Maintenance for MongoDB Replica Sets

Oct-2-2019

Pythian

About Me


```
{"name": "Igor Donchovski",
"lives_in": "Skopje",
"email": "donchovski@pythian.com",
"current_role": "Lead database consultant",
"education": [{"type": "College", "name": "FEIT", "graduated": "2008", "university": "UKIM"},
              {"type": "Master", "name": "FINKI", "graduated": "2013", "university": "UKIM"}],
"work": [{"role": "Web developer", "start": "2007", "end": "2012", "company": "Gord Systems"},
         {"role": "DBA", "start": "2012", "end": "2014", "company": "NOVP"},
         {"role": "Database consultant", "start": "2014", "end": "2016", "company": "Pythian"},
         {"role": "Lead database consultant", "start": "2016", "company": "Pythian"}],
"certificates": [{"name": "C100DBA", "issued": "2016", "description": "MongoDB certified DBA"}],
"social": [{"network": "LinkedIn", "link": "www.linkedin.com/in/igorle"},
           {"network": "Twitter", "link": "https://twitter.com/igorle", "handle": "@igorle"}],
"interests": ["Hiking", "Biking", "Traveling"],
"hobbies": ["Painting", "Photography", "Cooking"],
"proud_of": ["Volunteering", "Helping the Community"]}
```

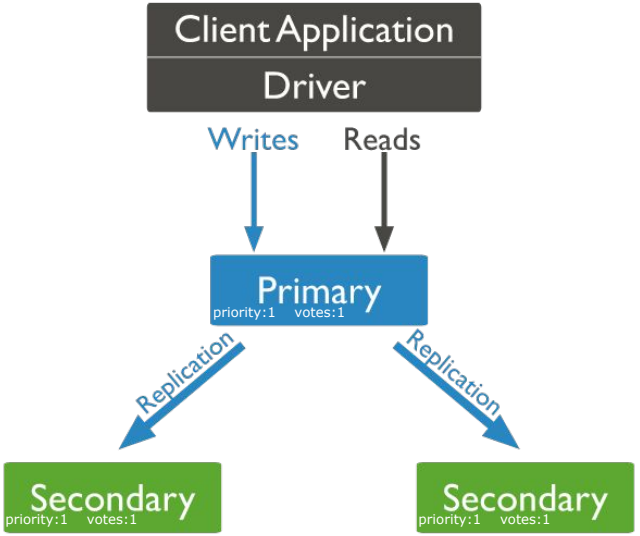
Overview

- How MongoDB replication works
- Replica set configuration, deployment topologies
- Reconfiguring replica set members
- Hardware changes, OS patching
- Database upgrades, downgrades
- Building indexes
- Backups and restores
- QA

Replica Set

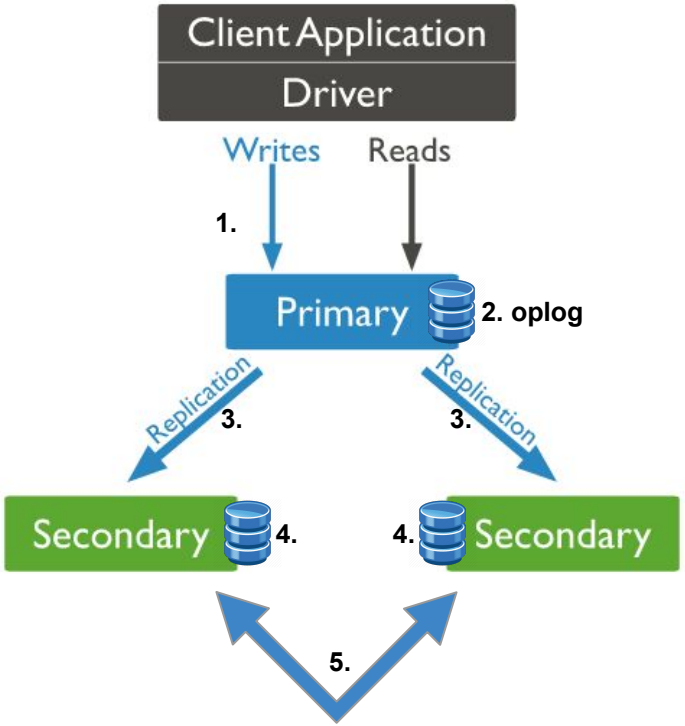
- Group of mongod processes that maintain the same data set
- Redundancy and high availability
- Increased read capacity (scaling reads)
- Automatic failover

# Members	# Nodes Required to Elect New Primary	Fault Tolerance
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3



Replication Concept

1. Write operations go to the Primary node
2. All changes are recorded into operations log
3. Asynchronous replication to Secondary
4. Secondaries copy the Primary oplog
5. Secondary can use sync source Secondary*



*settings.chainingAllowed (true by default)

Replica Set Oplog

- Special capped collection that keeps a rolling record of all operations that modify the data stored in the databases
- Idempotent
- Default oplog size

For Unix and Windows systems

Storage Engine	Default Oplog Size	Lower Bound	Upper Bound
In-memory	5% of physical memory	50MB	50GB
WiredTiger	5% of free disk space	990MB	50GB
MMAPv1	5% of free disk space	990MB	50GB

Deploy Replica Set

- Start each member with the appropriate options

```
mongod --config /etc/mongod.conf
```

- Initiate the replica set on one node

```
rs.initiate()
```

- Confirm the replica set configuration

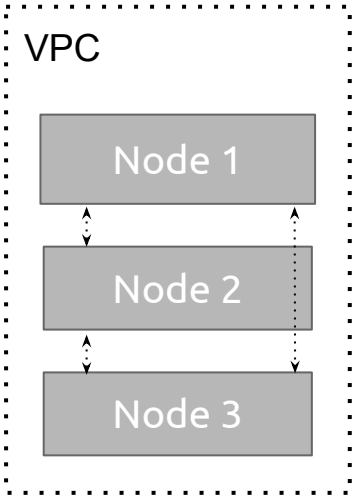
```
rs.conf()
```

- Add the rest of the members

```
rs.add()
```

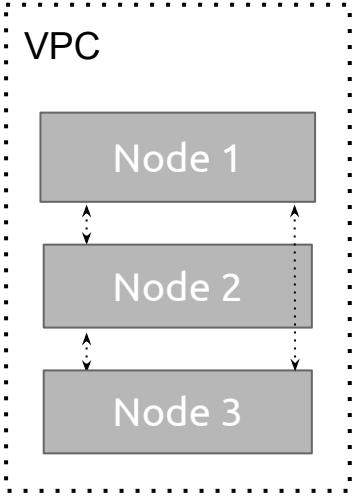
- Confirm the replica has Primary node

```
rs.status()
```



mongod.conf

```
storage:
  dbPath: /var/lib/mongo
replication:
   oplogSizeMB: 10240
   replSetName: "rs_test"
net:
    port: 27017
    ssl:
      mode: requireSSL
      PEMKeyFile: /etc/ssl/mongodb.pem
      CAFile:  /etc/ssl/ca.pem
systemLog:
    destination: file
    path: "/logs/mongodb.log"
security:
    keyFile: "/path/key/rs.key"
    authorization: "enabled"
```



