REPLICATION

Unit 4 - Replication in MongoDB

Introduction

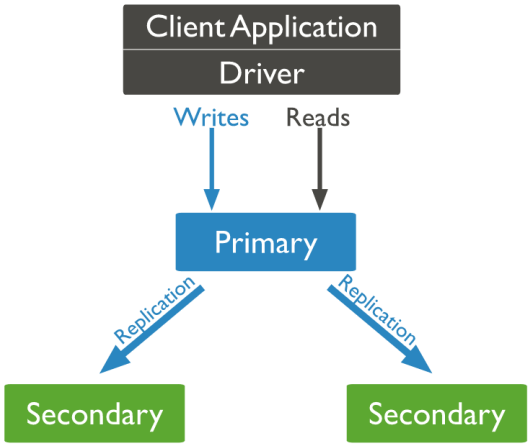
- A replica set in MongoDB is a group of mongodb processes that maintain the same data set.
- Replica sets provide redundancy and high availability and are the basis for all production deployments.
- What is replication?
  - Replication is the process of copying data across multiple servers.
- Why is replication required?
  - Fault Tolerance
    - To keep data safe and manage disaster recovery o
  - High (24*7) availability of data providing read scaling (extra copies to read from) (specially for distributed systems)
  - No downtime for maintenance (like backups, index rebuilds, compaction)

Redundancy and Data Availability

- Replication provides redundancy and increases data availability.
- With multiple copies of data on different database servers, replication provides a level of fault tolerance against the loss of a single database server.
- Replication can also provide increased read capacity as clients can send read operations to different servers.
- Maintaining copies of data in different data centers can increase data locality and availability for distributed applications.
- Additional copies can be maintained for dedicated purposes, such as disaster recovery, reporting, or backup.

Replication in MongoDB

- A replica set is a group of mongodb instances that maintain the same data set.
- A replica set contains several data bearing nodes and optionally one arbiter node.
- Of the data bearing nodes, one and only one member is deemed the primary node, while the other nodes are deemed secondary nodes.



Members of a Replica Set

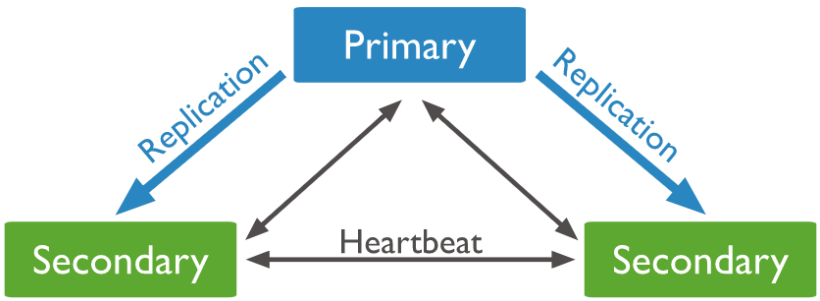
- δ Primary
- δ Secondary
- δ Arbiter (optional)

Primary node

- δ The primary node receives all write operations.
- δ A replica set can have only one primary capable of confirming writes with {w: "majority"} write concern; although in some circumstances, another mongod instance may transiently believe itself to also be primary.
- δ The primary records all changes to its data sets in its operation log, i.e. oplog.

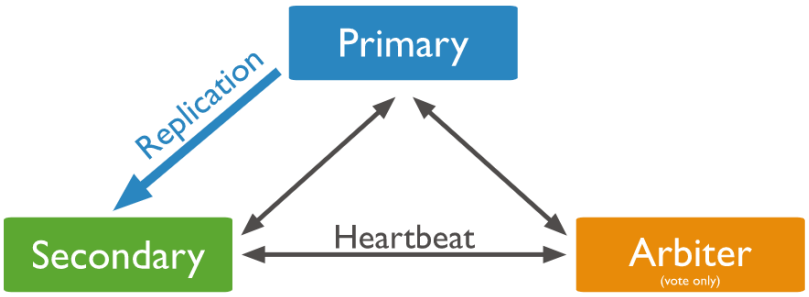
Secondary node

- δ The secondary nodes replicate the primary's oplog and apply the operations to their data sets such that the secondaries' data sets reflect the primary's data set.
- δ If the primary is unavailable, an eligible secondary will hold an election to elect itself the new primary.



