MongoDB Architecture

Google slide deck available here

Architecture

Overview

How a query request from an application is processed

Database server

Aspects to consider for a high performance database

Replication

What is it for and what it isn't for

Different roles in a replica set

Oplog and how replication works in MongoDB

Sharding

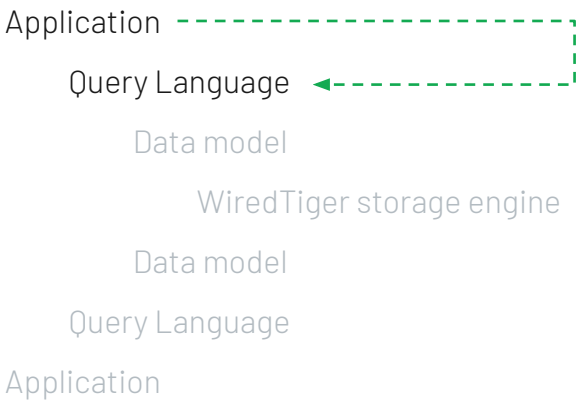
An introduction to the components of a sharded cluster

We'll cover MongoDB's architecture by firstly giving a broad overview of the various components used in processing a typical query, secondly we'll focus a little more into the database server and consider aspects which are important for a high performance database. In terms of the MongoDB database server, we'll discover how many lines of code it takes to create and test it.

Thirdly, we'll move to the area of Replication in MongoDB. We'll answer what it is for and what it is not designed for. We'll explore the various roles in a replica set before moving to the oplog which is the mechanism that powers replication in MongoDB.

Fourthly and finally, we'll introduce sharding in MongoDB and provide an overview of the various components in a sharded cluster.

Overview of how a query is processed



For the next few slides, we'll examine how a query is processed from an application by MongoDB to help illustrate a number of the major components in the database.

Firstly, the application makes the request to MongoDB, it parses this (which may involve some authentication and authorization checking but we'll leave those out for simplicity in this walkthrough) to the query layer.

Query layer

The query is passed to the query analyser, it analyses the shape of the query, selects the best index (if any) to fit the query, and then caches the query shape and which index best fits the query

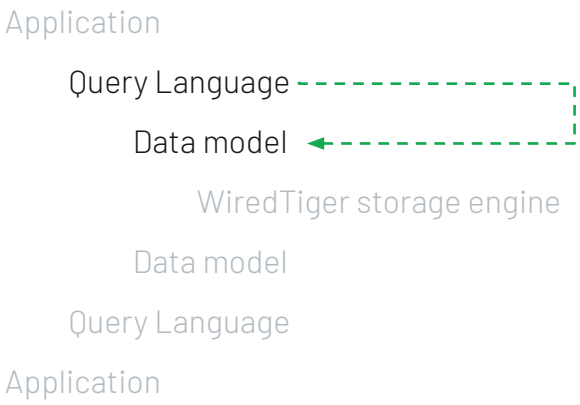
This information is added to the request and sent to the next layer for further processing

The request is passed to the query layer, the first processing occurs with the query analyser. It determines the shape of the query and then determines what index, if any, fit the query. If a index does fit, it's cached.

Indexes are designed to track a small portion of a collection's data in a format that is quick and easy to traverse.
Their goal is to speed up queries and updates. We cover indexing and indexes in more depth in a later lesson.

The shape and index information are then added to the request before passing it down to the next layer for processing.

Overview of how a query is processed



The query language request is passed down to the next layer, the data model. At this point, the request consists of the query, the query shape, and any applicable indexes but it does not yet have the details on which documents to retrieve. The data model layer will add these details to the request.

Data model layer

The request from the query layer is translated in this layer into a set of documents that need to be retrieved

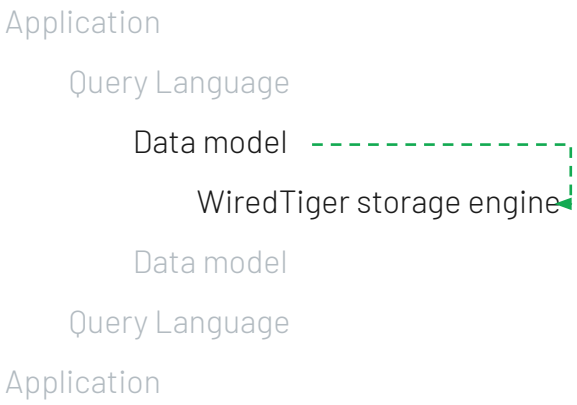
No knowledge in this layer of how the documents are stored but does understand indexes

Combines the results of this layer (the set of documents to be retrieved) and sends these to the storage layer

The request is processed in the data model layer where the set of documents that need to be retrieved are calculated and added to the request.

This layer has no knowledge of how the documents are physically stored but it does understand indexes.
It combined the set of documents to be retrieved with the existing request and then sends it to the next layer, the storage layer for processing.

Overview of how a query is processed



The request now has information from the query layer (index shape, index if any) and the data model layer (the specific documents to be retrieved). It still doesn't know how the documents are stored or how to retrieve them, these tasks are done in the storage layer.

In MongoDB, the storage layer interacts with the WiredTiger storage engine. WiredTiger is responsible for storing the information. It is a modern storage engine designed to use multi-core computer architecture. It provides for document-level concurrency control for write operations, meaning that locking occurs at a document level so it is possible to write to several documents in the same collection at any point. It also provides checkpointing where all the data is written to disk every 60 seconds. It also supports compression for all collections and indexes, which means the database requires less storage but requires more CPU to uncompress the data when it is required.

Storage layer

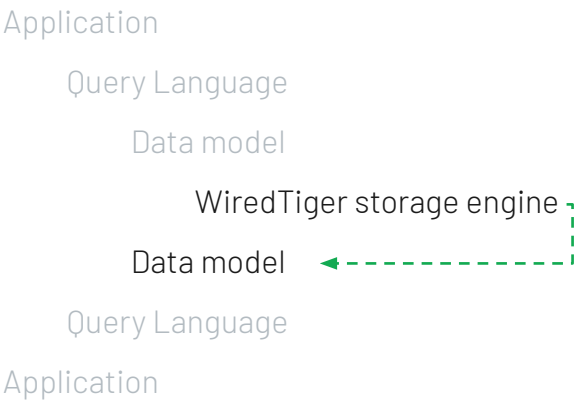
The request from the data model layer translates the request to the underlying format (WiredTiger) with the specific records to load from disk, if they are not already in memory

The results of this are then passed back up through each successive layer until it is returned to the application which made the request

The request from the data layer is translated by WiredTiger to it's underlying format to request the specific records from the disk if they are not already present in memory.

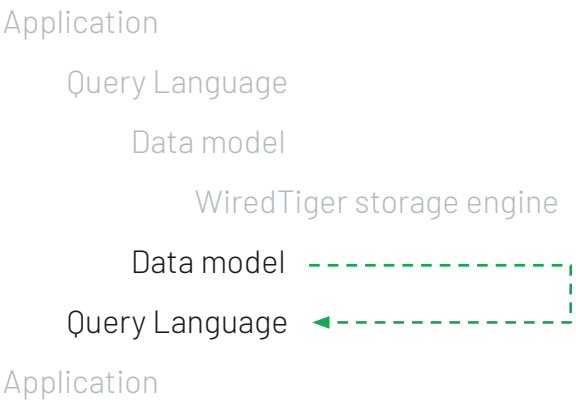
The request is then passed back up through each successive layer until it is return to the application which made the request of the database.

Overview of how a query is processed



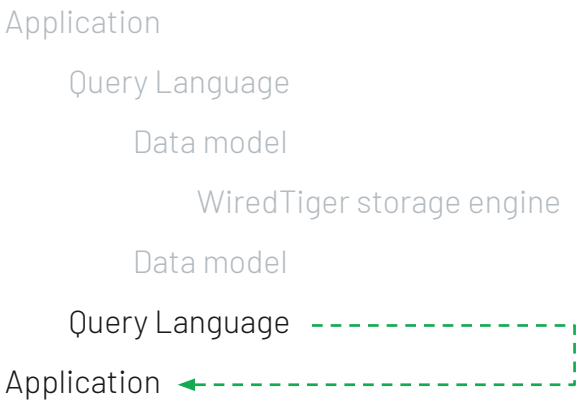
The complete documents are now passed back up from the storage layer to the data model layer.

Overview of how a query is processed



From the data model layer the data is passed back up the query layer.

Overview of how a query is processed



Finally the data is wrapped and passed from the MongoDB database back to the MongoDB Driver and from there to the Application.

Quiz

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- A. Data model
- B. Query
- C. Compression
- D. Storage

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- ✓ A. Data model
- ✓ B. Query
- ✗ C. Compression
- ✓ D. Storage

CORRECT: Data model - This is where the request gets the set of documents added which it will request from the storage layer
CORRECT: Query - This is the first layer a request gets processed by and it adds the query shape and an index if applicable.
INCORRECT: Compression - There is no compression layer, compression is handled in the storage layer or in the communication between the application or between nodes/instances of MongoDB.
CORRECT: Storage - This is the final layer that a request gets processed by and deals with getting the specific documents from memory or from the disk and passing the response back to the layer above it.

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- A. Data model
- B. Query
- C. Compression
- D. Storage

This is correct. This is where the request gets the set of documents added which it will request from the storage layer.

CORRECT: Data model - This is correct. This is where the request gets the set of documents added which it will request from the storage layer

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- A. Data model
- B. Query
- C. Compression
- D. Storage

This is correct. This is the first layer a request gets processed by and it adds the query shape and an index if applicable.

CORRECT: Query - This is correct. This is the first layer a request gets processed by and it adds the query shape and an index if applicable.

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- A. Data model
- B. Query
- C. Compression
- D. Storage

This incorrect. There is no compression layer, compression is handled in the storage layer or in the communication between the application or between nodes/instances of MongoDB.

INCORRECT: Compression - This is incorrect. There is no compression layer, compression is handled in the storage layer or in the communication between the application or between nodes/instances of MongoDB.

Quiz

Which of the following layers does a request from an application get passed through and returned by to the application?

- ✓ A. Data model
- ✓ B. Query
- ✗ C. Compression
- ✓ D. Storage

This is correct. This is the final layer that a request gets processed by and deals with getting the specific documents from memory or from the disk and passing the response back to the layer above it.