Deploy Replica Set

- Start each member with the appropriate options

  ```
  mongod --config /etc/mongod.conf
  ```

- Initiate the replica set on one node

  ```
  rs.initiate()
  ```

- Confirm the replica set configuration

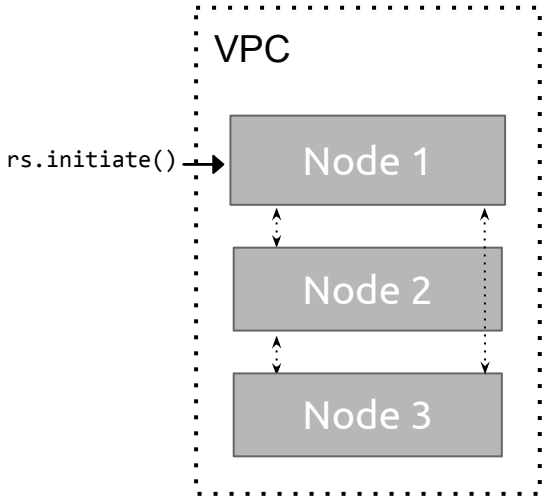
  ```
  rs.conf()
  ```

- Add the rest of the members

  ```
  rs.add()
  ```

- Confirm the replica has Primary node

  ```
  rs.status()
  ```



Deploy Replica Set

- Start each member with the appropriate options

  `mongod --config /etc/mongod.conf`

- Initiate the replica set on one node

  `rs.initiate()`

- Confirm the replica set configuration

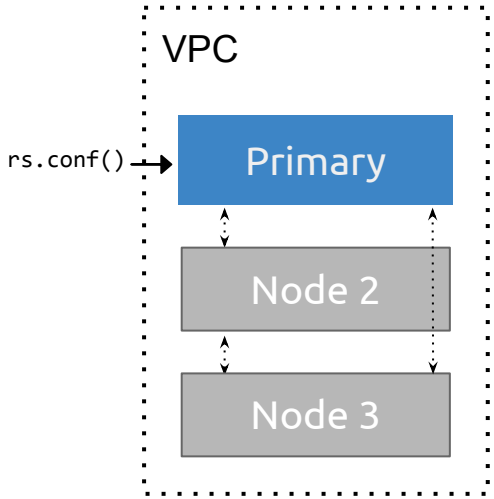
  `rs.conf()`

- Add the rest of the members

  `rs.add()`

- Confirm the replica has Primary node

  `rs.status()`



rs.conf()

```
{
   "_id" : "rs_test",
   "version" : 1,
   "protocolVersion" : NumberLong(1),
   "members" : [
       {"_id" : 0,
       "host" : "node1.net:27017",
       "arbiterOnly" : false,
       "buildIndexes" : true,
       "hidden" : false,
       "priority" : 1,
       "tags" : {},
       "slaveDelay" : NumberLong(0),
       "votes" : 1
   ],
"settings" : { ..............     --->
```

```
   "settings" : {
       "chainingAllowed" : true,
       "heartbeatIntervalMillis" : 2000,
       "heartbeatTimeoutSecs" : 10,
       "electionTimeoutMillis" : 10000,
       "catchUpTimeoutMillis" : -1,
       "getLastErrorModes" : {

       },
       "getLastErrorDefaults" : {
       "w" : 1,
       "wtimeout" : 0
       },
       "replicaSetId" : ObjectId("585ab9df685f726db2c6a840")
   }
}
```

Deploy Replica Set

- Start each member with the appropriate options

  `mongod --config /etc/mongod.conf`

- Initiate the replica set on one node

  `rs.initiate()`

- Confirm the replica set configuration

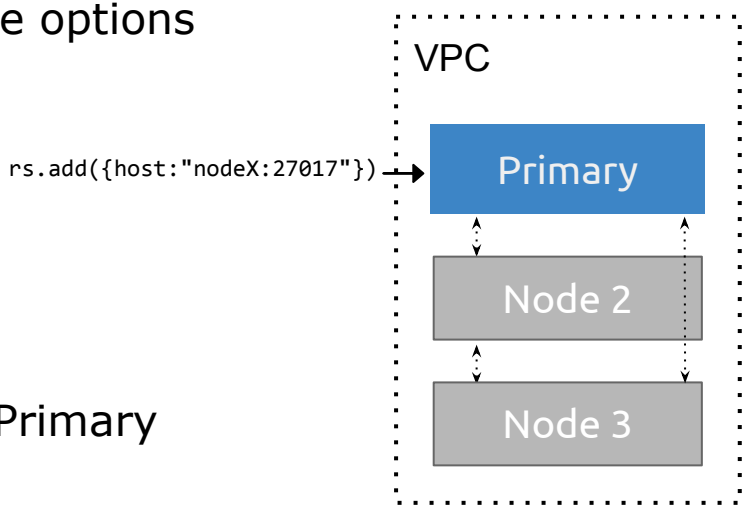
  `rs.conf()`

- Add the rest of the members from the Primary

  `rs.add()`

- Confirm the replica has Primary node

  `rs.status()`



Deploy Replica Set

- Start each member with the appropriate options

  ```
  mongod --config /etc/mongod.conf
  ```

- Initiate the replica set on one node

  ```
  rs.initiate()
  ```

- Confirm the replica set configuration

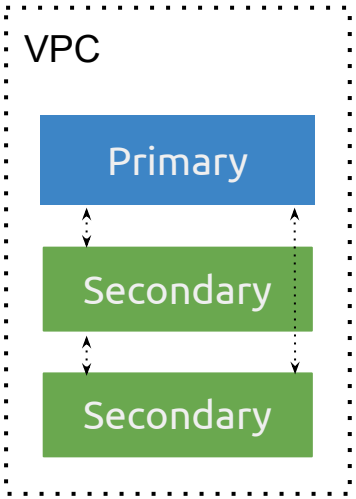
  ```
  rs.conf()
  ```

- Add the rest of the members

  ```
  rs.add()
  ```

- Confirm the replica has Primary node

  ```
  rs.status()
  ```



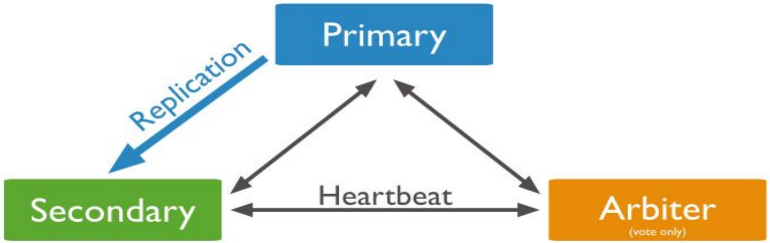
Configuration Options

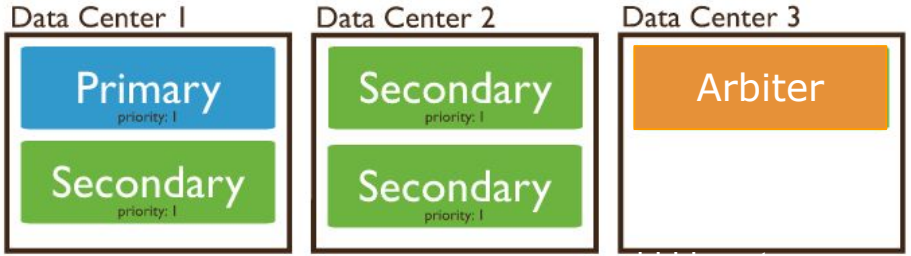
- 50 members per replica set (7 voting members)
- Arbiter node
- Priority 0 node
- Hidden node
- Delayed node