Arbiter Node

- δ An extra mongodb instance can be added to a replica set as an arbiter.
- δ Arbiters do not maintain a data set.
- δ The purpose of an arbiter is to maintain a quorum in a replica set by responding to heartbeat and election requests by other replica set members.
- δ Arbiters do not store a data set thus they can be a good way to provide replica set quorum functionality with a cheaper resource cost than a fully functional replica set member with a data set.
- δ If the replica set has an even number of members, add an arbiter to obtain a majority of votes in an election for primary.
- δ Arbiters do not require dedicated hardware.
- δ An arbiter will always be an arbiter whereas a primary may step down and become a secondary and a secondary may become the primary during an election.



Asynchronous Replication

- δ Secondary nodes apply operations from the primary asynchronously.
- δ By applying operations after the primary, sets can continue to function despite the failure of one or more members.

Automatic Failover

- δ When a primary does not communicate with the other members of the set for more than 10 seconds, an eligible secondary will hold an election to elect itself the new primary.

- The first secondary to hold an election and receive a majority of the members' votes becomes primary.
- Although the timing varies, the failover process generally completes within a minute.
  - For instance, it may take 10-30 seconds for the members of a replica set to declare a primary inaccessible.
  - One of the remaining secondary holds an election to elect itself as a new primary. The election itself may take another 10-30 seconds.

Read Operations

- By default, clients read from the primary; however, clients can specify a read preference to send read operations to secondaries.
- Asynchronous replication to secondaries means that reads from secondaries may return data that does not reflect the state of the data on the primary.
- In MongoDB, clients can see the results of writes before the writes are durable:
  - Regardless of write concern, other clients using "local" (i.e. the default) readConcern can see the result of a write operation before the write operation is acknowledged to the issuing client.
  - Clients using "local" (i.e. the default) readConcern can read data which may be subsequently rolled back.
- Replica sets provide a number of options to support application needs.
- Example
  - A replica set can be deployed with members in multiple data centers or control the outcome of elections by adjusting the members' priority of some members.
  - Replica sets also support dedicated members for reporting, disaster recovery, or backup functions.
  - A secondary member can be configured for a specific purpose.

Configuration of nodes

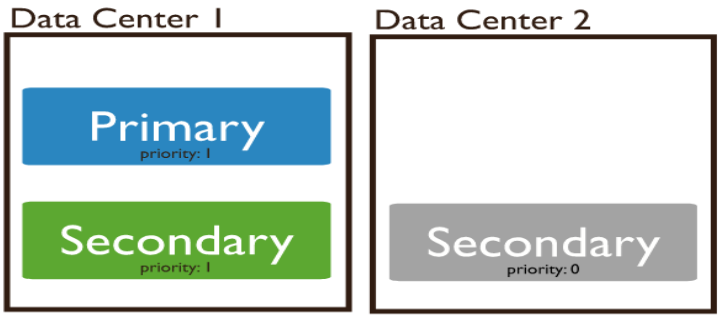
- Priority 0 Replica Set Members
  - Prevent it from becoming a primary in an election, which allows it to reside in a secondary data center or to serve as a cold standby.
- Hidden Replica Set Members
  - Prevent applications from reading from it, which allows it to run applications that require separation from normal traffic.
- Delayed Replica Set Members
  - Keep a running "historical" snapshot for use in recovery from certain errors, such as unintentionally deleted databases.

Replica set – Requirements

- δ The minimum requirements for a replica set may be:
  - ϖ A primary, a secondary, and an arbiter.
- δ Most deployments, however, will keep three members that store data
  - ϖ Primary and two secondary members
  - ϖ A replica set can have up to 50 members but only 7 voting members.
- δ In previous versions, replica sets can have up to 12 members.