CORRECT: Storage - This is correct. This is the final layer that a request gets processed by and deals with getting the specific documents from memory or from the disk and passing the response back to the layer above it.

Implementation

Important aspects for a database

Correctness

Latency

Throughput

Scalability

Usability

In terms of the implementation aspects of the database, we'll cover a number of implementation details and information related to the implementation in terms of platforms the application is built for as well as examples of some external libraries used by MongoDB.

Before moving to the specific implementation aspects, it is worth discussing five aspects that a high performance database implementation should satisfy.

Correctness

Correctness (major)

Latency

Throughput

Scalability

Usability

Correctness means that for a given database operation that the data is changed only in allowable ways.

This means that any programming errors cannot change the database in unforeseeable ways.

Latency

Correctness

Latency (major)

Throughput

Scalability

Usability

Latency is how long it takes to service the request from the application by the database. This is important as many applications and services have indicators and measures that they are designed to meet in terms of how long the query can take at a maximum.

Throughput

Correctness

Latency

Throughput (major)

Scalability

Usability

Throughput is concerned with the number of operations that a database can do at any given moment in time. MongoDB is a high performance database so ensuring that a high number of operations can occur concurrently is a key factor in the performance of the database.

Scalability

Correctness

Latency

Throughput

Scalability (minor)

Usability

Scalability in terms of a database is concerned with how the database manages when more and more data is added to it. At a certain point the limits of hardware come into play and other machines are required. In the case of MongoDB, sharding helps overcome this aspect of scalability.

Usability

Correctness

Latency

Throughput

Scalability

Usability (minor)

Usability in terms of a database is related to the difficulty in interacting and programming your applications. In MongoDB, MQL helps provide a simple syntax for queries and greatly adds in the usability of the database by making it easier to query and retrieve data.

MongoDB database server code base

Open source and hosted on GitHub.

~900,000 lines of C++17 code.

~1.4M lines if all unit-tests are counted.

~400,000 lines of JavaScript code for administrative utilities and tests.

Runs under continuous integration after (almost) every commit.

Predictable latency and throughput with C++.

The MongoDB server code base is open source and hosted on GitHub. It is approximately 900k of actual production C++17 code and with unit test it's in the region of 1.4 million lines of code.

In addition we have a lot of integration tests written in JavaScript, which are about ~400,000 lines of additional code, which gets executed by a continuous integration process after almost every commit.

As a database server, excluding correctness, the most important requirement is that MongoDB has predictable latency and throughput. This is very difficult to achieve with garbage-collected or interpreted languages and C++ gives direct access to memory allocations and deallocations.

MongoDB build, platforms and libraries

Uses SCons as a build system.

Multi-platform compilation for RHEL7.2, Ubuntu, Windows, OSX, ARM, PPC64LE, IBM s360x and others.

Uses external libraries, such as Boost 1.70.0, ASIO, MozJS and Abseil.

The server and its C++ unit-tests are built using SCons, which is a Python-based build system.

The server is built and tested on more than 12 different platforms, which include the most popular such as Linux and Windows, but also some pretty exotic ones, such as ARM, PowerPC and the IBM s360 mainframes.

It can be built both with little-endian or big-endian order in mind and also we take advantage of the different memory consistency models that these platforms offer.

In addition, the server uses some of the most popular C++ open source libraries, such as Boost, ASIO, MozJS (which is the JavaScript engine of Firefox) and the Google Abseil library, which offers a fully standard-compatible hash map, which is more performant than the one which comes with STL.

Quiz

Quiz

Which of the following criteria are required for a database?

- A. Correctness
- B. Latency
- C. Throughput
- D. Scalability
- E. Usability

Quiz

Which of the following criteria are required for a database?

- ✓ A. Correctness
- ✓ B. Latency
- ✓ C. Throughput
- ✗ D. Scalability
- ✗ E. Usability

CORRECT: Correctness - The correctness of data is a key criteria of any data in a database
CORRECT: Latency - Latency and specifically how long between the request for the data and when it is returned is another key criteria for any database.
CORRECT: Throughput - Throughput and specifically how many operations are processed by the database at any moment in time are another critical criteria for a database
INCORRECT: Scalability - The scalability of a database whilst an important criteria, it is not critical.
INCORRECT: Usability - The ease of use of the database is important but again not a critical criteria.

Quiz

Which of the following criteria are required for a database?

- ✓ A. Correctness
- ✓ B. Latency
- ✓ C. Throughput
- ✗ D. Scalability
- ✗ E. Usability

This is correct. The correctness of data is a key criteria of any data in a database.

CORRECT: Correctness - This is correct. The correctness of data is a key criteria of any data in a database

Quiz

Which of the following criteria are required for a database?

- ✓ A. Correctness
- ✓ B. Latency
- ✓ C. Throughput
- ✗ D. Scalability
- ✗ E. Usability

This is correct. Latency and specifically how long between the request for the data and when it is returned is another key criteria for any database.

CORRECT: Latency - This is correct. Latency and specifically how long between the request for the data and when it is returned is another key criteria for any database.

Quiz

Which of the following criteria are required for a database?

- ✓ A. Correctness
- ✓ B. Latency
- ✓ C. Throughput
- ✗ D. Scalability
- ✗ E. Usability

This is correct. Throughput and specifically how many operations are processed by the database at any moment in time are another critical criteria for a database.

CORRECT: Throughput - This is correct. Throughput and specifically how many operations are processed by the database at any moment in time are another critical criteria for a database

Quiz

Which of the following criteria are required for a database?

- ✓ A. Correctness
- ✓ B. Latency
- ✓ C. Throughput
- ✗ **D. Scalability**
- ✗ E. Usability

This incorrect. The scalability of a database whilst an important criteria, it is not critical.

INCORRECT: Scalability - This is incorrect. The scalability of a database whilst an important criteria, it is not critical.

Quiz

Which of the following criteria are required for a database?

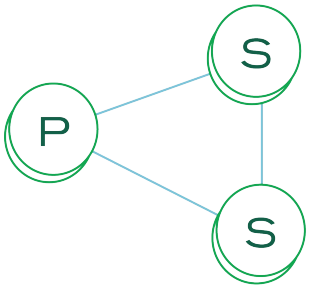
- A. Correctness
- B. Latency
- C. Throughput
- D. Scalability
- E. Usability

This incorrect. The ease of use of the database is important but again not a critical criteria.

INCORRECT: Usability - This is incorrect. The ease of use of the database is important but again not a critical criteria.

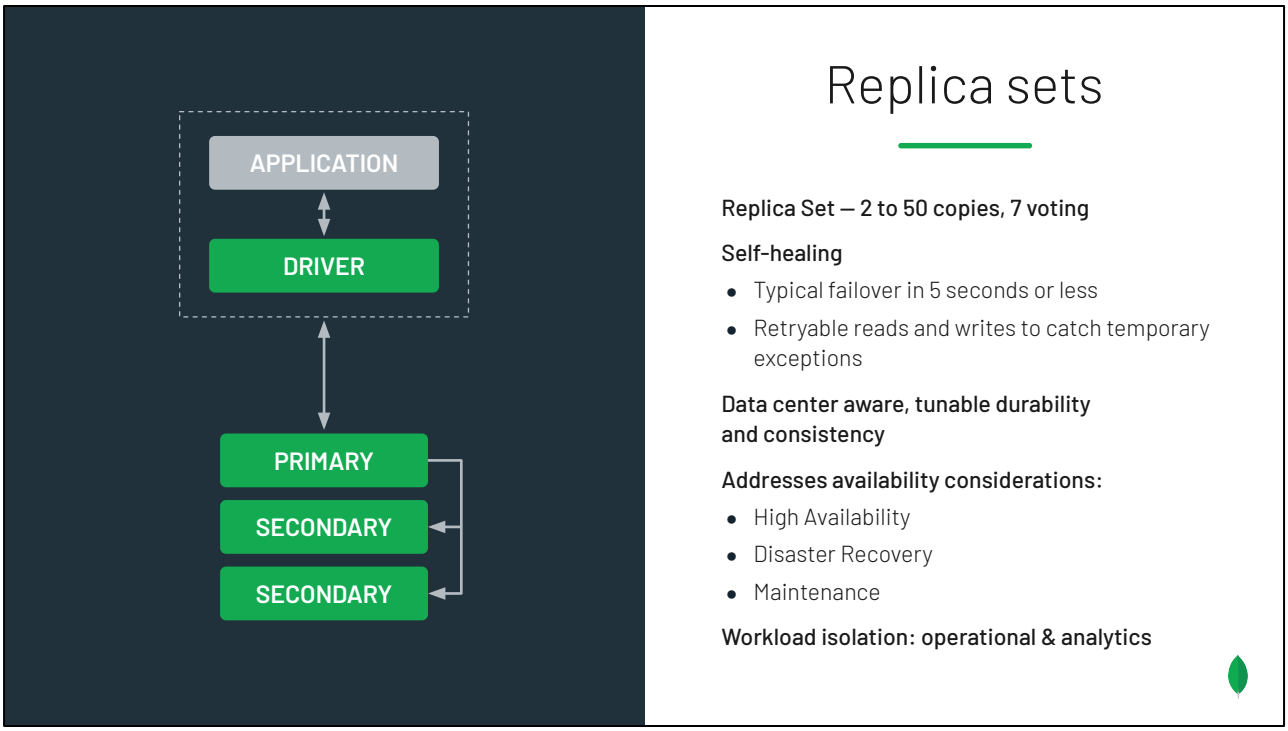
Replication

MongoDB uses replica sets for redundancy and high availability.

