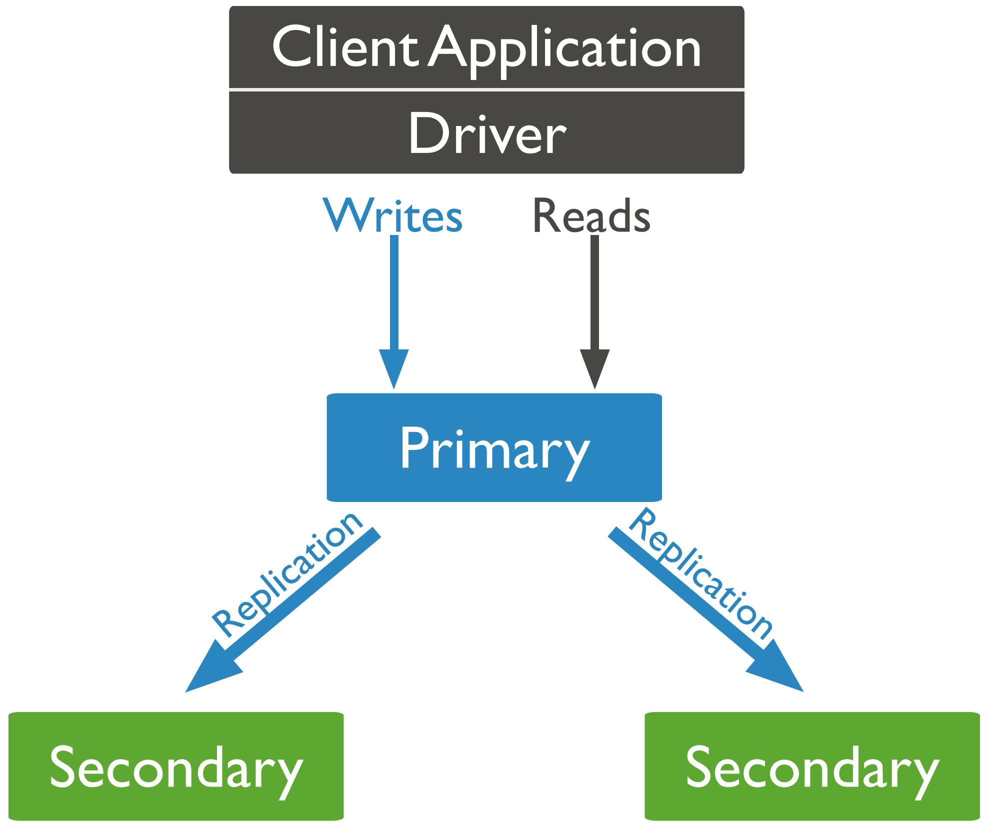
0

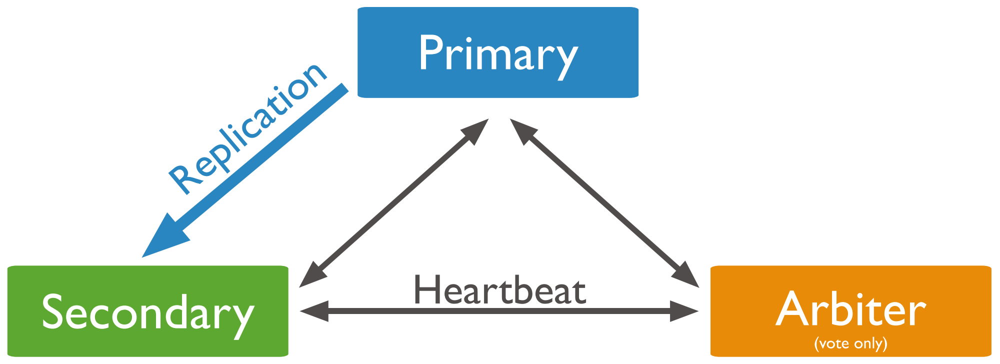
MONGODB REPLICATION

KEY CONCEPTS

- A replica set is a group of *mongod* instances that maintain the same data set:
  - 1 primary node
  - several secondary node
  - 1 arbiter
- Primary node
  - receives all write operations
  - confirming writes with { *w: "majority"* } write concern
- Secondary node
  - replicates the primary's oplog and apply the operations to their data sets
  - if the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary
  - secondaries may have additional configurations for special usage profiles. For example, secondaries may be non-voting or priority 0
- Arbiters
  - do not maintain a data set
  - maintain a quorum in a replica set by responding to heartbeat and election requests by other replica set members

