MongoDB Replication

Overview

How does replication work in MongoDB?

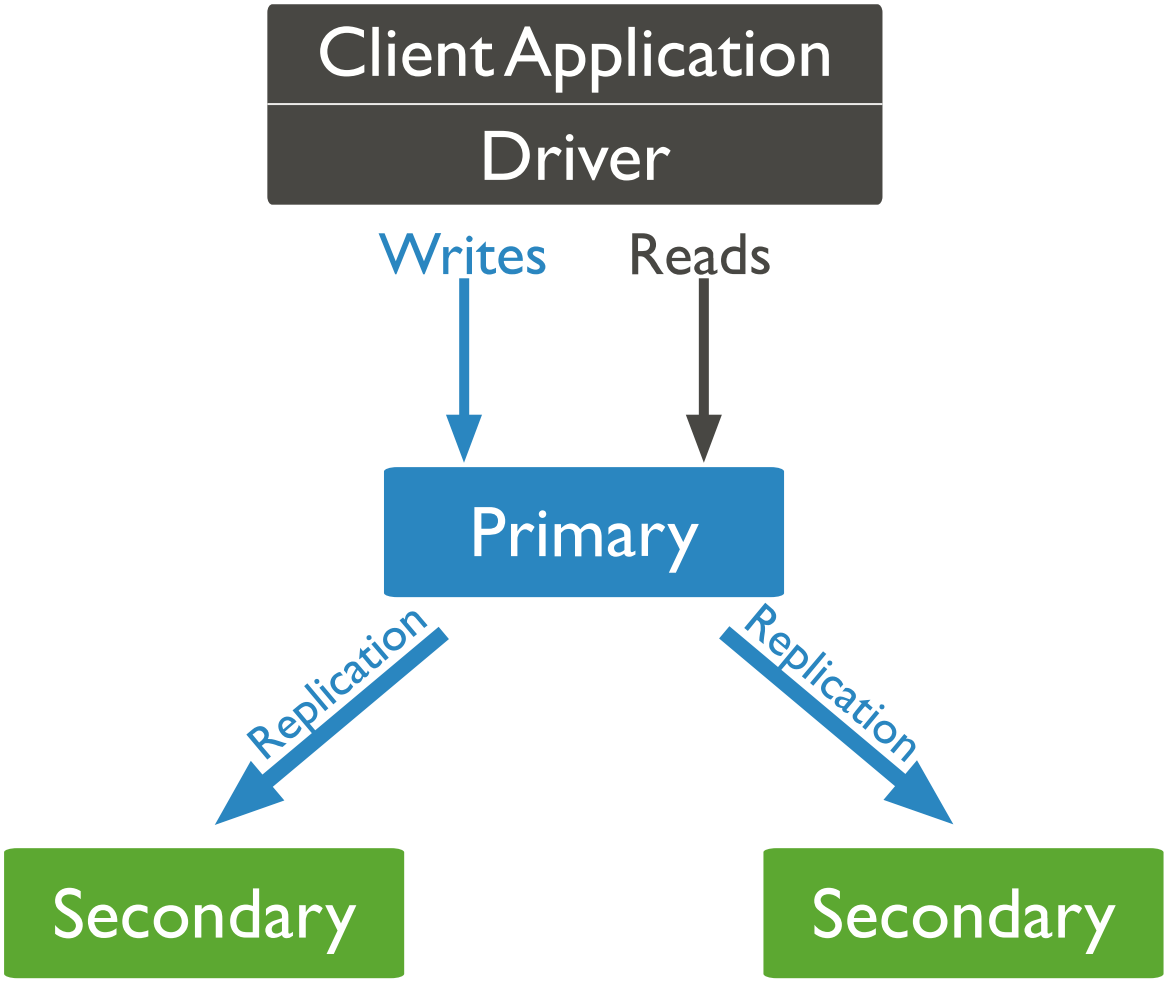
Replication exists primarily to offer data redundancy and high availability. We maintain the durability of data by keeping multiple copies or replicas of that data on physically isolated servers. That's replication: the process of creating redundant data to streamline and safeguard data availability and durability.

Replication allows you to increase data availability by creating multiple copies of your data across servers. This is especially useful if a server crashes or if you experience service interruptions or hardware failure.

If your data only resides in a single database, any of these events would make accessing the data impossible. But thanks to replication, your applications can stay online in case of database server failure, while also providing disaster recovery and backup options.

With MongoDB, replication is achieved through a replica set. Writer operations are sent to the primary server (node), which applies the operations across secondary servers, replicating the data.

If the primary server fails (through a crash or system failure), one of the secondary servers takes over and becomes the new primary node via election. If that server comes back online, it becomes a secondary once it fully recovers, aiding the new primary node.



How do I enable replication in MongoDB?

To start, you'll need MongoDB installed on three or more nodes. Each of the nodes in the cluster will need to be able to communicate with the others over a standard port (27017 by default). Additionally, each replica set member needs to have a hostname that is resolvable from the others.

Overview: Network Connectivity

- Establish a virtual private network. Ensure that your network topology routes all traffic between members within a single site over the local area network.
- Configure access control to prevent connections from unknown clients to the replica set.