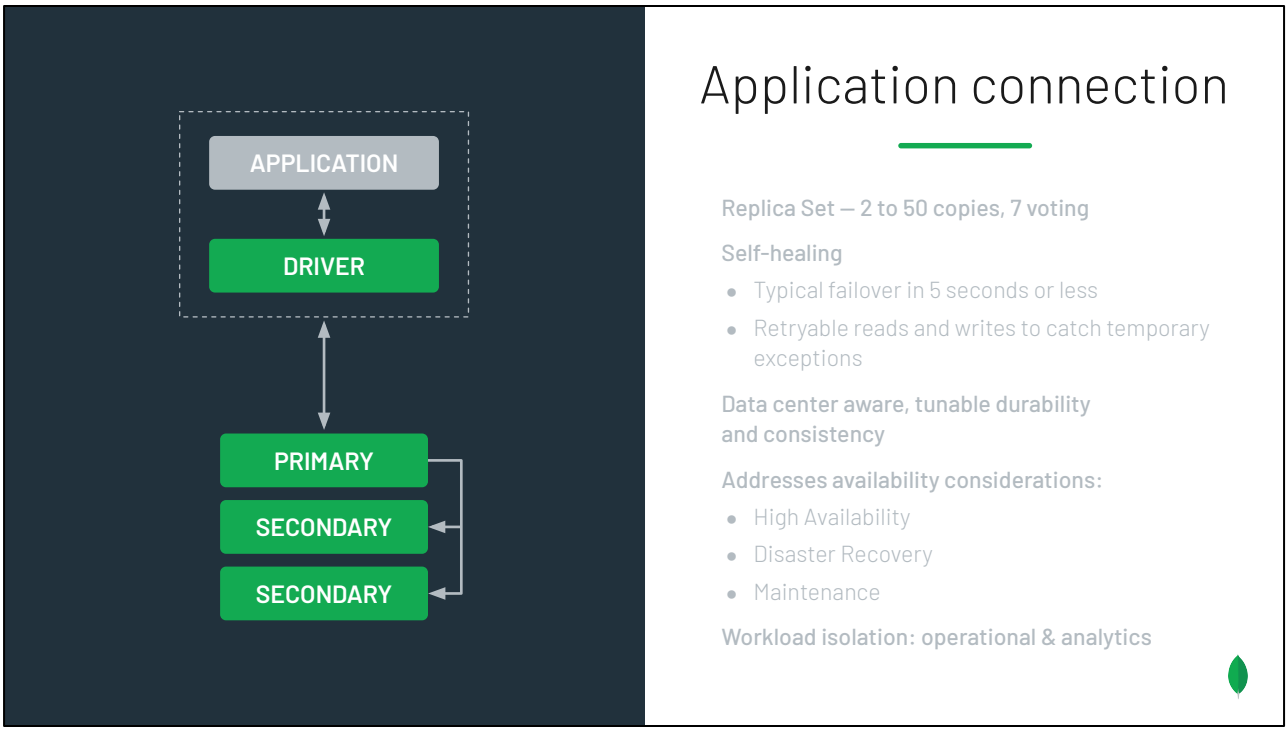


Replication is the mechanism of how data is distributed in MongoDB, in the next slides we'll dive deeper into this topic. MongoDB uses the concept of replicas and replica sets. These are used for redundancy and for high availability. They are not designed to help scale your data, this is a common misconception. Sharding is the mechanism for scaling data in MongoDB and we'll look at it shortly.

We'll look next at the technical aspects of a replica set and it's features to get a better understand of what it provides in terms of functionality.

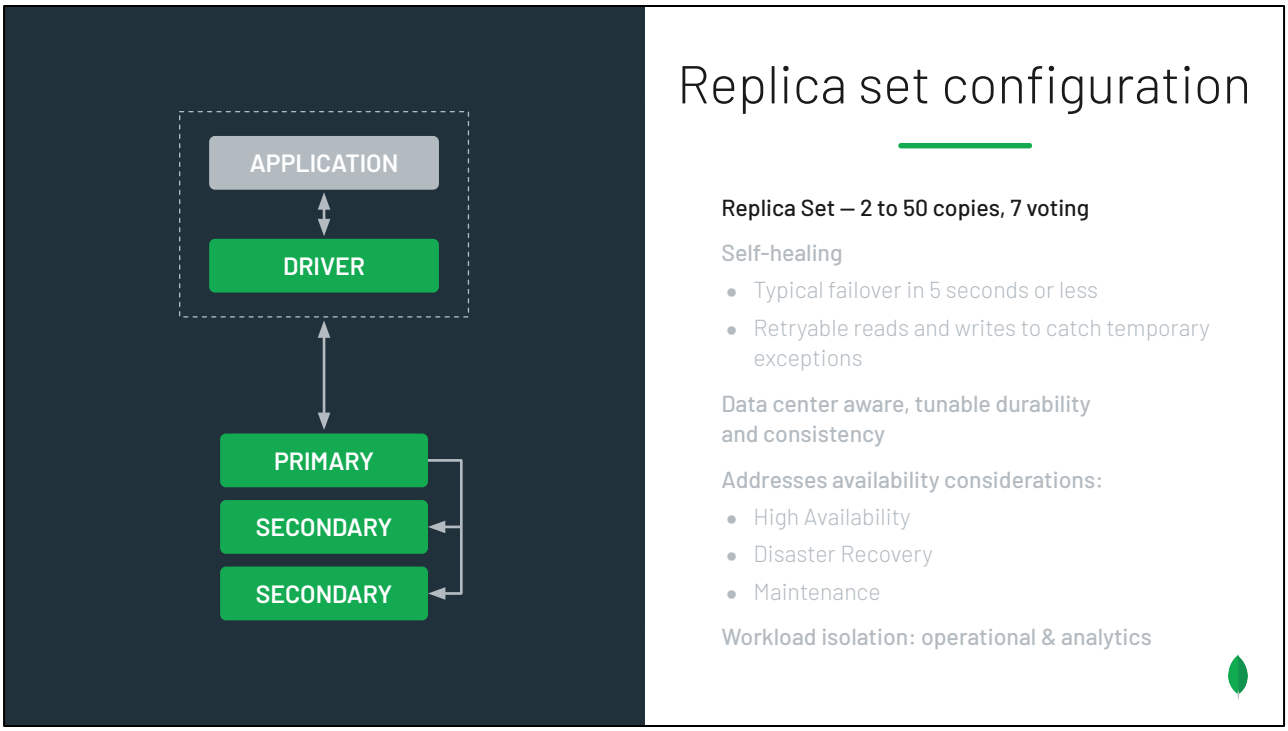


In this lesson, we will start by looking at the main features and configuration aspects of MongoDB's replica sets.

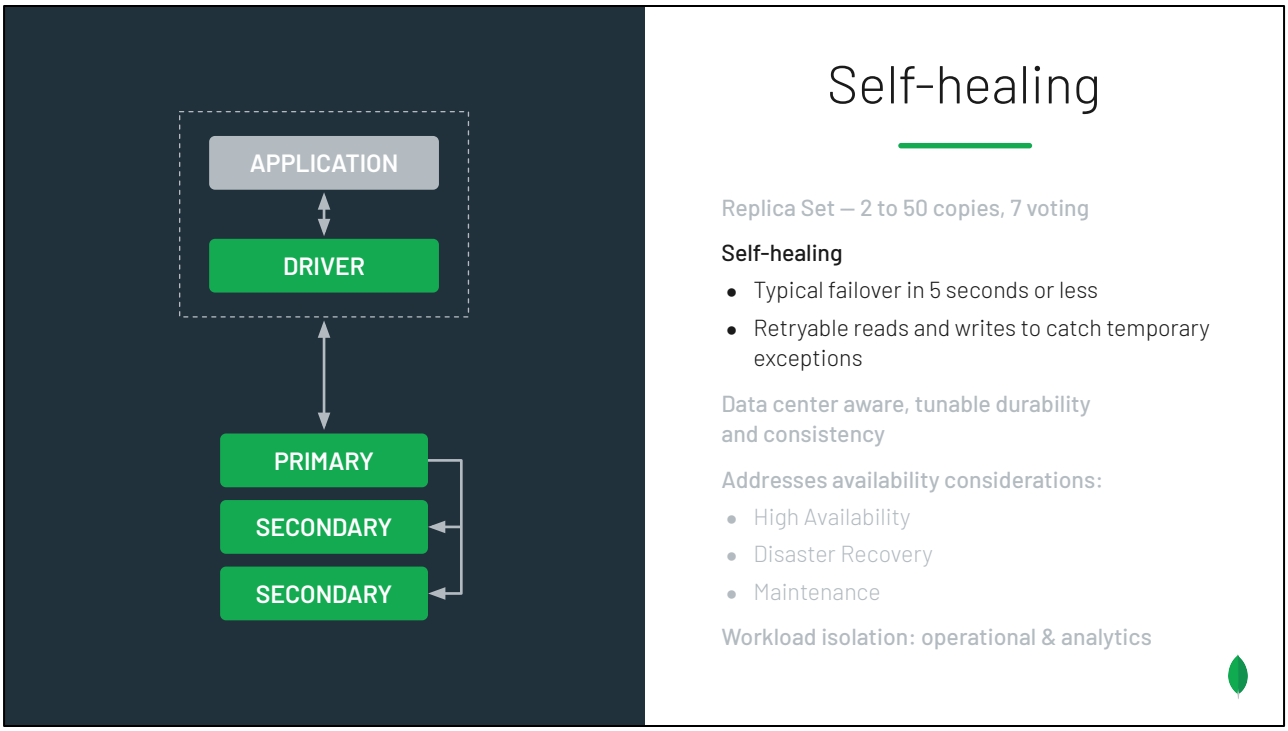


The highlighted diagram shows the typical configuration of how an application uses a driver to connect to a MongoDB replica set. The replication occurs in a replica set with the primary replicating the operations to the secondary members.

We'll look at members and roles, a little later in this lesson.