Arbiter Node

- Does not hold copy of data
- Votes in elections

```
rs.addArb("arbiter.net:27017")
```



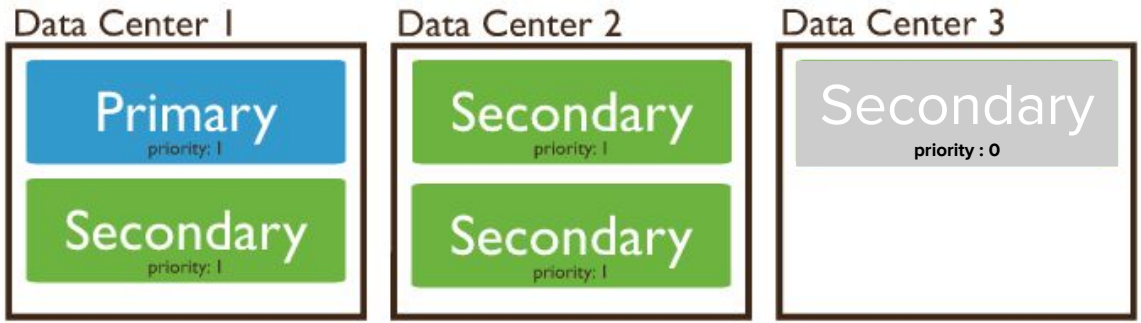


Priority 0 Node

Priority - floating point (i.e. decimal) number between 0 and 1000

- Cannot become primary, cannot trigger election
- Visible to application (accepts reads/writes)
- Votes in elections

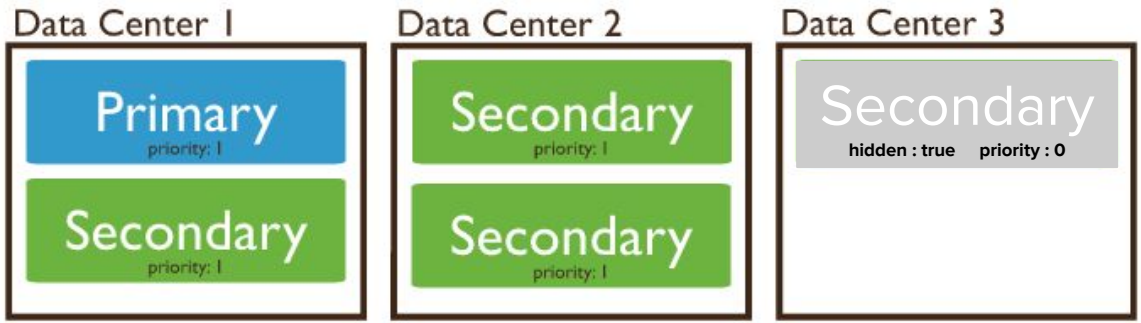
```
rs.add( { host: "mongodb5.net:27017", priority: 0 } )
```



Hidden Node

- Not visible to application
- Never becomes primary, but can vote in elections
- Use cases (reporting, backups)

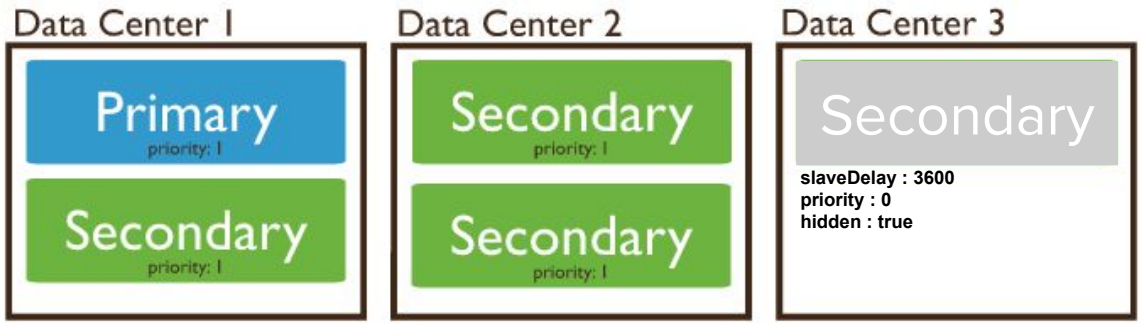
```
rs.add( { host: "mongodb5.net:27017", priority: 0, hidden: true } )
```



Delayed Node

- Must be priority 0 member
- Should be hidden member (not mandatory)
- Mainly used for backups (historical snapshot of data)
- Recovery in case of human error

```
rs.add( { host: "mongodb5.net:27017", priority: 0, hidden: true, slaveDelay=3600 } )
```



Maintenance

It's highly recommended that you test your changes in pre-production before applying on a running production environment

Increase Oplog Size

- Connect to each replica set member
- Change the oplog size of the replica set member (size in MB)

```
MongoDB >= 3.6: db.adminCommand({replSetResizeOplog: 1, size: 32768})
```

- Confirm the size of the oplog

```
use local

db.oplog.rs.stats()
```

- Update mongod.conf file with new value `(oplogSizeMB: 32768)`

Replica maintenance

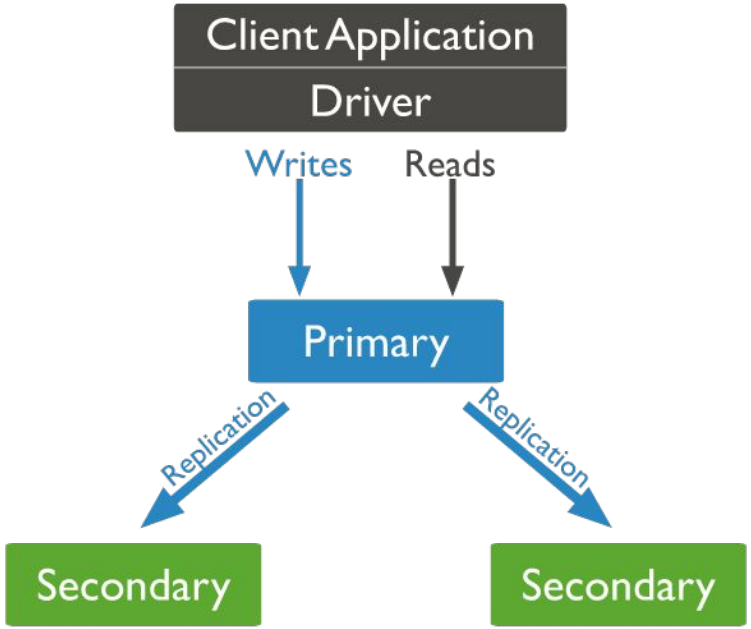
Includes changes for priority, hidden, slaveDelay, tags