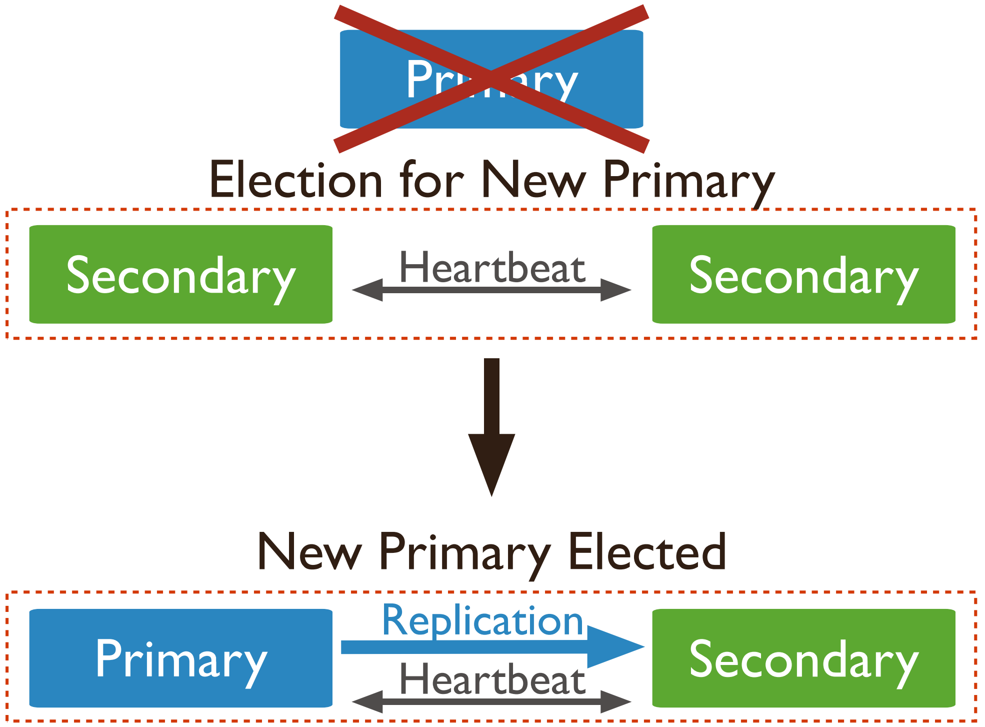
ARCHITECTURE





AUTOMATIC FAILOVER

- When a primary does not communicate with the other members of the set for more than the configured *electionTimeoutMillis* period (10 seconds by default)
- The replica set cannot process write operations until the election completes successfully
- The replica set can continue to serve read queries if such queries are configured to run on secondaries while the primary is offline
- The median time for primary election should not typically exceed 12 seconds

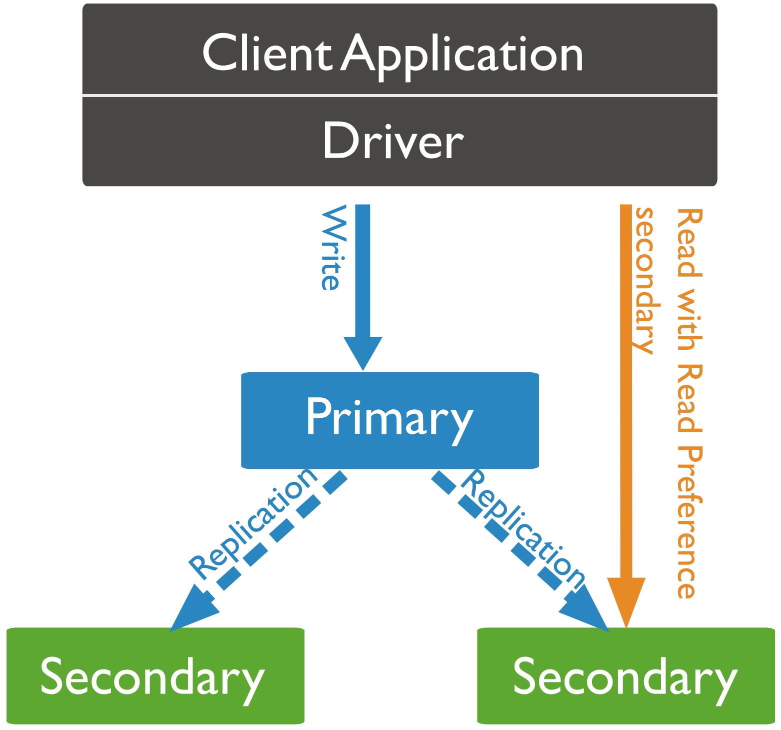


FAULT TOLERANCE

Number of Members	Majority Required to Elect a New Primary	Fault Tolerance
3	2	1
4	3	1
5	3	2
6	4	2