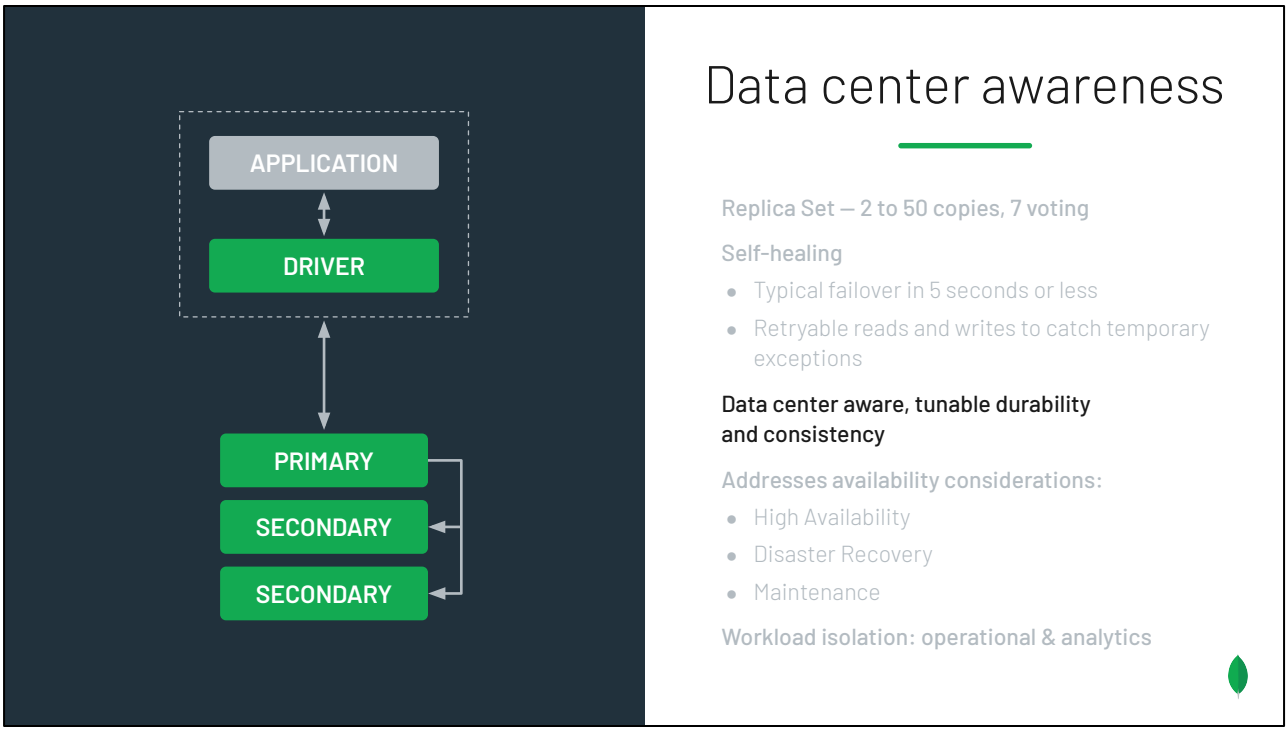


MongoDB replica sets can have between 2 and 50 instances or copies of the data, however only a maximum of 7 of these can participate in election to determine which is the primary node (there is only one at any given point in time).

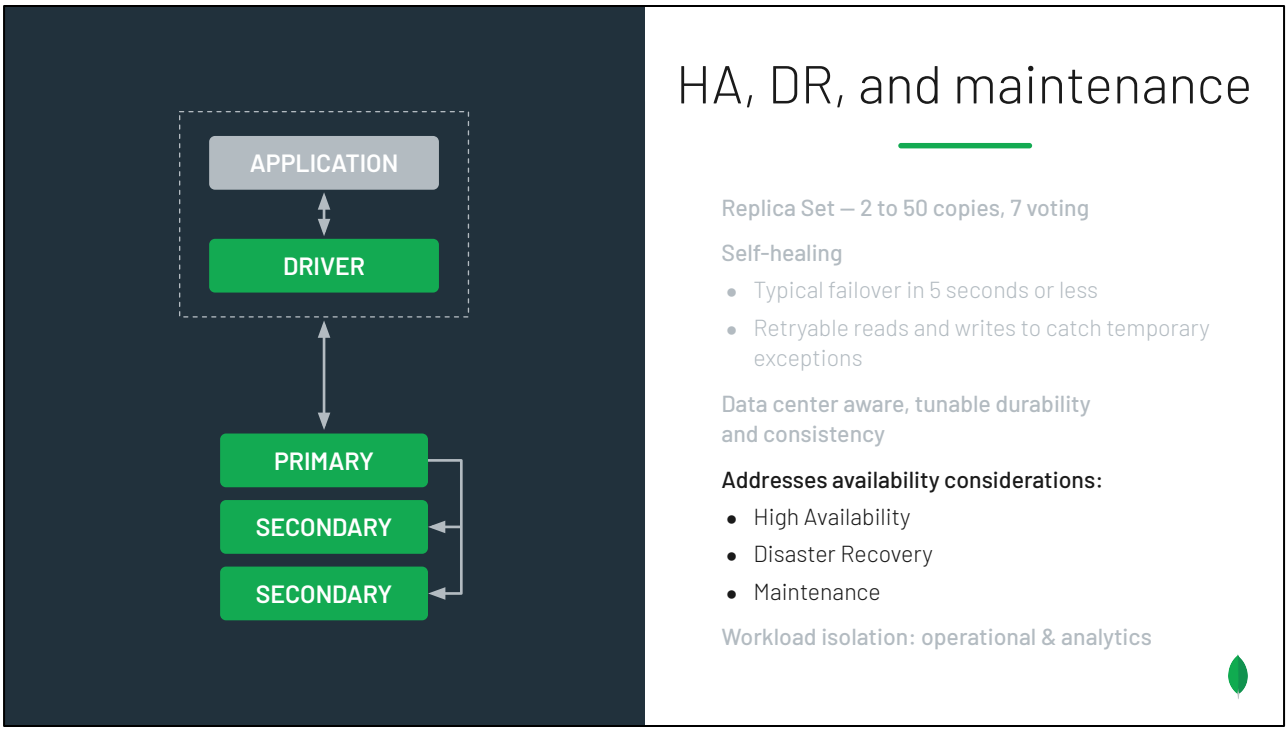


The design of replica sets facilitates rapid failure and recovery, they can be 'tagged' to indicate which data center or geographical location they are situated in. These can be used to read or write to only a specific set / sub set of these members.

Failover normally occurs in 5 seconds or less, writes and reads are automatically retried once to catch this kind of temporary exception. This means for the vast majority of scenarios when a failover occurs that they aren't noticed and don't impact the application.

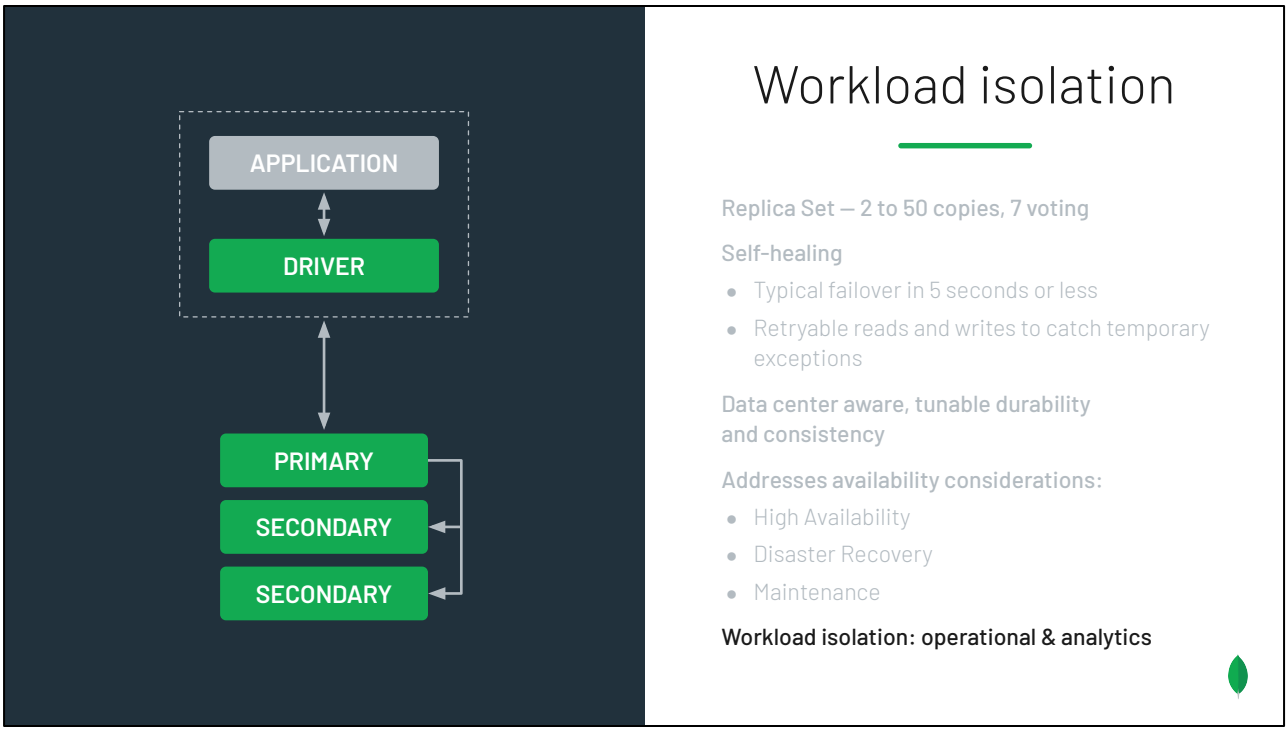


Replica sets can be ‘tagged’ to allow the individual members to become data center aware. They can be used to support tunable durability and consistency of read or of writes by using these tags along with read concerns and/or write concerns within an application.