- Save rs.conf() to a variable

```
> cfg = rs.conf()
```

- Print out the cfg variable in your shell

```
> cfg
```



cfg=rs.conf()

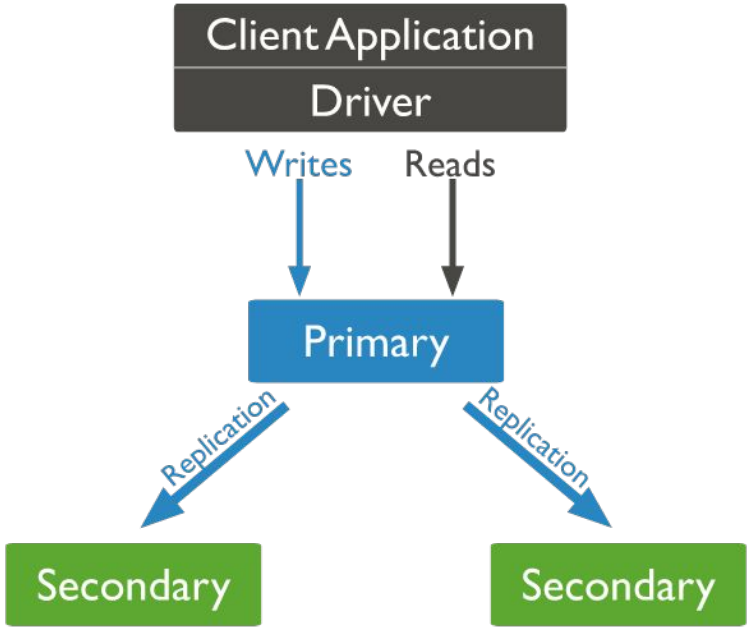
```
{
   "_id" : "rs_test",
   "version" : 1,
   "protocolVersion" : NumberLong(1),
   "members" : [
        {"_id" : 0,
        "host" : "node1.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {},
        "slaveDelay" : NumberLong(0),
        "votes" : 1
        },
        {"_id" : 1,
        "host" : "node2.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {},
        "slaveDelay" : NumberLong(0),
        "votes" : 1
        },
        {"_id" : 2,
        "host" : "node3.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 1,
        "tags" : {},
        "slaveDelay" : NumberLong(0),
        "votes" : 1 }],
    "settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 10000,
        "catchUpTimeoutMillis" : -1,
        "getLastErrorModes" : {

        },
        "getLastErrorDefaults" : {
        "w" : 1,
        "wtimeout" : 0
        },
        "replicaSetId" :
ObjectId("585ab9df685f726db2c6a840")
   }
}
```

Replica maintenance

- Change the necessary settings for the desired nodes

```
> cfg.members[0].priority = 0.5
> cfg.members[1].priority = 2
> cfg.members[2].priority = 0
> cfg.members[2].hidden = true
> cfg.members[2].slaveDelay = 3600
> cfg.settings.electionTimeoutMillis = 12000
```



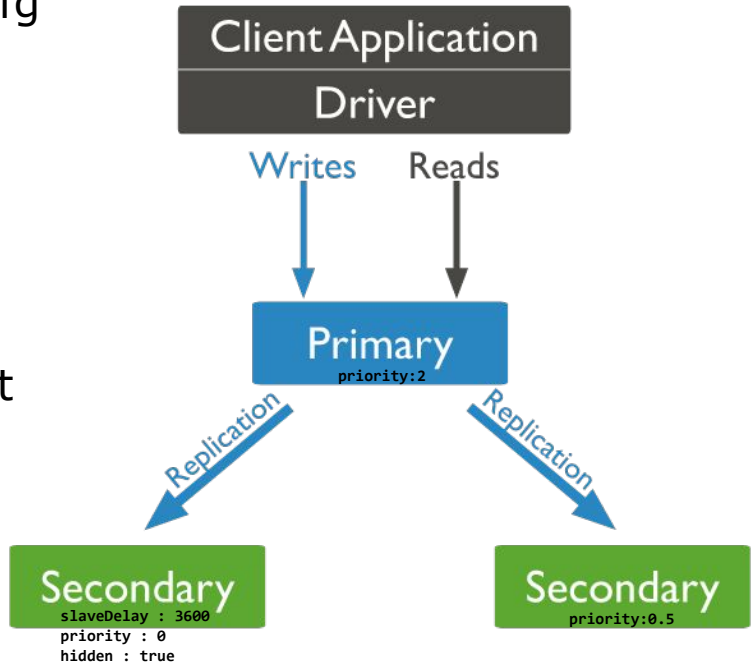
Replica maintenance

- Confirm the changes, print the variable cfg in your shell again

```
> cfg
```

- All changes look good?
- Assign the configuration to the replica set

```
> rs.reconfig(cfg)
```



rs.conf()

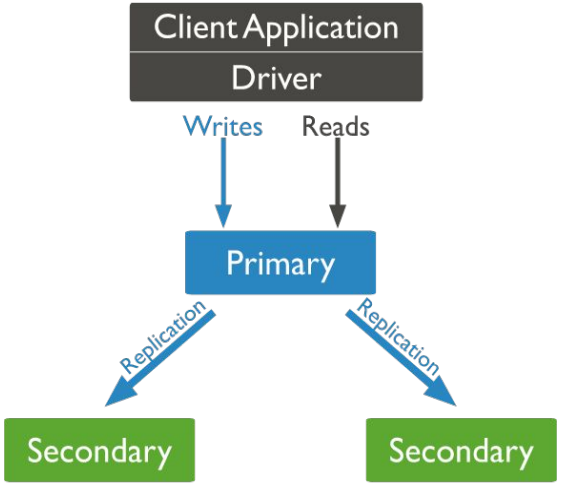
```
{
   "_id" : "rs_test",
   "version" : 1,
   "protocolVersion" : NumberLong(1),
   "members" : [
        {"_id" : 0,
        "host" : "node1.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 0.5,
        "tags" : {},
        "slaveDelay" : NumberLong(0),
        "votes" : 1
        },
        {"_id" : 1,
        "host" : "node2.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : false,
        "priority" : 2,
        "tags" : {},
        "slaveDelay" : NumberLong(0),
        "votes" : 1
        },
        {"_id" : 2,
        "host" : "node3.net:27017",
        "arbiterOnly" : false,
        "buildIndexes" : true,
        "hidden" : true,
        "priority" : 0,
        "tags" : {},
        "slaveDelay" : 3600,
        "votes" : 1 }],
    "settings" : {
        "chainingAllowed" : true,
        "heartbeatIntervalMillis" : 2000,
        "heartbeatTimeoutSecs" : 10,
        "electionTimeoutMillis" : 12000,
        "catchUpTimeoutMillis" : -1,
        "getLastErrorModes" : {

        },
        "getLastErrorDefaults" : {
        "w" : 1,
        "wtimeout" : 0
        },
        "replicaSetId" :
ObjectId("585ab9df685f726db2c6a840")
   }
}
```