Priority 0 Replica Set Members

- δ A priority 0 member is a secondary that cannot become primary.
- δ Priority 0 members cannot trigger elections.
- δ Otherwise these members function as normal secondaries.
- δ A priority 0 member maintains a copy of the data set, accepts read operations, and votes in elections.
- δ Configure a priority 0 member to prevent secondaries from becoming primary, which is particularly useful in multi-data center deployments.
- δ In a three-member replica set, in one data center hosts the primary and a secondary.
- δ A second data center hosts one priority 0 member that cannot become primary.



Priority 0 Members as Standbys

- δ A priority 0 member can function as a standby.
- δ In some replica sets, it might not be possible to add a new member in a reasonable amount of time.
- δ A standby member keeps a current copy of the data to be able to replace an unavailable member
- δ In many cases, we need not set standby to priority 0.
- δ However, in sets with varied hardware or geographic distribution, a priority 0 standby ensures that only qualified members become primary.
- δ A priority 0 standby may also be valuable for some members of a set with different hardware or workload profiles.

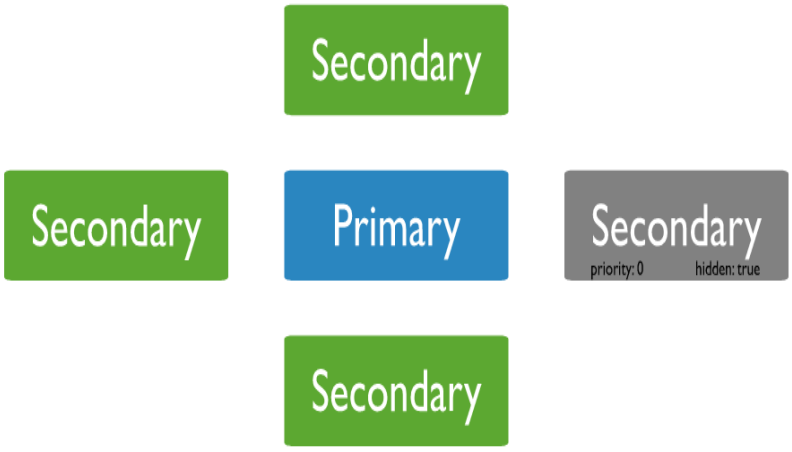
- In these cases, deploy a member with priority 0 so it can't become primary. (hidden member)

Priority 0 Members and Failover

- When configuring a priority 0 member, consider potential failover patterns, including all possible network partitions.
- Always ensure that our main data center contains both a quorum of voting members and contains members that are eligible to be primary.

Hidden Replica Set Members

- A hidden member maintains a copy of the primary's data set but is invisible to client applications.
- Hidden members are good for workloads with different usage patterns from the other members in the replica set.
- Hidden members must always be priority 0 members and so cannot become primary.
- The db.isMaster() method does not display hidden members.
- Hidden members, however, may vote in elections.



Read Operations – Hidden

- Clients will not distribute reads to hidden members.
- As a result, these members receive no traffic other than basic replication.
- Use hidden members for dedicated tasks such as reporting and backups.
- Delayed members should be hidden.

Voting – Hidden

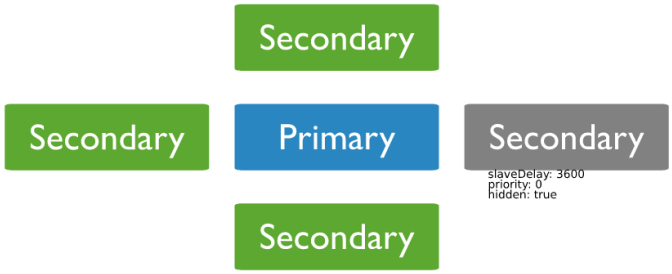
- Hidden members may vote in replica set elections.
- If we stop a voting hidden member, ensure that the set has an active majority or the primary will step down.