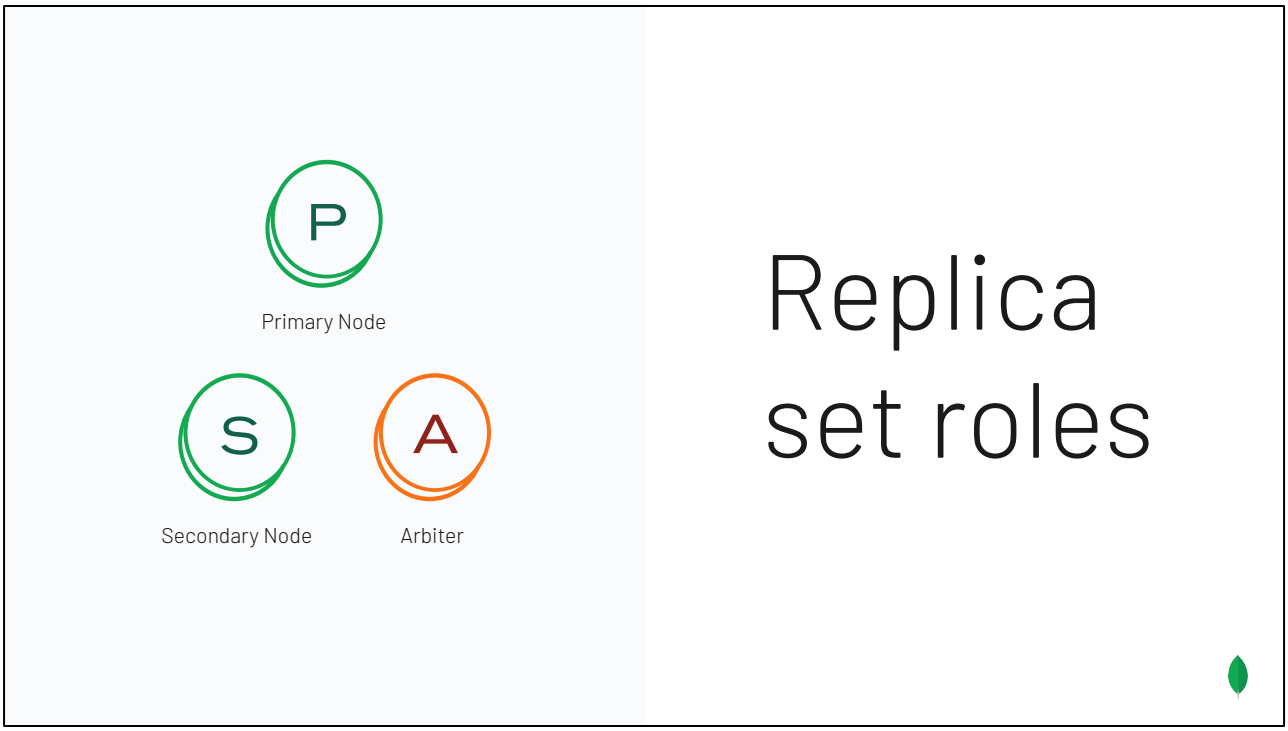
A replica set's purpose is to address three main factors:

- High Availability – Ensure application availability during many types of failures
- Disaster Recovery – Address the Recovery Time Objective (RTO) and Recovery Point Objective (RPO) goals for business continuity
- Maintenance – Perform upgrades and other maintenance operations with no application downtime



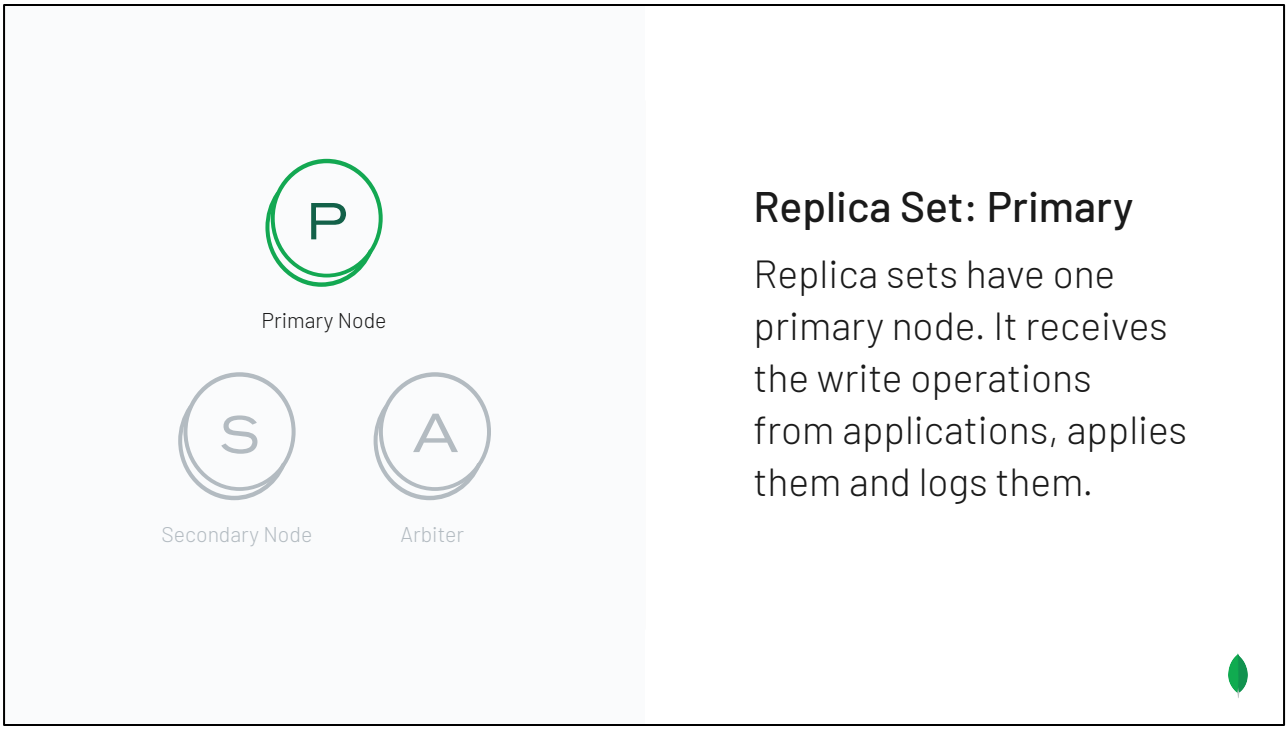
Secondaries can be used for a variety of applications – failover, hot backup, rolling upgrades, data locality and privacy and workload isolation

Tags can be used to facilitate the isolation of workloads such as analytics.



Let’s look at the various roles in a replica set to help understand the architecture of replication a little better.

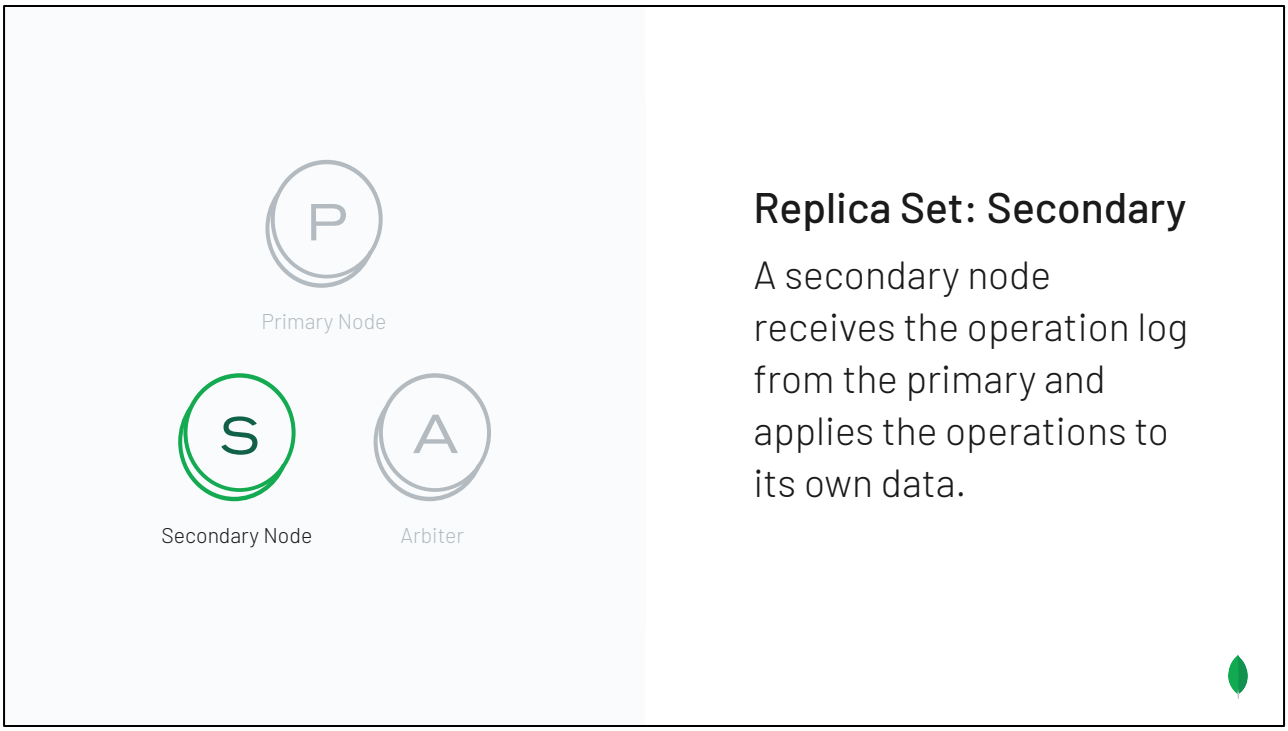
There are three roles or membership types in a replica set, Primary, Secondary, and Arbiter. We’ll cover these in following slides.



Let's look at the first role in a replica set, that of the primary member.

The primary node received all the write operations from applications, it applies them to its data and logs them to an operations log.

If something occurs to this node, an election is triggered. MongoDB uses a Raft like election protocol with additional logic to deal with various additional aspects like roles (primary/secondary) that don't exist in the current Raft implementation. Several of these have been committed back to the Raft OSS project.

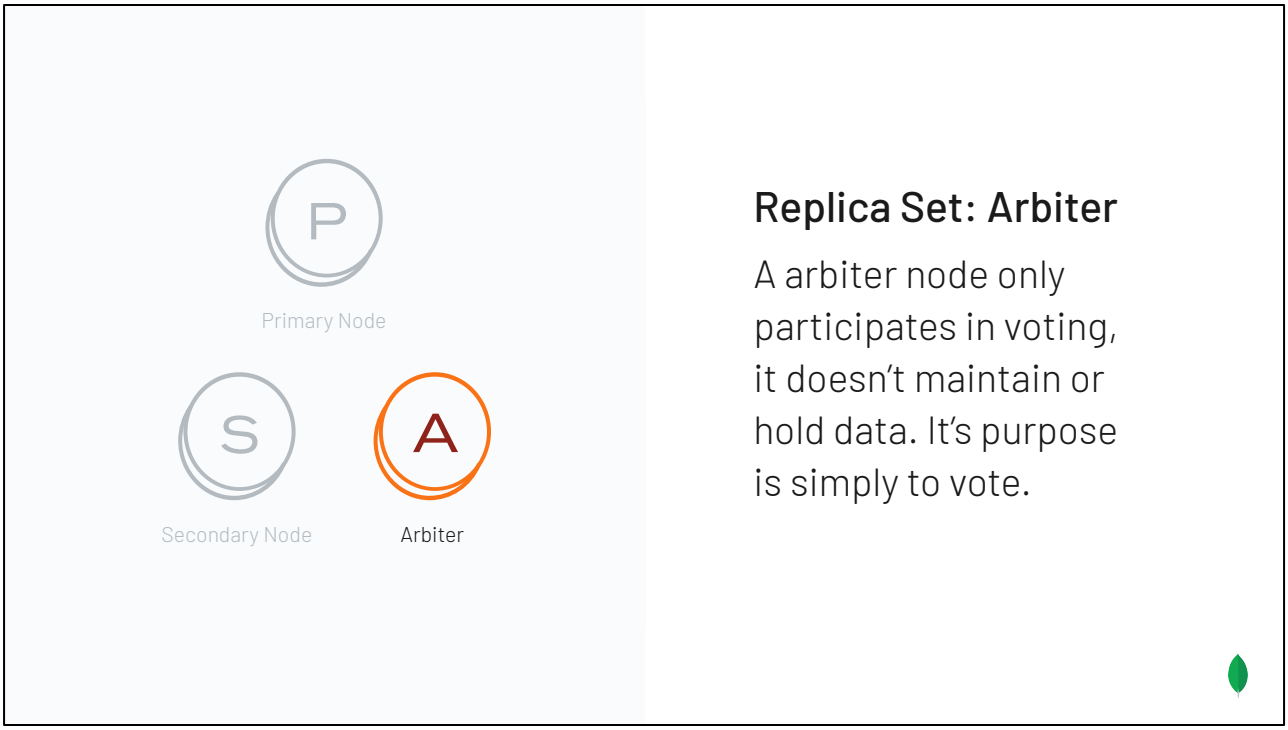


Let's look at the next role in a replica set, that of a secondary member.

