance

- Stepdown pauses for up to *secondaryCatchUpPeriodSecs* for secondaries to catch up (default 10 seconds)
- If still no electable secondary catches up, the primary does not step down and the method errors

Rolling Upgrades - Sharded clusters

- Same rolling approach

Order:

1. Config server replica set
2. Shard replica sets
3. mongos nodes

Rolling Upgrades and Maintenance Risks

- Before 4.2 all connections were killed in a short time
- Writes fail while stepDown is in process
- Taking one secondary offline will increase HA error risks!
- `rs.freeze()` can temporarily prevent a secondary from becoming primary

Rolling Upgrades and Maintenance Risks (2)

- Control the features that persist data incompatible with earlier versions
- Enabling backwards-incompatible features can complicate downgrades
- Allow your deployment to run without enabling these features for a burn-in period

```
db.adminCommand({ setFeatureCompatibilityVersion: <version> })
```

Secondary Reads - Pros and Cons

- Up to 3x the number of read per second for a typical 3-node cluster
- Much better use of resources than just being there for failovers
- Elections, step-downs, or taking a machine down for maintenance all will be more costly

Tags: Replication Use Cases

- Send different types of traffic to different nodes
  - Reporting Nodes
  - Nodes using Memory engine for caching, real-time analytics, etc.
  - In a Geo-distributed replset, app can query from `nearest` member node

Tags: Replication Use Cases (2)

- Routing Queries based on indexes available
  - Custom set of indexes on few member nodes for specific queries
  - Fewer indexes on `PRIMARY` to improve writes

Tags: Sharding Use Cases

- Taking it to the next level you can tag things to break down:
  - Sending writes to the best datacenter
  - Sending some types of workloads for a sharded collection to the fastest hardware, then as it ages "replace" the document to move it to slower gear
  - Using `client_id` to dedicate shards for ultra-premium accounts, while letting the general accounts be evenly sharded over all shards

Rollbacks

- Primary accepted write operations that the secondaries had not successfully replicated
- Rollback data is available in BSON format
- Flow control helps minimize lag
- Write concern majority helps avoid rollbacks
- The rollback time limit is configurable using the parameter `rollbackTimeLimitSecs` (default 24 h)

Rollbacks

- For each collection whose data is rolled back, the rollback files are located in /dbpath/rollback/collectionUUID e.g.

```
/dbpath/rollback/20f74796-d5ea-42f5-8c95-f79b39bad190/removed.2020-02-19T04-57-11.0.bson
```

- To match the UUID with the collection name

```
grep "rollback file" /var/log/mongodb/mongod.log
```

- Read the data with `bsondump`

Disaster Recovery & Geo Redundancy

- DR is not necesarily Geo redundancy, which do you really need, maybe both?
  - How much latency and data loss is acceptable?
- Sharding makes it harder
  - A replica set member in different Geo location?
  - 3rd party tool to replicate between sharded clusters?

Exercises

Rebuild a replicaset member

Rebuild a replicaset member

1. Connect to the second member of the replicaset

```
mongo --port 27018
```

2. Shutdown the instance from the console

```
use admin
db.shutdownServer()
```

3. Verify the second member is down from the `PRIMARY`

```
mongo --port 27017

rs.status()
```

Rebuild a replicaset member

4. We should see something like below

```
    {
      "_id" : 1,
      "name" : "localhost:27018",
      "health" : 0,
      "state" : 8,
      "stateStr" : "(not reachable/healthy)",
      ...
    },
```

5. Delete the existing data directory

```
rm -rf /mongodb/data/rs2/*
```

Rebuild a replicaset member

6. Start `mongod` with empty data directory

```
screen -S rs2 -d -m mongod --replSet myReplSet --dbpath /mongodb/data/rs2 \
  --port 27018 --oplogSize 200 --wiredTigerCacheSizeGB 0.25
```

7. Verify the state changes from the `PRIMARY`

```
mongo --port 27017

# run this a couple times to see how the status changes from STARTUP to SECONDARY
rs.status().members
```

8. Inspect the logs to see the initial sync process

```
screen -x rs2
```

Exercises

Step Down the current `PRIMARY`

Step Down (Demote) the Current PRIMARY

1. Connect to the current PRIMARY

```
mongo --port 27018
```

2. Stepdown to demote current PRIMARY status

```
rs.stepDown(60)
```

3. Wait a few seconds and verify status

```
rs.status()
```

PERCONA
TRAINING

Questions