Delayed Replica Set Members

- Delayed members contain copies of a replica set's data set.
- However, a delayed member's data set reflects an earlier, or delayed, state of the set.
  - For example, if the current time is 09:52 and a member has a delay of an hour, the delayed member has no operation more recent than 08:52.
- Since delayed members are a "rolling backup" or a running "historical" snapshot of the data set, they may help to recover from various kinds of human error.
  - For example, a delayed member can make it possible to recover from unsuccessful application o upgrades and operator errors including dropped databases and collections

Delayed members

- Must be priority 0 members.
- Set the priority to 0 to prevent a delayed member from becoming primary.
- Should be hidden members.
- Always prevent applications from seeing and querying delayed members.
- Do vote in elections for primary.



Replica Set Deployment Architectures

- The standard replica set deployment for production system is a three-member replica set.
- These sets provide redundancy and fault tolerance.
- Avoid complexity when possible but let application requirements dictate the architecture.
- Two important points to take care
  - Determine the Number of Members
  - Determine the Distribution of Member

Determine the Number of Members

- Deploy an Odd Number of Members
  - An odd number of members ensure that the replica set is always able to elect a primary.
  - If you have an even number of members, add an arbiter to get an odd number.
  - Arbiters do not store a copy of the data and require fewer resources.
- Consider Fault Tolerance
  - Fault tolerance for a replica set is the number of members that can become unavailable
  - It is the difference between the number of members in the set and the majority needed to elect a primary.
  - Without a primary, a replica set cannot accept write operations.
  - Fault tolerance is an effect of replica set size, but the relationship is not direct.
  - Adding a member to the replica set does not *always* increase the 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

- Use Hidden and Delayed Members for Dedicated Function
  - Add hidden or delayed members to support dedicated functions, such as backup or reporting.
- Load Balance on Read-Heavy Deployments
  - In a deployment with very high read traffic, the read throughput can be improved by distributing reads to secondary members.
  - As the deployment grows, add or move members to alternate data centers to improve o redundancy and availability.
- Add Capacity Ahead of Demand
  - The existing members of a replica set must have spare capacity to support adding a new o member.
  - Always add new members before the current demand saturates the capacity of the set.

Determine the Distribution of Members

- δ Distribute Members Geographically
  - ϖ To protect our data if our main data center fails, keep at least one member in an alternate data center.
  - ϖ Set these members' members[n].priority to 0 to prevent them from becoming primary.
- δ Keep a Majority of Members in One Location
  - ϖ When a replica set has members in multiple data centers, network partitions can prevent communication between data centers.
  - ϖ To replicate data, members must be able to communicate to other members.
  - ϖ In an election, members must see each other to create a majority.
  - ϖ To ensure that the replica set members can confirm a majority and elect a primary, keep a majority of the set's members in one location.

Deployment Patterns

- δ Are of three kinds
  - ϖ Three Member Replica Sets
  - ϖ Replica Sets with Four or More Members
  - ϖ Geographically Distributed Replica Sets
- δ Three Member Replica Sets
  - ϖ Three-member replica sets provide the minimum recommended architecture for a replica set.
- δ Replica Sets with Four or More Members
  - ϖ Four or more members replica sets provide greater redundancy and can support greater distribution of read operations and dedicated functionality.
- δ Geographically Distributed Replica Sets
  - ϖ Geographically distributed sets include members in multiple locations to protect against facility-specific failures, such as power outages.

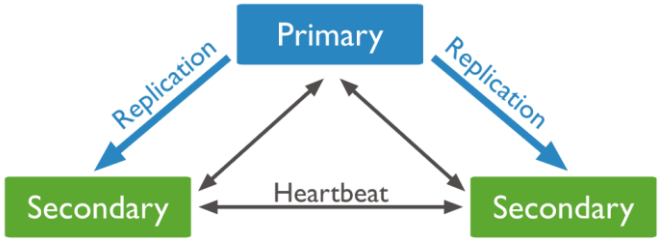
Three Member Replica Sets

- δ The minimum architecture of a replica set has three members.
- δ A three-member replica set can have either
  - ϖ three members that hold data,
  - ϖ or two members that hold data and an arbiter.