All other data bearing nodes in a replica set are secondaries. Each of these receives the operation log from the primary (or sometimes from another secondary in a chained fashion) and it then applies these operations against its own data.

In the case of writing data, you can select various levels of consistency. If you select a write concern of majority, this will mean that your write operation won't be successfully acknowledged back to your driver/application until the write has been confirmed as been written to the majority of the nodes in the replica set.

Read and write concerns as well as read preferences allow you as a developer to tune your application to the data durability you require, you can trade speed off against consistency and vice versa. This is a complex area and if you want to dive deeper you can checkout the deeper material available on the MongoDB University or the official documentation.



Let's look at the last role in a replica set, that of an arbiter member.

Arbiters are voting only members, they're a historically artefact from MongoDB's earlier replication protocol. Ideally, they should not be used in your replica set.

Quiz

Quiz

Which of the following node types are data-bearing in MongoDB replication?

- A. Arbiter
- B. Secondary
- C. Primary

Quiz

Which of the following node types are data-bearing in MongoDB replication?

- ✗ A. Arbiter
- ✓ B. Secondary
- ✓ C. Primary

INCORRECT: Arbiter - An arbiter is a voting only member of a replica set, it does not store any data.
CORRECT: Secondary - A secondary hold data and can vote
CORRECT: Primary - A primary holds data and takes the write operations from an application, it then replicates these operations to the other secondary nodes who themselves apply the operations to their data.

Quiz

Which of the following node types are data-bearing in MongoDB replication?

- A. Arbiter
- B. Secondary
- C. Primary

This incorrect. An arbiter is a voting only member of a replica set, it does not store any data.

INCORRECT: Arbiter - This is incorrect. An arbiter is a voting only member of a replica set, it does not store any data.

Quiz

Which of the following node types are data-bearing in MongoDB replication?

A. Arbiter

B. Secondary

C. Primary

This is correct. A secondary hold data and can vote.

CORRECT: Secondary - This is correct. A secondary hold data and can vote.

Quiz

Which of the following node types are data-bearing in MongoDB replication?

- A. Arbiter
- B. Secondary
- C. Primary

This is correct. A primary holds data and takes the write operations from an application, it then replicates these operations to the other secondary nodes who themselves apply the operations to their data.

CORRECT: Primary - This is correct. A primary holds data and takes the write operations from an application, it then replicates these operations to the other secondary nodes who themselves apply the operations to their data.

Replication: use cases

High Availability

Reduce Read Latency

Replication is not for:

Scaling (use Sharding)

Disaster Recovery (use Backups and multiple geographically separate DCs)

There are two use cases for replication, high availability and to reduce read latency.

High availability is focused on ensuring that despite individual node failure that the replica set itself is available to service any client requests made to the database.

Reducing read latency applies to situations where members within the replica set could be across data centers and potentially also geographically separate. If a member is closer in terms of distance on the network and more specific in terms of the latency from it to the client, it could be used to service the request faster at the cost of potentially being slightly less fresh data.

Replication is not for scaling rather sharding should be used for scaling.
It is also not for disaster recovery, you should use backups and spread your deployment across multiple geographically separate data centers.

High Availability (HA)

Data still available following:

Equipment failure (e.g. server, network switch)

Datacenter failure

Replication in MongoDB uses automatic failover to provide HA

Remaining servers have an election

HA is not 'Active Active' but rather fast failover

High availability ensures that your client can maintain access to the data despite various types of failure from equipment to hosting such as a failure at the datacenter.

Replication is the mechanism used by MongoDB to provide automatic failover. If a failure occurs where the Primary node is lost then the other remaining members have an election to determine who will be the next Primary node.

Failover in MongoDB does not involve hot swapping or changing to another standby Primary rather one of the secondary members is quickly elected, hence it is a fast failover rather than active active type of high availability.

In terms of MongoDB election specifics, there are multiple data bearing members in a replica set and if the primary fails, then the remaining servers hold an election to elect a new primary. It isn't immediate and there is a short period in terms of seconds which this is occurring where the replica set isn't available for writing, it may be available for reading depending on the read concerns (we'll cover these later in the lesson).

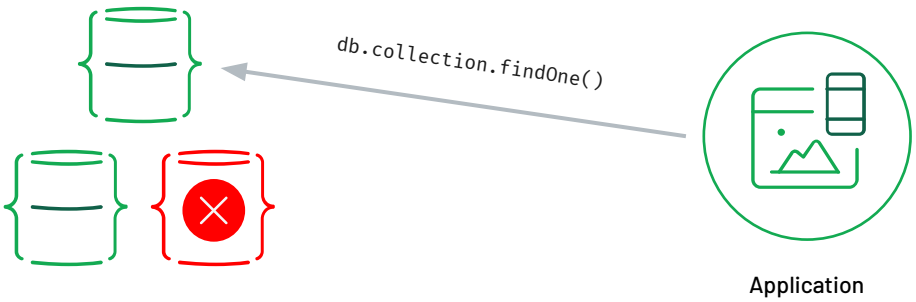
High Availability (HA)

Here's an example of how HA works with a typical 3 member replica set. The client makes a findOne() request.

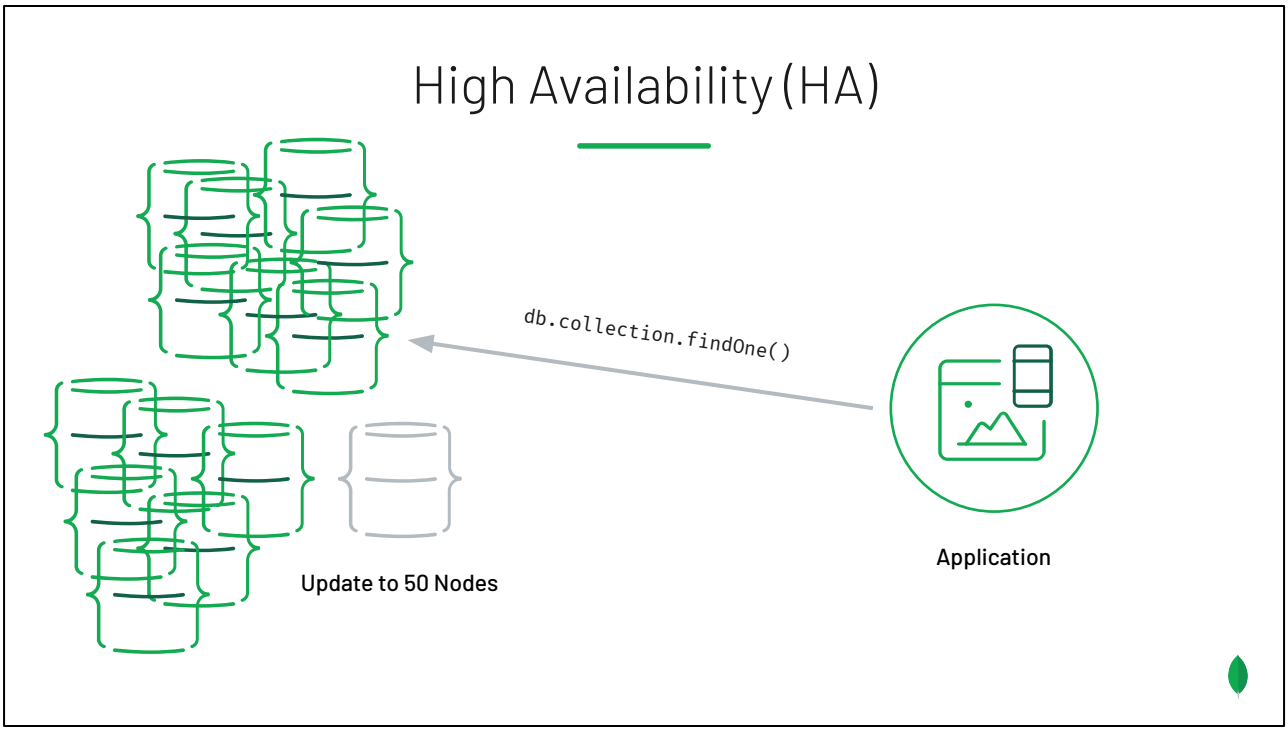
High Availability (HA)

However, the Primary it was attempting to read from had a critical failure and can no longer service this operation.