READ OPERATIONS

- By default, clients read from the primary
- Asynchronous replication to secondaries means that reads from secondaries may return data that does not reflect the state of the data on the primary
- Multi-document transactions that contain read operations must use read preference primary. All operations in a given transaction must route to the same member
- Until a transaction commits, the data changes made in the transaction are not visible outside the transaction



DEPLOY A REPLICA SET

- Three member replica sets provide enough redundancy to survive most network partitions and other system failures
- These sets also have sufficient capacity for many distributed read operations
- Replica sets should always have an odd number of members to ensure that elections will proceed smoothly
- Maintain as much separation between members as possible by hosting the mongod instances on separate machines
- Place each mongod instance on a separate host server serviced by redundant power circuits and redundant network paths
- Install MongoDB on each system that will be part of your replica set

CONSIDERATIONS

- Architecture
  - deploy each member to its own machine
  - if possible bind to the standard port *27017*
- Hostnames
  - use a logical DNS hostname instead of an ip address
- IP Binding
  - use the *bind_ip* option to ensure that MongoDB listens for connections from applications on configured addresses
  - *mongod --bind_ip localhost,My-Hostname*
- *Connectivity*
  - establish a virtual private network
  - configure access control
  - configure networking and firewall rules

MEMBERS CONFIGURATION

- Set *replication.replSetName* option to the replica set name
- Set *net.bindIp* option to the hostname/ip
- Set any other settings as appropriate for your deployment

Replica Set Member	Hostname
Member 0	mongodb0.example.net
Member 1	mongodb1.example.net
Member 2	mongodb2.example.net

```
mongod --replSet "rs0" --bind_ip localhost,<hostname(s)|ip address(es)>
```

INITIATE THE REPLICA SET

- Connect a mongo shell to one of the mongod instances

```
mongo
```

- Initiate the replica set

```
rs.initiate( {
            _id : "rs0",
            members: [
                        { _id: 0, host: "mongodb0.example.net:27017" },
                        { _id: 1, host: "mongodb1.example.net:27017" },
                        { _id: 2, host: "mongodb2.example.net:27017" }
            ]
})
```

VERIFY REPLICA SET

- View the replica set configuration

```
rs.conf()
```

- Ensure that the replica set has a primary

```
rs.status()
```

CONFIGURATION INFO

```
{ "_id" : "rs0",
"version" : 1,
"protocolVersion" : NumberLong(1),
"members" : [
                { "_id" : 0, "host" :
                "mongodb0.example.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")
}}
```

DEPLOY A REPLICA SET FOR TESTING (1)

- Create the necessary data directories for each member

```
mkdir -p /srv/mongodb/rs0-0 /srv/mongodb/rs0-1 /srv/mongodb/rs0-2
```

- Start your mongod instances in their own shell windows

```
1) mongod --replSet rs0 --port 27017 --bind_ip localhost,<hostname(s)|ip address(es)> --dbpath
/srv/mongodb/rs0-0  --oplogSize 128
2) mongod --replSet rs0 --port 27018 --bind_ip localhost,<hostname(s)|ip address(es)> --dbpath
/srv/mongodb/rs0-1  --oplogSize 128
3) mongod --replSet rs0 --port 27019 --bind_ip localhost,<hostname(s)|ip address(es)> --dbpath
/srv/mongodb/rs0-2 --oplogSize 128
```

DEPLOY A REPLICA SET FOR TESTING (2)

- Connect to one of your mongod instances through the mongo shell

```
mongo --port 27017
```

- Initiate the replica set

```
rsconf = {
            _id: "rs0",
            members: [
                        { _id: 0, host: "<hostname>:27017" },
                        { _id: 1, host: "<hostname>:27018" },
                        { _id: 2, host: "<hostname>:27019" }
            ]
 }
rs.initiate( rsconf )
```

ADD MEMBERS

- Start the new mongod instance

```
mongod --dbpath /srv/mongodb/db0 --replSet rs0  --bind_ip localhost,<hostname(s)|ip address(es)>
```

- Connect to the replica set’s primary
- Add the new member to the replica set

```
rs.add( { host: "mongodb3.example.net:27017", priority: 0, votes: 0 } )
```

- Ensure that the new member has reached *SECONDARY* state
- Update the newly added member’s priority and votes if needed

```
var cfg = rs.conf();
cfg.members[4].priority = 1
cfg.members[4].votes = 1
rs.reconfig(cfg)
```

REMOVE MEMBERS

- Shut down the mongod instance for the member you wish to remove
- Connect to the replica set's current primary
- Use *rs.remove()*

```
rs.remove("mongod3.example.net:27017")

rs.remove("mongod3.example.net")
```