Creating Replica set in mongodb

ANJU MUNOTH

Folder requirements

- Create 3 subfolders data1,data2,data3 in a folder called as replicaExample

Terminal1

- Start a command prompt from bin folder and start the mongod instance in the following manner giving the correct dbpath with respect to the folders created in the previous slide

- mongod --port 27017 --dbpath "C:\Users\User\Desktop\MYMATERIAL\mongodb\replicaExample\shard1\data1" --replSet rs1

Terminal2

- Start a command prompt from bin folder and start the mongod instance in the following manner giving the correct dbpath with respect to the folders created in the previous slide

- mongod --port 27018 --dbpath "C:\Users\User\Desktop\MYMATERIAL\mongodb\replicaExample\shard1\data2" --replSet rs1

Terminal 3

- Start a command prompt from bin folder and start the mongod instance in the following manner giving the correct dbpath with respect to the folders created in the previous slide

- mongod --port 27019 --dbpath "C:\Users\User\Desktop\MYMATERIAL\mongodb\replicaExample\shard1\data3" --replSet rs1

In 4th terminal

- **Connect to One of the replica Server to Enable ReplicateSet**
- mongo --host localhost --port 27017
- Type the above command in a new command prompt by opening it from bin folder

Creating the replica sets

- Once query is executed, anyone of the nodes will be elected as Primary by replica-set using round-bin.

```
rs.initiate(
{
_id: “rs1",
members: [
{ _id : 0, host : "localhost:27017" },
{ _id : 1, host : "localhost:27018" },
{ _id : 2, host : "localhost:27019" }
]
})
```

rs.status()

- Type rs.status() in the 4th terminal
- This will show the various config details as who is the primary and who all are secondary and a lot of other meta information

View the replica set configuration.

- Use rs.conf() to display the replica set configuration object:

```
rs.conf()
```

Add an Arbiter to Replica Set

- Arbiters are mongod instances that are part of a replica set but do not hold data.
- Arbiters participate in elections in order to break ties. If a replica set has an even number of members, add an arbiter.

- Arbiters have minimal resource requirements and do not require dedicated hardware.
- Can deploy an arbiter on an application server or a monitoring host.

Add an Arbiter

- Create a data directory (e.g. storage.dbPath) for the arbiter. The mongod instance uses the directory for configuration data. The directory will not hold the data set. For example, create the /data/arb directory:

  mkdir /data/arb

- Start the arbiter, specifying the data directory and the name of the replica set to join.
- The following starts an arbiter using the /data/arb as the dbPath and rs for the replica set name:

  mongod --port 27017 --dbpath /data/arb --replSet rs --bind_ip localhost

- Connect to the primary and add the arbiter to the replica set.
- Use the rs.addArb() method,

- rs.addArb("localhost:27017")
- This operation adds the arbiter running on port 27017 on the localhost.

Convert a Standalone to a Replica Set

- Shut down the standalone mongod instance.
- Restart the instance. Use the --replSet option to specify the name of the new replica set.
- For example, the following command starts a standalone instance as a member of a new replica set named rs0. The command uses the standalone's existing database path of /srv/mongodb/db0:

mongod --port 27017 --dbpath /srv/mongodb/db0 --replSet rs0

- Connect a mongo shell to the mongod instance.
- Use rs.initiate() to initiate the new replica set:

rs.initiate()

- The replica set is now operational. To view the replica set configuration, use rs.conf(). To check the status of the replica set, use rs.status().

Add Members to a Replica Set

Step 0:Prepare the Data Directory

- Before adding a new member to an existing replica set, prepare the new member's data directory using one of the following strategies:
- Make sure the new member's data directory does not contain data. The new member will copy the data from an existing member.
- If the new member is in a recovering state, it must exit and become a secondary before MongoDB can copy all data as part of the replication process. This process takes time but does not require administrator intervention.
- Manually copy the data directory from an existing member. The new member becomes a secondary member and will catch up to the current state of the replica set. Copying the data over may shorten the amount of time for the new member to become current.
- Ensure that you can copy the data directory to the new member and begin replication within the window allowed by the oplog. Otherwise, the new instance will have to perform an initial sync, which completely resynchronizes the data, as described in Resync a Member of a Replica Set.
- Use rs.printReplicationInfo() to check the current state of replica set members with regards to the oplog.