High Availability (HA)



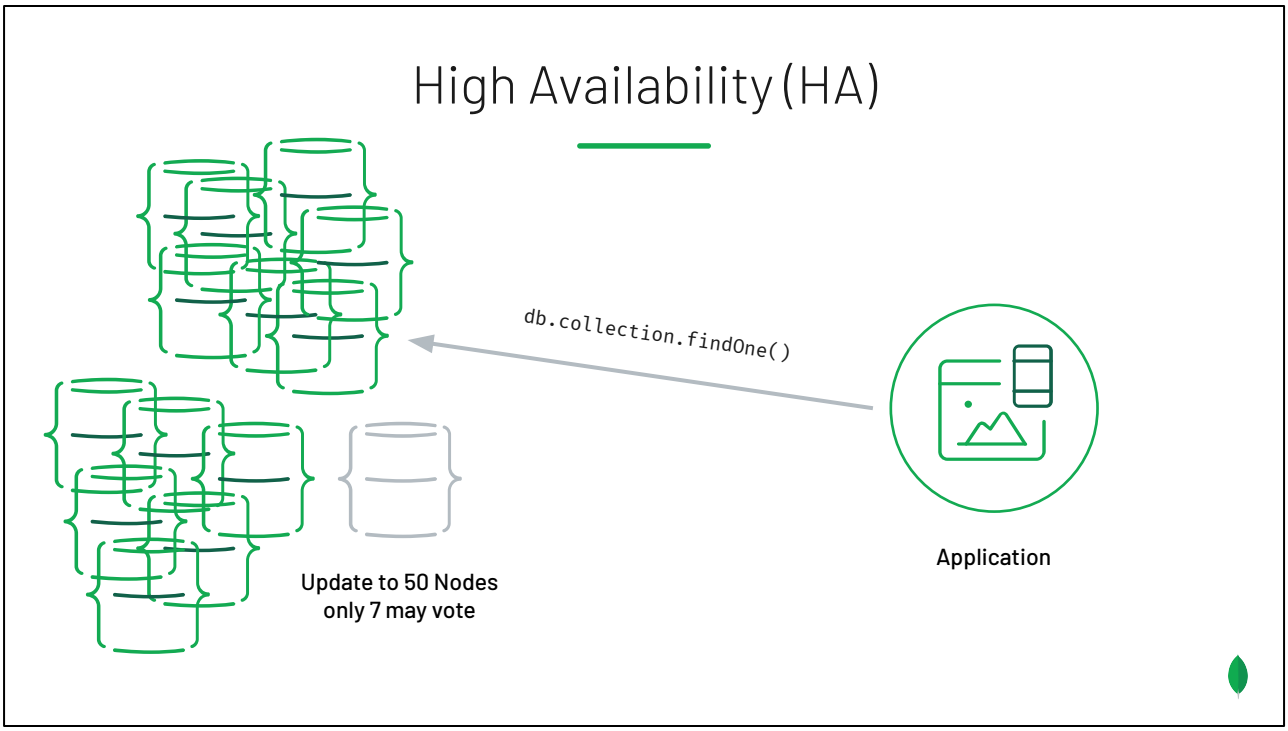
MongoDB automatically makes one retry attempt by default for failed read or write operations.
In the second attempt here, it tries to read again but from the new Primary which will be able to service the read and the document will be returned.



In order to support the demands of global applications, replica sets can have up to 50 nodes.

These can be located in many data centers to add additional resilience through their geographical disbursement.

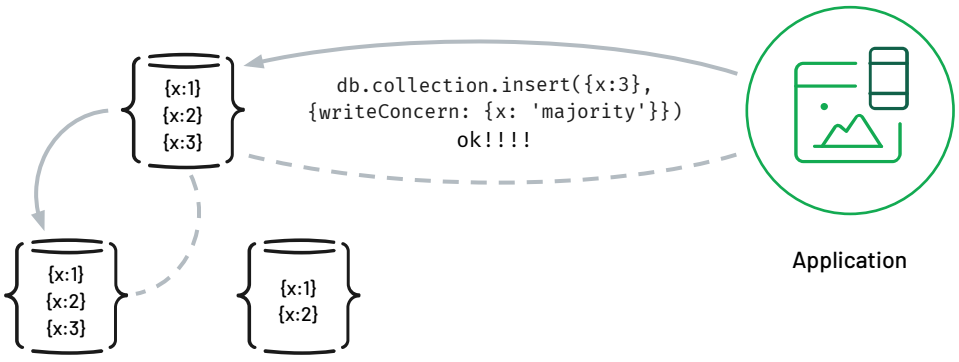
Using several different geographical locations allows for nodes to be placed closer to their users and facilitate additional features such as read preferences to ensure those clients target the closest node to their location to provide faster responses in terms of reduced network latency.



An important note is that the maximum number of voting members in a replica set is 7.

This has implicits to where these voting members should be located. For instance, these should be spread in such a fashion that a failure of a single data center will still allow for sufficient members (a majority of 4 if 7 voting members) are present in other locations to allow for elections to proceed without requiring manual intervention.

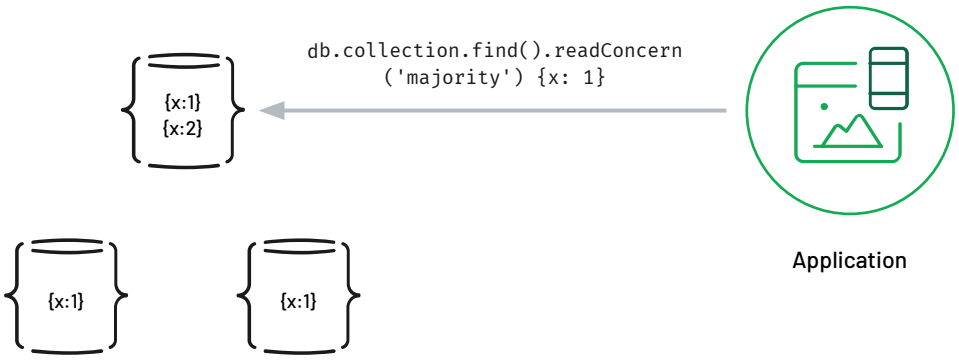
Write concern



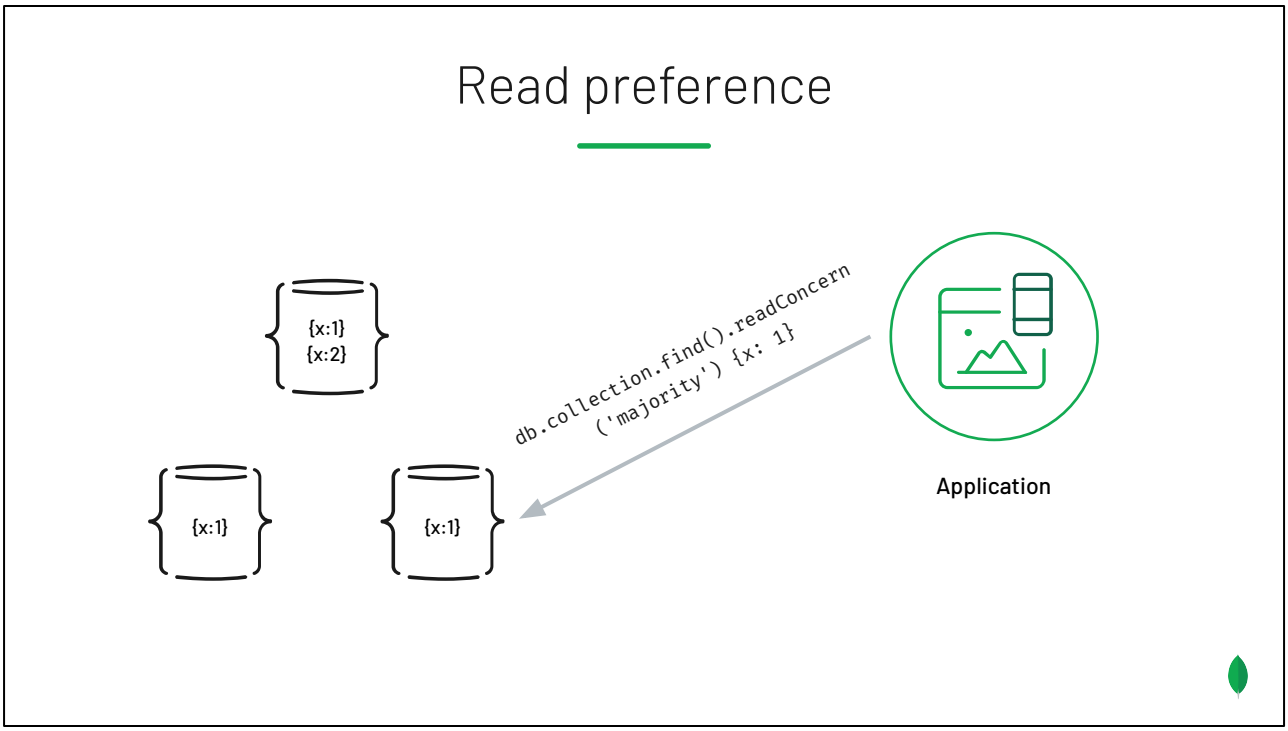
This example highlights how the write concern ‘majority’ will ensure that the document will be written to two of the three data bearing nodes in the replica set before the insert function returns successfully.

It’s important to note the acknowledgement of the write doesn’t mean that the data or document won’t be then replicated to the other member in this example. It means that in the context of only this write concern and insert that the document must only get acknowledgement of having successfully being written to two of the three data bearing nodes before returning. The third node will be updated from the primary as part of the normal replication of the oplog to it as a secondary.

Read concern



A read concern is a similar concept to a write concern, however it is focused on read and ensuring that the data returned is indeed what is present on the nodes, in this example on the majority of the nodes.



Read preference differs from Read concerns, it is focused on the routing of read request to members in a replica set. Read preferences allow for nodes other than the primary to be selected when a client makes a read request. The example here selects the member (whether secondary or primary) that is closest to the client in terms of network latency.

The main reasons to use read preferences is to support one of the following:

- Running systems operations that do not affect the front-end application.
- Providing local reads for geographically distributed applications.
- Maintaining availability during a failover.

Quiz

Quiz

Which of the true for replication in MongoDB?

- A. Allows for 50 voting members
- B. Allows for reads to select the nearest nodes to the client application
- C. Does not automatically retry a read or a write in the event of a failure
- D. Has a short window during a failover where a new primary is elected

Quiz

Which of the true for replication in MongoDB?