Primary with Two Secondary Members

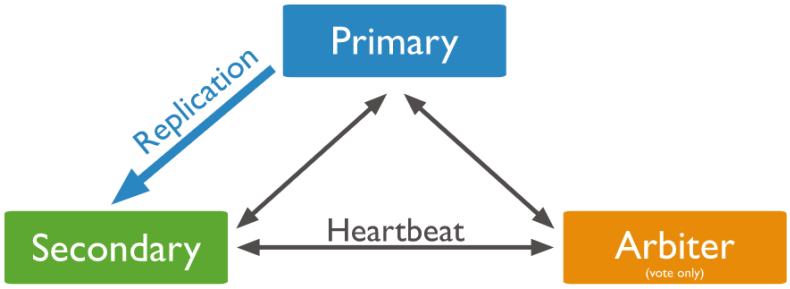
- δ A replica set with three members that store data has
  - ϖ One primary.

- Two secondary members.
  - Both secondaries can become the primary in an election.



Primary with a Secondary and an Arbiter

- A three-member replica set with two members that store data has:
  - One primary.
  - One secondary member.
    - The secondary can become primary in an election.
  - One arbiter.
    - The arbiter only votes in elections.



Replica Sets with Four or More Members

- Although the standard replica set configuration has three members, we can deploy larger sets.
- Add additional members to a set to increase redundancy or to add capacity for distributing secondary read operations.

- While adding new members to a replica set, consider the following
  - Odd Number of Voting Members

- χ Ensure that the replica set has an odd number of voting members.
- χ If we have an even number of voting members, deploy an arbiter so that the set has an odd o number.
- χ Example
  - ⇒ The following replica set includes an arbiter to ensure an odd number of voting members.

- δ A replica set can have up to 50 members, but only 7 voting members.
- δ If the replica set already has 7 voting members, additional members must be non-voting members
- δ Example
  - ϖ The following 9 members replica set has 7 voting members and 2 non-voting members.

Location of the Members

- δ A majority of the replica set's members should be in the application's main data center.

Write concern

- δ { w: <value>, j: <boolean>, wtimeout: <number> }
  - ϖ the w option to request acknowledgement that the write operation has propagated to a specified number of mongod instances or to mongod instances with specified tags.
  - ϖ the j option to request acknowledgement that the write operation has been written to the journal, and

- ϖ the wtimeout option to specify a time limit to prevent write operations from blocking indefinitely.

δ w: 1

- ϖ Requests acknowledgement that the write operation has propagated to the standalone mongod or the primary in a replica set.
- ϖ w: 1 is the default write concern for MongoDB.

δ w: 0

- ϖ Requests no acknowledgement of the write operation.
- ϖ However, w: 0 may return information about socket exceptions and networking errors to the application.

δ w:"majority"

- ϖ Requests acknowledgment that write operations have propagated to the majority of voting nodes, including the primary, and have been written to the on-disk journal for these nodes.

δ If w: 0 is specified but include j: true, the j: true prevails to request acknowledgement from the standalone mongod or the primary of a replica set.

δ Numbers greater than 1 are valid only for replica sets to request acknowledgement from specified number of members, including the primary.

Read concern

δ The readConcern query option for replica sets and replica set shards determines which data to return from a query.

Level	Description
"local"	Default. The query returns the instance's most recent data. Provides no guarantee that the data has been written to a majority of the replica set members (i.e. may be rolled back).
"majority"	The query returns the instance's most recent data acknowledged as having been written to a majority of members in the replica set.
"linearizable"	The query returns data that reflects all successful writes issued with a write concern of **"majority"** *and* acknowledged prior to the start of the read operation.

Oplog

- The oplog exists internally as a capped collection, so we cannot modify its size in the course of normal operations.
- In most cases the default oplog size is an acceptable size; however, in some situations we may need a larger or smaller oplog.
- Example
  - There may be a need to change the oplog size if the applications perform large numbers of multi-updates or deletes in short periods of time.
- The size of the oplog by default is
  - For 64-bit Linux, Solaris, FreeBSD, and Windows systems, MongoDB allocates 5% of the available free disk space, but will always allocate at least 1 gigabyte and never more than 50 gigabytes.
  - To view oplog status, including the size and the time range of operations, rs.printReplicationInfo() can be issued.

