Replication in MongoDB
Key Concepts

• A replica set is a group of *mongod* instances that maintain the same data set (redundancy):
  o **1 primary** node, and **only one**
  o **secondary** nodes (all the other nodes containing a copy of the data)
  o **1 arbiter** (optional)

• Primary node
  o receives all **write** operations
  o confirming writes with `{ w: "majority" } write concern`, i.e., the number of data-bearing members (primary and secondaries, but not arbiters) that must acknowledge a write operation before the operation returns as successful

• Secondary node
  o replicates the **primary’s oplog** and apply the operations to their data sets
  o if the primary is unavailable, an eligible secondary will hold an **election** to elect itself the new primary
  o secondaries may have additional configurations for special usage profiles. For example, secondaries may be **non-voting** or with different **priority** levels

• Arbiters
  o does **not** hold **data**! (will always be an arbiter, cannot be elected primary)
  o maintains a quorum in a replica set by responding **election** requests by other replica set members (and keeping the heartbeat)
Architecture

Client Application
  Driver
    Writes  Reads
  Primary
    Replication  Replication
  Secondary
  Secondary

Primary
  Replication
  Secondary
  Heartbeat
  Arbiter (one only)
Write concern

For write concern of
- $w > 1$ or
- $w: \text{"majority"}$
the primary waits until the required number of secondaries acknowledge the write before returning write concern acknowledgment.

For write concern of $w: 1$, the primary can return write concern acknowledgment as soon as it locally applies the write.
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

<table>
<thead>
<tr>
<th>Number of Members</th>
<th>Majority Required to Elect a New Primary</th>
<th>Fault Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
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
  o deploy each member to its own machine
  o if possible, bind to the standard port 27017

• Hostnames
  o use a logical DNS hostname instead of an ip address

• IP Binding
  o use the bind_ip option to ensure that MongoDB listens for connections from applications on configured addresses
    o `mongod --bind_ip localhost,My-Hostname`

• Connectivity
  o establish a virtual private network
  o configure access control
  o 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

<table>
<thead>
<tr>
<th>Replica Set Member</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member 0</td>
<td>mongodbo.example.net</td>
</tr>
<tr>
<td>Member 1</td>
<td>mongodb1.example.net</td>
</tr>
<tr>
<td>Member 2</td>
<td>mongodb2.example.net</td>
</tr>
</tbody>
</table>

`mongod --replSet "rs0" --bind_ip localhost,<hostname(s) | ip address(es)>`
Deploy a Replica Set for Testing (1)

• Create the necessary data directories for each member

```bash
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()`

```javascript
rs.remove("mongod3.example.net:27017")
rs.remove("mongod3.example.net")
```
Verify replica set

• View the replica set configuration

   rs.conf()

• Ensure that the replica set has a primary

   rs.status()
```json
{
"_id": "rs0",
"version": 1,
"protocolVersion": NumberLong(1),
"members": [
    {
        "_id": 0,
        "host": "mongodb0.example.net:27017",
        "arbiterOnly": false,
        "buildIndexes": true,
        "hidden": false,
        "priority": 1, // higher values to make a member more eligible to become primary, zero is ineligible to become primary.
        "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")
}
}
```