Read preference

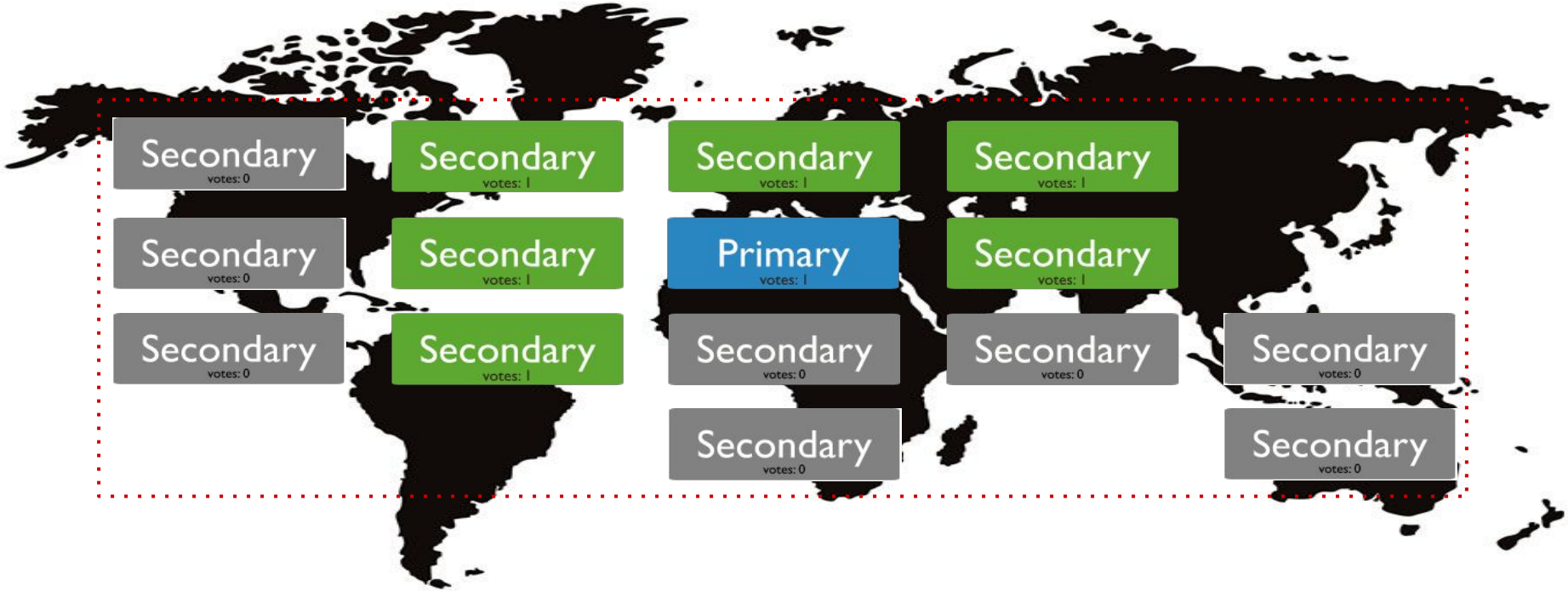
How read operations are routed to replica set members

1. primary (by default)
2. primaryPreferred
3. secondary
4. secondaryPreferred
5. nearest (least network latency)



MongoDB 3.4 maxStalenessSeconds (>= 90 seconds)

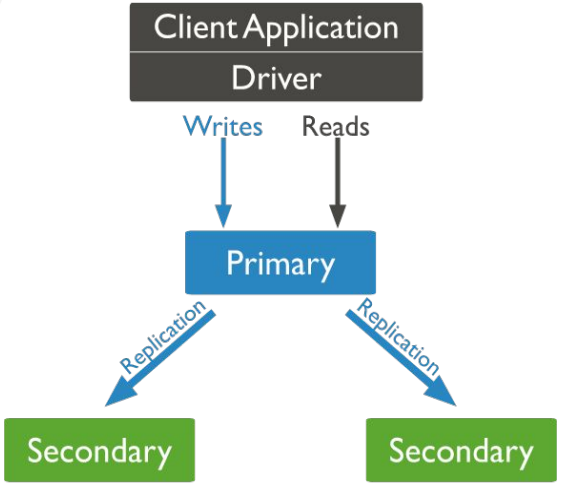
Read preference



Pythian
love your data®

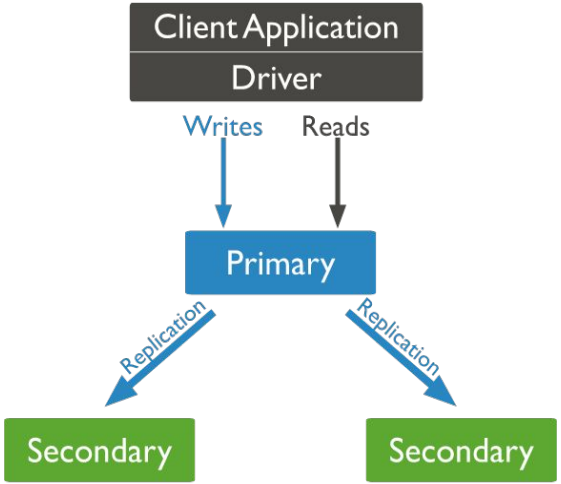
Tag sets

```
{
   "_id" : "rs_test",
   "version" : 1,
   "protocolVersion" : NumberLong(1),
   "members" : [
       { "_id" : 0, "host" : "mongodb0.net:27017", ...,  "tags": { }, ... },
       { "_id" : 1, "host" : "mongodb1.net:27017", ...,  "tags": { }, ... },
       { "_id" : 2, "host" : "mongodb2.net:27017", ...,  "tags": { }, ... }
   ],
   "settings" : {
       ...
   } …
}
```