- Configure networking and firewall rules so that incoming and outgoing packets are permitted only on the default MongoDB port and only from within your deployment. See the IP Binding considerations.
- Ensure that each member of a replica set is accessible by way of resolvable DNS or hostnames.

For more detail, check out: production notes in our documentation and security hardening.

1: Start each member of the replica set with the appropriate options.

The following example specifies the replica set name and the ip binding through the `--replSet` and `--bind_ip` command-line options:

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

For `<hostname(s)|ip address(es)>`, specify the hostname(s) and/or ip address(es) for your mongod instance that remote clients (including the other members of the replica set) can use to connect to the instance.

2: Connect a mongo shell to one of the mongod instances.

From the same machine where one of the mongod is running, start the mongo shell. To connect to the mongod listening to localhost on the default port of 27017, simply issue:

```
mongo -u $USERNAME -p $PASSWORD
```

Depending on how you installed MongoDB and set up your environment, you may need to specify the path to the mongo binary.

3: Initiate the replica set.

From the mongo shell, run the `full rs.initiate({...})` on replica set member 0. This command initializes the replica set, and should only be run on the first replica set member. On subsequent nodes, you can run the command without parameters - just `rs.initiate()`.

```
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" }
  ]
})
```

MongoDB initiates a replica set, using the default replica set configuration.

4: View the replica set configuration.

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

```
rs.conf()
```

The replica set configuration object resembles the following:

```
{
   "_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
      },
      {
         "_id" : 1,
         "host" : "mongodb1.example.net:27017",
         "arbiterOnly" : false,
         "buildIndexes" : true,
         "hidden" : false,
         "priority" : 1,
```

```
            "tags" : {

            },
            "slaveDelay" : NumberLong(0),
            "votes" : 1
        },
        {
            "_id" : 2,
            "host" : "mongodb2.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")
    }
}
```

5: Ensure that the replica set has a primary.

Use `rs.status()` to identify the primary in the replica set.

How does MongoDB detect replication lag?

Replication lag is a delay in data being copied to a secondary server after an update on the primary server. Short windows of replication lag are normal, and should be considered in systems that choose to read the eventually-consistent secondary data. Replication lag can also prevent secondary servers from assuming the primary role if the primary goes down.

If you want to check your current replication lag:

- In a Mongo shell connected to the primary, call the rs.printSlaveReplicationInfo() method. This returns the `syncedTo` value for each member, which shows the time when the last oplog entry was written to the secondary server.

  Replication lag can be due to several things, including:

- **Network Latency:** Check your ping and traceroute to see if there is a network routing issue or packet loss. *See: ping diagonistic documentation, troubleshooting replica sets.*

- **Disk Throughput:** Sometimes the secondary server is unable to flush data to disk as rapidly as the primary. This is common on multi-tenant systems, especially if the system accesses disk devices over an IP network. System-level tools, like vmstat or iostat can help you find out more. *See: production notes, mongostat.*

- **Concurrency:** Long-running operations on the primary can block replications. Set up your write concern so that write operations don't return if replication can't keep up with the load. Alternatively, check slow queries and long-running operations via the database profiler. *See: Write Concern.*

- **Appropriate Write Concern:** If the primary requires a large amount of writes (due to a bulk load operation or a sizable data ingestion), the secondaries may not be able to keep up with the changes on the oplog. Consider setting your write concern to "majority" in order to ensure that large operations are properly replicated.

What is the difference between replication and sharding?

- **Replication:** The primary server node copies data onto secondary server nodes. This can help increase data availability and act as a backup, in case if the primary server fails.

- **Sharding:** Handles horizontal scaling across servers using a shard key. This means that rather than copying data holistically, sharding copies pieces of the data (or "shards") across multiple replica sets. These replica sets work together to utilize all of the data.

Think of it like a pizza. With replication, you are making a copy of a complete pizza pie on every server. With sharding, you're sending pizza slices to several different replica sets. Combined together, you have access to the entire pizza pie.

Replication and sharding can work together to form something called a sharded cluster, where each shard is replicated in turn to preserve the same high availability.

What is the benefit of replication?

Replication has several benefits. It increases data availability and reliability thanks to there being multiple live copies of your data.

Replication is also helpful in case of an event like hardware failure or a server crash. Rather than suffer downtime (or, even worse, losing your data entirely), replication ensures your data is safely protected across multiple servers. If you have distributed analytics teams, you can effectively collaborate on business intelligence projects.

Does replication affect latency?

Replication does not meaningfully affect read or write latency to primary servers. Application latency can be improved in cases where it makes sense to read data from replica set secondary nodes, provided you're okay showing customers eventually-consistent data.

Conclusion

Rather than having to configure and manage everything yourself, you can always use MongoDB Atlas. It streamlines and automates your replica sets, making the process

effortless for you. MongoDB Atlas can also deploy globally sharded replica sets with few clicks, enabling data locality, disaster recovery, and multi-region deployments.

Related Content:

- How to Create a Database in MongoDB
- Beginners Guide: MongoDB Basics
- Embedded Documents in MongoDB
- Examples of Using MongoDB

Contact Us

Atlas Status

Manage Cookies

Customer Portal

Customer Support

Social

GitHub

LinkedIn

X

Facebook

Stack Overflow

YouTube

Twitch