Add Members to a Replica Set

- Start the new mongod instance. Specify the data directory and the replica set name. The following example specifies the /srv/mongodb/db0 data directory and the rs0 replica set:

  mongod --dbpath /srv/mongodb/db0 --replSet rs0

- Connect to the replica set's primary.
- Can only add members while connected to the primary. If you do not know which member is the primary, log into any member of the replica set and issue the db.isMaster() command.
- Use rs.add() to add the new member to the replica set.
- Pass the member configuration document to the method. For example, to add a member , issue the following command:

  rs.add({ host: "localhost:27017", priority: 0, votes: 0 })

- Ensure that the new member has reached SECONDARY state. To check the state of the replica set members, run rs.status():

  rs.status()

Priority 0 Secondary

- A priority 0 member is a member that cannot become primary and cannot trigger elections.
- Priority 0 members can acknowledge write operations issued with write concern of w : <number>.
- For "majority" write concern, the priority 0 member must also be a voting member (i.e. members[n].votes is greater than 0) to acknowledge the write.
- Non-voting replica set members (i.e. members[n].votes is 0) cannot contribute to acknowledging write operations with "majority" write concern.
- **Other than the aforementioned restrictions, secondaries that have priority 0 function as normal secondaries: they maintain a copy of the data set, accept read operations, and vote in elections**

Priority 0 Members as Standbys

- A priority 0 member can function as a standby. In some replica sets, it might not be possible to add a new member in a reasonable amount of time. A standby member keeps a current copy of the data to be able to replace an unavailable member.
- In many cases, you need not set standby to priority 0. However, in sets with varied hardware or geographic distribution, a priority 0 standby ensures that only qualified members become primary.
- 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.
- If your set already has seven voting members, also configure the member as non-voting.

Priority 0 Members as Standbys

- In a replica set, by default all secondary members are eligible to become primary through the election process.
- Can use the priority to affect the outcome of these elections by making some members more likely to become primary and other members less likely or unable to become primary.
- To prevent a secondary member from ever becoming a primary in a failover, assign the secondary a priority of 0

Steps to create a 0 priority secondary

Step1:

- Retrieve the current replica set configuration.
- The rs.conf() method returns a replica set configuration document that contains the current configuration for a replica set.
- In a mongo shell connected to a primary, run the rs.conf() method and assign the result to a variable:

cfg = rs.conf()

- The returned document contains a members field which contains an array of member configuration documents, one document for each member of the replica set.

Steps to create a 0 priority secondary

Step2:

- Assign priority value of 0.
- To prevent a secondary member from becoming a primary, update the secondary member's members[n].priority to 0.
- To assign a priority value to a member of the replica set, access the member configuration document using the array index. If the secondary member to change corresponds to the configuration document found at position 2 of the members array.

cfg.members[2].priority = 0

- The configuration change does not take effect until you reconfigure the replica set.

Steps to create a 0 priority secondary

Step 3:

- Reconfigure the replica set.
- Use rs.reconfig() method to reconfigure the replica set with the updated replica set configuration document.
- Pass the cfg variable to the rs.reconfig() method:

rs.reconfig(cfg)

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.

Hidden members: Behaviour

Read Operations

- Clients will not distribute reads with the appropriate read preference 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.
- In a sharded cluster, mongos do not interact with hidden members.