Tag sets

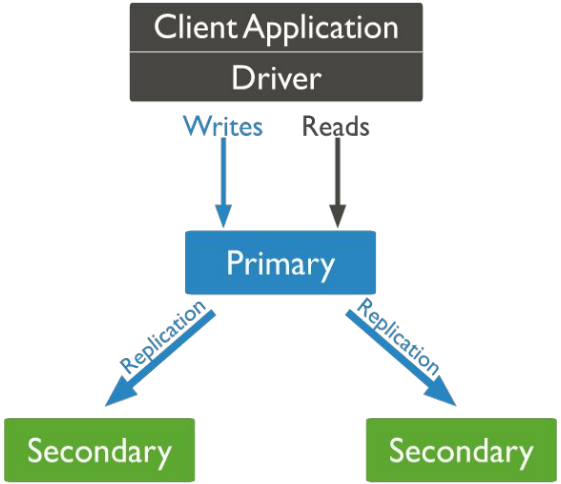
```
cfg = rs.conf();
conf.members[0].tags = { "dc": "europe", "usage": "production" };
conf.members[1].tags = { "dc": "europe", "usage": "reporting" };
conf.members[2].tags = { "dc": "north_america", "usage": "production" };
conf.members[3].tags = { "dc": "north_america", "usage": "reporting" };
conf.members[4].tags = { "dc": "africa", "usage": "production" };
conf.members[5].tags = { "dc": "africa", "usage": "reporting" };
conf.members[6].tags = { "dc": "apac", "usage": "reporting" };
rs.reconfig(cfg);
```




```
db.foo.find({}).readPref( "secondary", [ { "dc": "europe", "usage": "production" }, {} ] )
db.foo.find({}).readPref( "secondary", [ { "dc": "africa"}, { "usage": "reporting" }, {} ] )
db.foo.find({}).readPref( "secondary", [ { "dc": "apac", "usage": "production" }] )
```

Tag sets

```
cfg = rs.conf();
conf.members[0].tags = { "dc": "europe", "usage": "production" };
conf.members[1].tags = { "dc": "europe", "usage": "reporting" };
conf.members[2].tags = { "dc": "north_america", "usage": "production" };
conf.members[3].tags = { "dc": "north_america", "usage": "reporting" };
conf.members[4].tags = { "dc": "africa", "usage": "production" };
conf.members[5].tags = { "dc": "africa", "usage": "reporting" };
conf.members[6].tags = { "dc": "apac", "usage": "reporting" };
rs.reconfig(cfg);
```




```
db.foo.find({}).readPref( "secondary", [ { "dc": "europe", "usage": "production" }, {} ] )
db.foo.find({}).readPref( "secondary", [ { "dc": "africa"}, { "usage": "reporting" }, {} ] )
db.foo.find({}).readPref( "secondary", [ { "dc": "apac", "usage": "production" }] )  <--  Error
```


Pythian
love your data

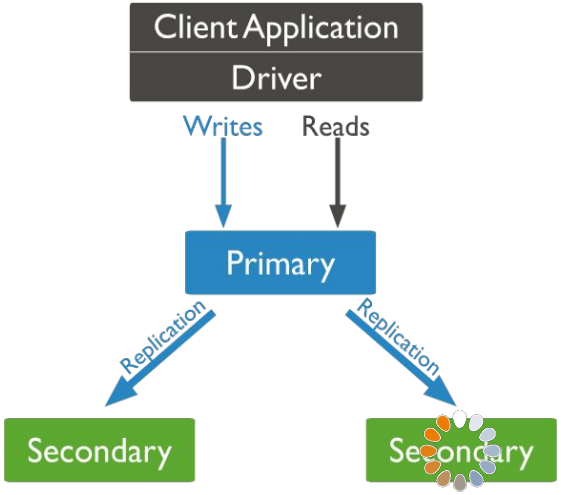
Resync Replica member

- Stop mongod process on the stale node
- Remove everything from the `--dbPath` directory
- Start mongod process
- Wait initial sync to finish automatically

  (This procedure can be used to change storage engines)

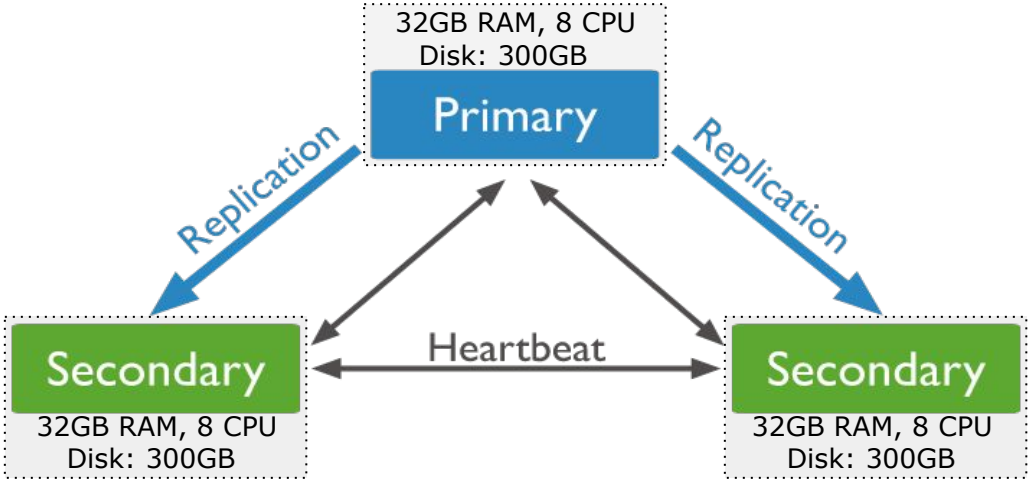
  Alternative

- Copy the data files from a Secondary that is locked for writes `--db.fsyncLock()`
- Sync the stale node



Hardware/OS upgrades

- Increase CPU
- Increase DISK
- Increase RAM
- Network changes
- OS patches
- OS upgrade
- Oplog (MongoDB < 3.6)
- More



Hardware/OS upgrades

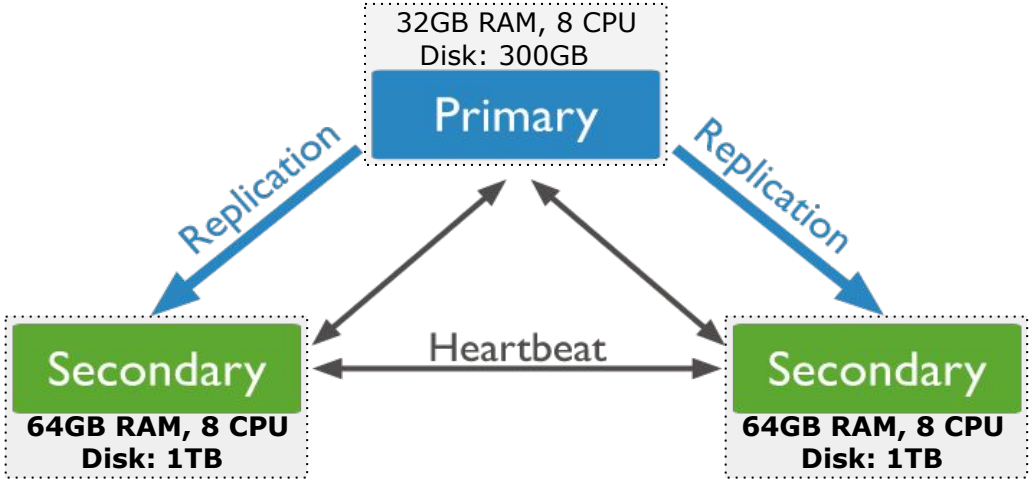
- Always start with Secondary nodes (`hidden, delayed, priority:0`)
- Confirm your replication window

```
> rs.printReplicationInfo()
```

```
> rs.printSlaveReplicationInfo()
```