--smallfiles

- Sets MongoDB to use a smaller default file size.
- The smallfiles option reduces the initial size for data files and limits the maximum size to 512 megabytes.
- smallfiles also reduces the size of each journal file from 1 gigabyte to 128 megabytes.
  - Use --smallfiles if you have a large number of databases that each holds a small quantity of data.
  - The --smallfiles option can lead the mongod instance to create a large number of files, which can affect performance for larger databases.

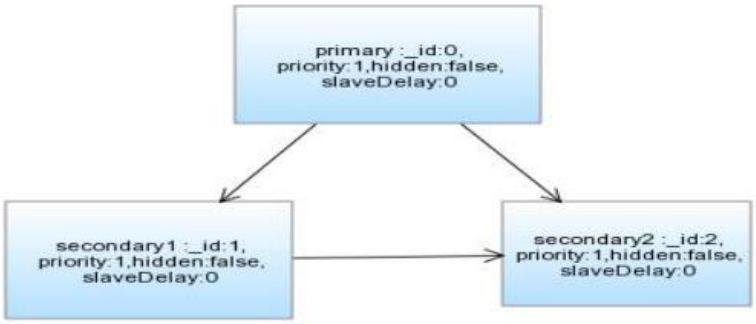
Type of attributes and commands

- members[n]._id
- members[n].host
- members[n].arbiterOnly Default: false
- members[n].hidden Default: false
- members[n].priority Default: 1.0
- members[n].slaveDelay Default: 0
- members[n].votes Default: 1

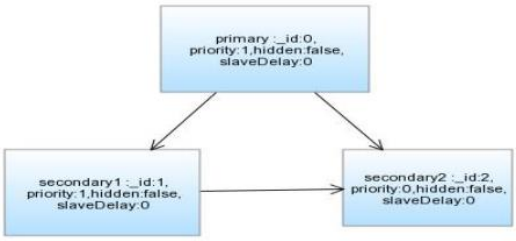
Type of nodes and attributes

Type of Node	Attributes							
	Priority	Readability	Writability	Capable of triggering election	Voting right	Hidden	Delayed	Availability of data set
Primary	1	✓	✓	NA	NA	X	X	✓
Secondary (normal)	1	✓	X	✓	✓	X	X	✓
Secondary (priority:0)	0	✓	X	X	✓	X	X	✓
Secondary (hidden)	0	X	X	X	✓	✓	X	✓
Secondary (delayed)	0	X	X	X	✓	✓	✓	✓(Stale Data)
Arbiter	NA	NA	NA	NA	✓	NA	NA	NA

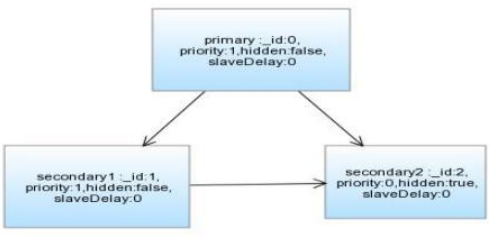
Default configuration of replication set members



- δ To set the priority of any of the member of the replication set test to "0"
- δ Example - member="_id:2"
  - ϖ cfg=rs.conf()
  - ϖ cfg.members[2].priority=0
  - ϖ rs.reconfig[cfg]



- δ To set the hidden property of any of the member of the replication set to "true"
- δ Example: member="_id:2"
  - ϖ cfg=rs.conf()
  - ϖ cfg.members[2].priority=0
  - ϖ cfg.members[2].hidden=true



- ϖ rs.reconfig[cfg]

δ To set the slaveDelay property of any of the member of the replication set to a specific time duration

δ Example:

member=”_id:2”,time=60seconds

- ϖ cfg=rs.conf()
- ϖ cfg.members[2].priority=0
- ϖ cfg.members[2].hidden=true
- ϖ cfg.members[2].slaveDelay=60
- ϖ rs.reconfig[cfg]