Voting

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

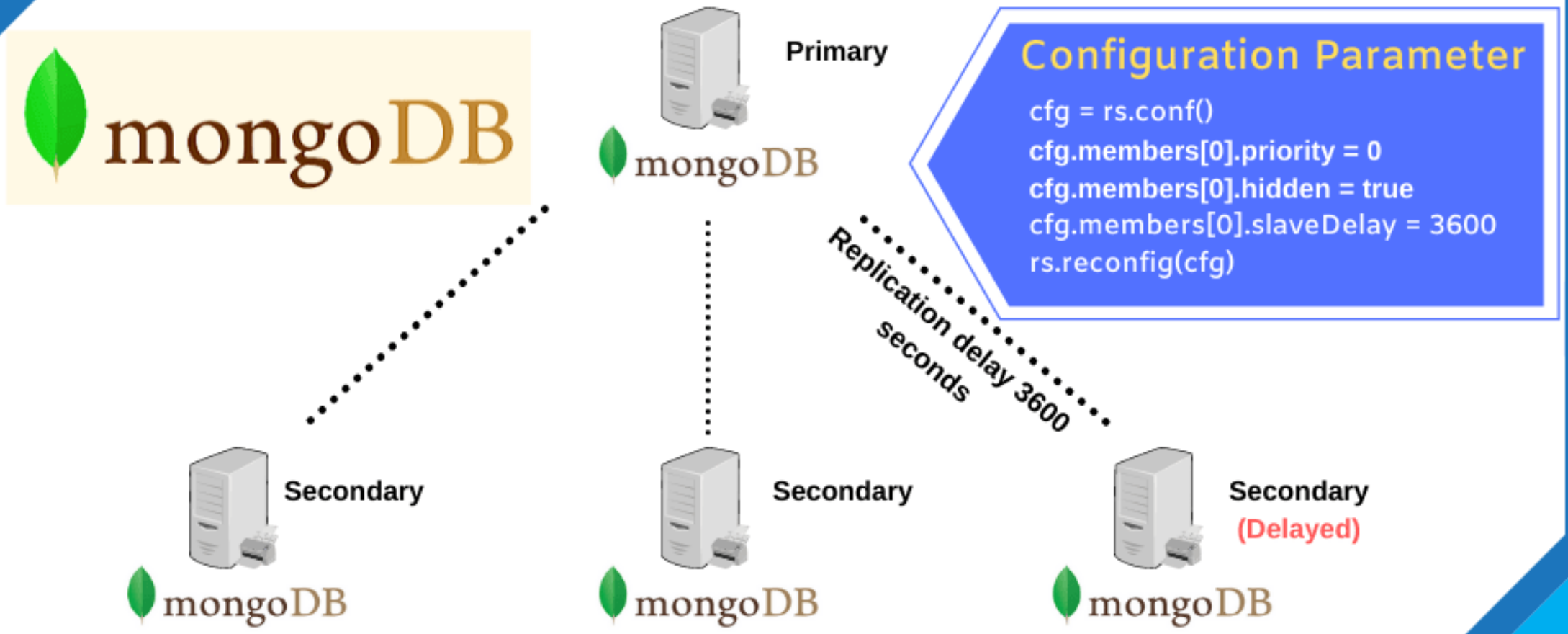
Delayed Replica Set Members

- 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.
- Because delayed members are a "rolling backup" or a running "historical" snapshot of the data set, they may help you recover from various kinds of human error.
- For example, a delayed member can make it possible to recover from unsuccessful application 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, if members[n].votes is set to 1.
- Delayed members copy and apply operations from the source oplog on a delay. When choosing the amount of delay, consider that the amount of delay:
  - must be equal to or greater than your expected maintenance window durations.
  - must be smaller than the capacity of the oplog.

Delayed Replica Set Member in MongoDB



Why a delayed replica set member?

- Safeguard the data from human mistakes like wrong code push on the primary. Accidental data insert or update.
- For backup and recovery purposes when you get a lot of concurrent data changes happen from the primary. At any given point in if you have to revert any specific data set.
- Can used for point in time recovery.
- Can also be used while making BCP (Bussines Continuity Plan) setup.

How to configure it?

- From primary node while adding a new delayed node just add ‘cfg.members[0].slaveDelay = 3600’ where 3600 demotes the seconds.

- cfg = rs.conf();// Start the confiuration
- cfg.members[0].priority = 0 ;// Seting the node priority so that it never become primary
- cfg.members[0].hidden = true ;// It Can’t vote in node eletion process
- cfg.members[0].slaveDelay = 3600;// Delay in seconds
- rs.reconfig(cfg);// Save the configuration

Replica Set Oplog

- The oplog (operations log) is a special capped collection that keeps a rolling record of all operations that modify the data stored in your databases.
- MongoDB applies database operations on the primary and then records the operations on the primary's oplog.
- The secondary members then copy and apply these operations in an asynchronous process.
- All replica set members contain a copy of the oplog, in the local.oplog.rs collection, which allows them to maintain the current state of the database.
- To facilitate replication, all replica set members send heartbeats (pings) to all other members.
- Any member can import oplog entries from any other member.
- Each operation in the oplog is idempotent. That is, oplog operations produce the same results whether applied once or multiple times to the target dataset.

Oplog Size

- When you start a replica set member for the first time, MongoDB creates an oplog of a default size.
- For Unix and Windows systems
- The default oplog size depends on the storage engine:

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

Oplog size

- Before mongod creates an oplog, can specify its size with the oplogSizeMB option.
- However, after you have started a replica set member for the first time, you can only change the size of the oplog using the Change the Size of the Oplog procedure.

Oplog Status

- To view oplog status, including the size and the time range of operations, issue the rs.printReplicationInfo() method.

- Under various exceptional situations, updates to a secondary's oplog might lag behind the desired performance time.
- Use db.getReplicationInfo() from a secondary member and the replication status output to assess the current state of replication and determine if there is any unintended replication delay.