- Understand how long your Secondaries can be "offline" if you need to stop mongod process
- Finish the maintenance on Secondaries one by one
- Wait Secondaries to sync with Primary

Hardware/OS upgrades

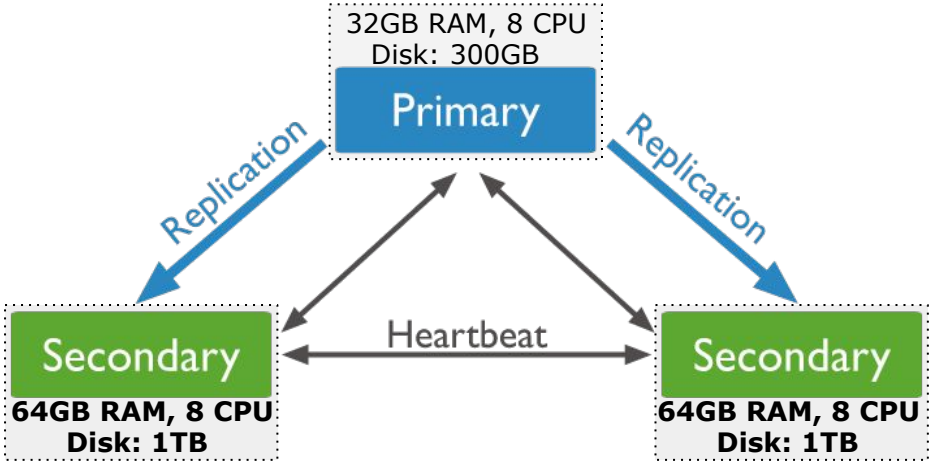


Hardware/OS upgrades

- Step down the Primary

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

- Confirm new Primary has been elected
- Do the same changes on the former Primary

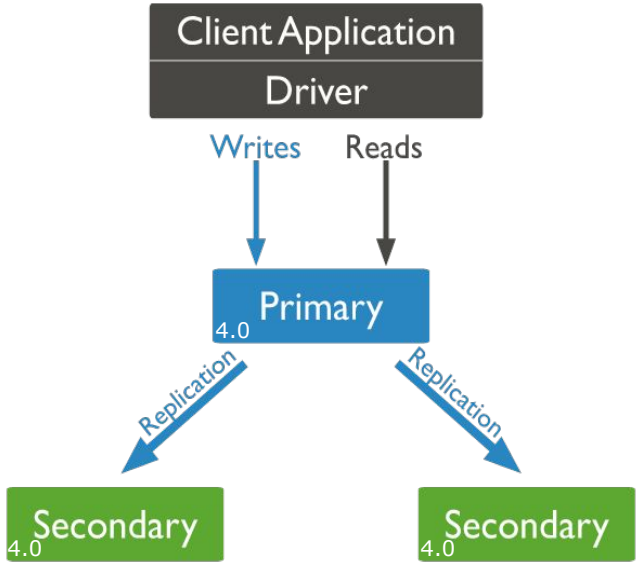


Database Upgrades

- Replica set major version upgrade (4.0 > 4.2)
- All nodes must be running version 4.0
- Study the compatibility changes with the new version, confirm your driver compatibility
- Confirm your feature compatibility is (current) version 4.0 on all nodes

```
> db.adminCommand( { getParameter: 1,
featureCompatibilityVersion: 1 } )

"featureCompatibilityVersion" : { "version" : "4.0" }
```



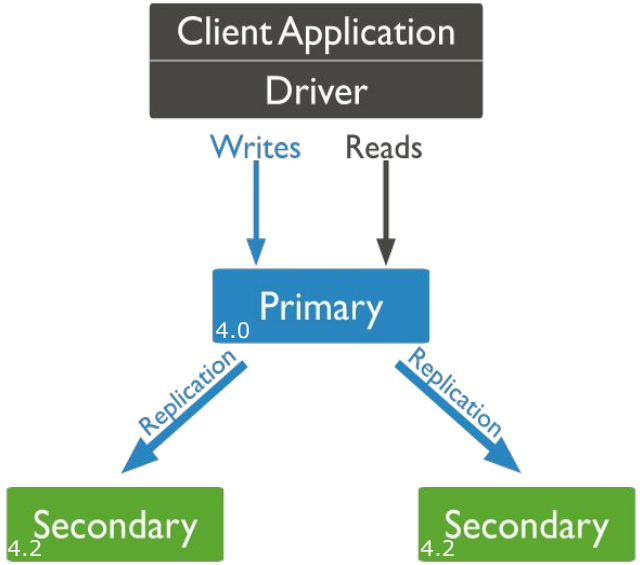
Database Upgrades

- Take backup prior to starting the upgrade process
- Upgrade the Secondary nodes one by one
  - Shutdown mongod and upgrade binaries
  - Restart mongod
- Step down the Primary

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

- Confirm new Primary was elected

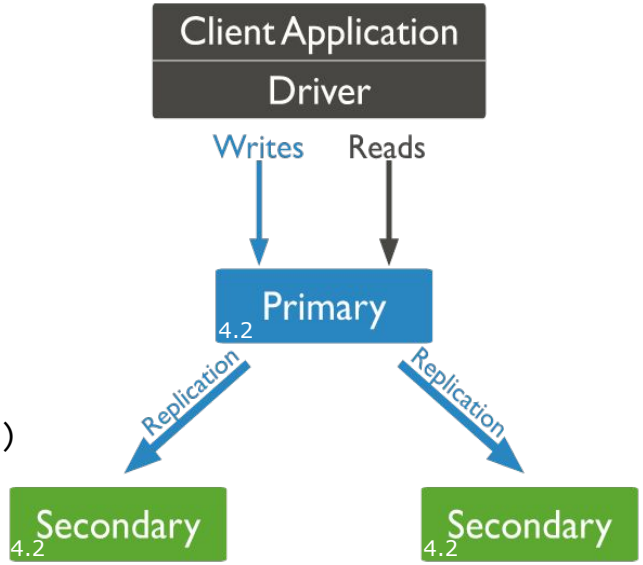
```
> rs.status()
```



Database Upgrades

- Upgrade the former Primary following the same steps as for the Secondaries
  - Shutdown mongod and upgrade binaries
  - Restart mongod
- Run the database at least for 1 week
- Enable 4.2 feature compatibility

```
> db.adminCommand( { setFeatureCompatibilityVersion: "4.2" } )
```



Database Upgrades

- Version 4.2
  - Removed MMapv1 storage engine
- Version 4.0
  - Removed support for MONGODB-CR authentication
  - Removed pv0 protocol for Replica Sets
- Version 3.6
  - Default bind to localhost
- Version 3.0
  - WiredTiger is the default storage engine

Database Downgrades

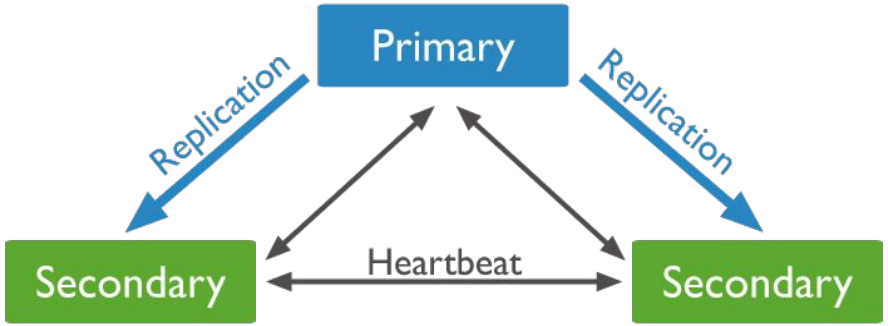
Same procedure as for the upgrades, Secondaries before Primary

- Remove incompatible features. Each database version has its own set of feature compatibility (3.2 → 3.4) (3.4 → 3.6) (3.6 → 4.0) (4.0 → 4.2)
- Download previous version binaries
- Downgrade all Secondary nodes
- Downgrade Arbiter
- Step down the Primary
- Downgrade the former Primary

DDL Operation (1)

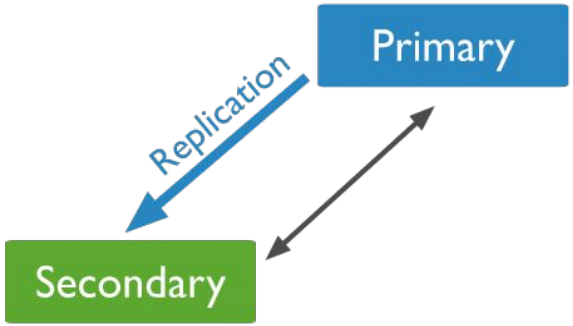
- Adding index on a collection, prior MongoDB 4.2
- Connect to the Primary node

```
db.people.createIndex( { zipcode: 1 }, { background: true } )
```



DDL Operation (2)

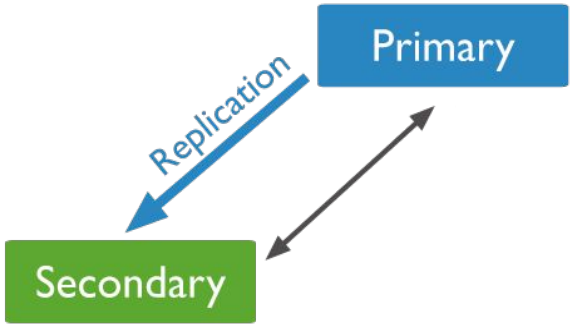
- Stop one Secondary
- Restart on different port
- Remove the setting for replica set



Secondary
--port=27777

DDL Operation (2)

- Add the Index on standalone node
- Rejoin the node to the replica after the index was added
- Repeat the process on the other Secondary
- Step down the Primary
- Add the index



Secondary
--port=27777

```
>db.people.createIndex({zipcode:1})
```

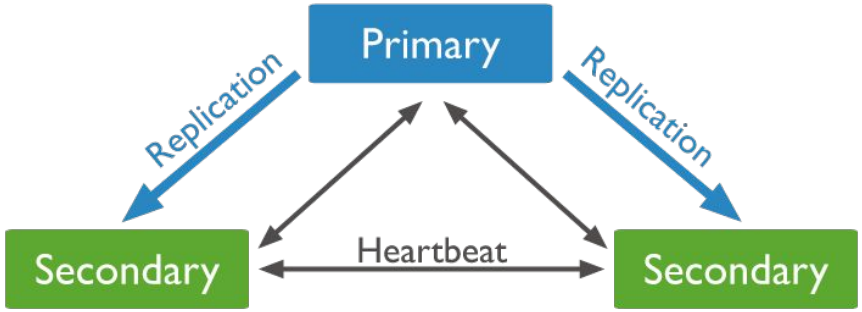
Backups

- MongoDB Cloud manager backup requires monthly subscription
- Mongodump as logical backup for smaller datasets
- Backup with file system snapshot
  - Pick one Secondary
  - Lock the database for writes

```
> db.fsyncLock()
```

  - Take snapshot
  - Unlock the database

```
> db.fsyncUnlock()
```



Restore in new Replica set

- Get the backup to your first node in the replica
- Start standalone *mongod* using the backup files
- Drop *local* database and shutdown the standalone
- Start new single node replica set
- Add additional members to the replica set
  - Copy the files from the single node to the other nodes, or
  - Perform initial sync on the other nodes

Useful when restoring QA/Staging from production backups

Enable access control

`--transitionToAuth` option for performing a no-downtime upgrade to enforcing authentication on running replica set (available since MongoDB 3.4)

- Connect to the Primary node and create necessary users
- Modify your Applications to start using authentication
- Create and copy a *keyfile* on all nodes in your replica set

- Restart all Secondary nodes with *transitionToAuth* configuration variable
- Step down the Primary node and restart it with transitionToAuth
- At this point all database connections should start using Authentication

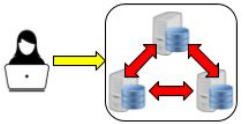
- Again restart all Secondaries, now without *transitionToAuth* configuration variable
- At last, step down the Primary and restart mongod without *transitionToAuth*
- The replica set is now enforcing Authentication

Upgrade replica to use TLS

1. Restart the processes with `ssl.Mode: allowSSL`

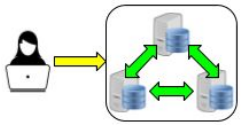
```
net:
  ssl:
    mode: allowSSL
```

   Switch the clients to use TLS/SSL



2. Upgrade to preferSSL by issuing the command on each node

```
db.adminCommand( { setParameter: 1, sslMode: "preferSSL" } )
```

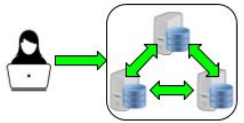


3. Upgrade to requireSSL by issuing the command on each node

```
db.adminCommand( { setParameter: 1, sslMode: "requireSSL" } )
```

   Update the config file to persist the settings

```
net:
  ssl:
    mode: requireSSL
```



Monitoring Replica Set

- Replica set has no Primary
- Number of unhealthy members is above threshold
- Replication lag is above threshold
- Replica set elected new Primary
- Host of any type has restarted
- Host of type Secondary is recovering
- Host of any type is down
- Host of any type has experienced Rollback
- Network issues between members of the replica set
- Monitoring backup status

Summary

- Replica set with odd number of voting members
- Hidden or Delayed member for dedicated functions (reporting, backups …)
- Have more than one eligible Primary in the replica set
- Always first patch the Secondaries before starting the Primary
- Run replica set members with same version, hardware/OS for all nodes
- Monitor your replica set status and nodes
- Monitor replication lag and Oplog size
- Regularly upgrade your database as new version mature
- Take backup and regularly run restore tests
- Secure your database by enabling authentication and authorisation