A. Allows for 50 voting members

B. Allows for reads to select the nearest nodes to the client application

C. Does not automatically retry a read or a write in the event of a failure

D. Has a short window during a failover where a new primary is elected

INCORRECT: Allows for 50 voting members - MongoDB allows for replica sets with 50 members, however only 7 of these may vote.
CORRECT: Allows for reads to select the nearest nodes to the client application - Drivers can use the read preference of nearest to ensure they read their data from the nearest nodes to them.
INCORRECT: Does not automatically retry a read or a write in the event of a failure - MongoDB retries reads and writes once in the event of a failure.
CORRECT: Has a short window during a failover where a new primary is elected - There is a short time, typically less than 5 seconds during this window the new primary is elected.

Quiz

Which of the true for replication in MongoDB?

- A. Allows for 50 voting members
- B. Allows for reads to select the nearest nodes to the client application
- C. Does not automatically retry a read or a write in the event of a failure
- D. Has a short window during a failover where a new primary is elected

This incorrect. MongoDB allows for replica sets with 50 members, however only 7 of these may vote.

INCORRECT: Allows for 50 voting members - This is incorrect. MongoDB allows for replica sets with 50 members, however only 7 of these may vote.

Quiz

Which of the true for replication in MongoDB?

A. Allows for 50 voting members

B. Allows for reads to select the nearest nodes to the client application

C. Does not automatically retry a read or a write in the event of a failure

D. Has a short window during a failover where a new primary is elected

This is correct. Drivers can use the read preference of nearest to ensure they read their data from the nearest nodes to them.

CORRECT: Allows for reads to select the nearest nodes to the client application - This is correct. Drivers can use the read preference of nearest to ensure they read their data from the nearest nodes to them.

Quiz

Which of the true for replication in MongoDB?

- ✗ A. Allows for 50 voting members
- ✓ B. Allows for reads to select the nearest nodes to the client application
- ✗ C. Does not automatically retry a read or a write in the event of a failure
- ✓ D. Has a short window during a failover where a new primary is elected

This incorrect. MongoDB retries reads and writes once in the event of a failure.

INCORRECT: Does not automatically retry a read or a write in the event of a failure - This is incorrect. MongoDB retries reads and writes once in the event of a failure.

Quiz

Which of the true for replication in MongoDB?

- A. Allows for 50 voting members
- B. Allows for reads to select the nearest nodes to the client application
- C. Does not automatically retry a read or a write in the event of a failure
- D. Has a short window during a failover where a new primary is elected

This is correct. There is a short time, typically less than 5 seconds during this window the new primary is elected.

CORRECT: Has a short window during a failover where a new primary is elected - This is correct. There is a short time, typically less than 5 seconds during this window the new primary is elected.

What is the Oplog?

Operations log

Capped collection

Stores all operations that modify the data.

The primary records it's operations and the secondaries replicate the oplog in an asynchronous fashion.

Each replica set member holds a copy of the log, in `local.oplog.rs`

MongoDB uses an operations log or oplog for replication. It is a capped collection, a fixed sized collection that rewrites its oldest entries once it is full. It's very similar to the concept of a circular buffer, a circular queue, or a ring buffer which are all essential implementations of the same idea.

It stores only the operations that modify the data. Each operation that is stored is idempotent. That is, oplog operations produce the same results whether applied once or multiple times to the target dataset.

The primary writes it's oplog and the secondaries replicate these oplog entries in an asynchronous fashion by copying and then applying the operations to their own databases and collections.

Every member of the replica set holds a copy of the log in their local database in the oplog.rs collection.

Oplog: type of replication

Statement based replication

Document based replication

Binary replication (file copy)

See: http://smalldatum.blogspot.com/2020/02/describing-replication.html

There are three main types of replication used in databases.
Statement based replication, where the statement that changes the database is recorded into the log and that is shared to other members. A single statement can change multiple documents.
Document based replication, where the changes to each document are stored in the log and shared with other members. Each change to each document is a single operation record. This is how MongoDB stores the changes and it is what is replicated to other members.
Binary replication or file copy, is where the file storing the data is simply copied to the other members or the changed parts of the file.

The main advantage of document based replication over statement based replication is that as the changes are idempotent they can be replayed in parallel and make that replaying a crash safe process (in terms of recovery).

For more details checkout this blog from one of our principal engineers - http://smalldatum.blogspot.com/2020/02/describing-replication.html

```
db.bar.insertMany([{a:1}, {a:2}, {a:3}])

use local
db.oplog.rs.find({"o.msg": {$ne: "periodic
noop"}}, {op:1,
o:1}).sort({$natural:-1}).limit(3)

{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 3 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 2 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 1 } }
```

The Oplog

The operations log (oplog) holds **idempotent** statements that modify the database. MongoDB does this at the document level, each log entry represents one change document.

The operations log holds idempotent statements that modify data in the database. This is done at the document level with each log entry corresponding or representing a change to one document.

```
db.bar.insertMany([{a:1}, {a:2}, {a:3}])
```

```
use local
db.oplog.rs.find({"o.msg": {$ne: "periodic
noop"}}, {op:1,
o:1}).sort({$natural:-1}).limit(3)

{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 3 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 2 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 1 } }
```

Adding data to MongoDB (and to the oplog)

Insert some test data into the collection **bar**.

In order to better understand the operations log or Oplog as it is more commonly known/referred to, let's work through a simple example.

In this case let's insert three documents into the bar collection. This assumes you are connected to a MongoDB replica set primary. Only replica sets provide the operations log, if you run a standalone instance then these command will not work.

The oplog itself special capped collection that keeps a rolling record of all operations that modify the data stored in your databases.

```
db.bar.insertMany([{a:1}, {a:2}, {a:3}])
```

```
use local
db.oplog.rs.find({"o.msg": {$ne: "periodic
noop"}}, {op:1,
o:1}).sort({$natural:-1}).limit(3)
```

```
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 3 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 2 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 1 } }
```

Searching the Oplog to find what we added

Search the local database for the corresponding oplog entries.

In the MongoShell, we connect to the instance and then change to the ‘local’ database which stores various instance details including the operations log in the collection ‘oplog.rs’.

Each member keeps their own oplog copy in this collection. The fact that the oplog is a circular buffer means that we know the ordering of the inserted documents can be used with the $natural operator to retrieve them in the order they were written. By using an inverse natural order we can look starting at the end of the oplog and find the latest documents which contain the operations that were written to the collection.

We then search this collection for the last three operations excluding any periodic no operation messages which are frequently logged to track time within the log.

```
db.bar.insertMany([{a:1}, {a:2}, {a:3}])

use local
db.oplog.rs.find({"o.msg": {$ne: "periodic
noop"}}, {op:1,
o:1}).sort({$natural:-1}).limit(3)
```

```
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 3 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 2 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 1 } }
```

What is stored in the Oplog documents

The field **"op"** represents the type of operations and **"o"** represents the document being modified.

The documents in the oplog collection show one document for each of the documents we inserted.

The "op" field is the type of operation, in this case "i" means an insert operation. The "o" field gives the sub document which includes the details on the operation, in this case the document itself we inserted with the ObjectId automatically added for us.

```
db.bar.insertMany([{a:1}, {a:2}, {a:3}])

use local
db.oplog.rs.find({"o.msg": {$ne: "periodic
noop"}}, {op:1,
o:1}).sort({$natural:-1}).limit(3)

{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 3 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 2 } }
{ "op" : "i", "o" : { "_id" :
ObjectId("..."), "a" : 1 } }
```

Idempotency + the Oplog

The operations log (oplog) holds **idempotent** statements that modify the database. MongoDB does this at the document level, each log entry represents one change document.

Idempotency is an important aspect of this log. Idempotent statements can be applied many times without changing the final result after they have been applied the first time.

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.

This approach of logging operations can require a larger log for specific types of workload such as when updating multiple documents at once, when you are deleting the same amount of data as you are inserting, or when there are a significant number of in-place update operations.

Quiz

Quiz

Which kind of information does the operations log (oplog) store and replicate to the other nodes?

- A. Statements that modify the databases
- B. Changes per document modified
- C. Binary file changes to the file storing the database

Quiz

Which kind of information does the operations log (oplog) store and replicate to the other nodes?

- A. Statements that modify the databases
- B. Changes per document modified
- C. Binary file changes to the file storing the database

INCORRECT: Statements that modify the databases - The oplog does not store changes to the configuration of the database.
CORRECT: Changes per document modified - The oplog stores changes per document.
INCORRECT: Binary file changes to the file storing the database - The oplog does not track changes to any specific file at the level of file system changes.

Quiz

Which kind of information does the operations log (oplog) store and replicate to the other nodes?

- A. Statements that modify the databases
- B. Changes per document modified
- C. Binary file changes to the file storing the database

This incorrect. The oplog does not store changes to the configuration of the database.

INCORRECT: Statements that modify the databases - This is incorrect. The oplog does not store changes to the configuration of the database.

Quiz

Which kind of information does the operations log (oplog) store and replicate to the other nodes?

A. Statements that modify the databases

B. Changes per document modified

C. Binary file changes to the file storing the database

This is correct. The oplog stores changes per document.

CORRECT: Changes per document modified - This is correct. The oplog stores changes per document.

Quiz

Which kind of information does the operations log (oplog) store and replicate to the other nodes?

A. Statements that modify the databases

B. Changes per document modified

C. Binary file changes to the file storing the database

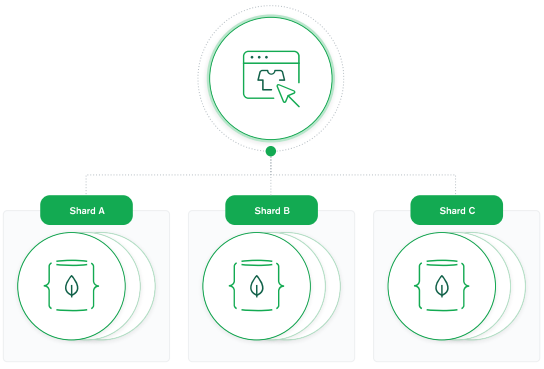
This incorrect. The oplog does not track changes to any specific file at the level of file system changes.

INCORRECT: Binary file changes to the file storing the database - This is incorrect. The oplog does not track changes to any specific file at the level of file system changes.

Sharding

We'll cover sharding in an overview in this lesson. A later lesson has a deeper dive around the mechanisms and best practices of sharding, this section is an introduction to this component within the architecture of MongoDB. The goal of this piece of MongoDB's architecture is to provide a mechanism to scale your database as your data grows beyond the limits of the hardware capacity of any single machine.

MongoDB scales up using sharding to partition data. Each shard consists of a replica set.

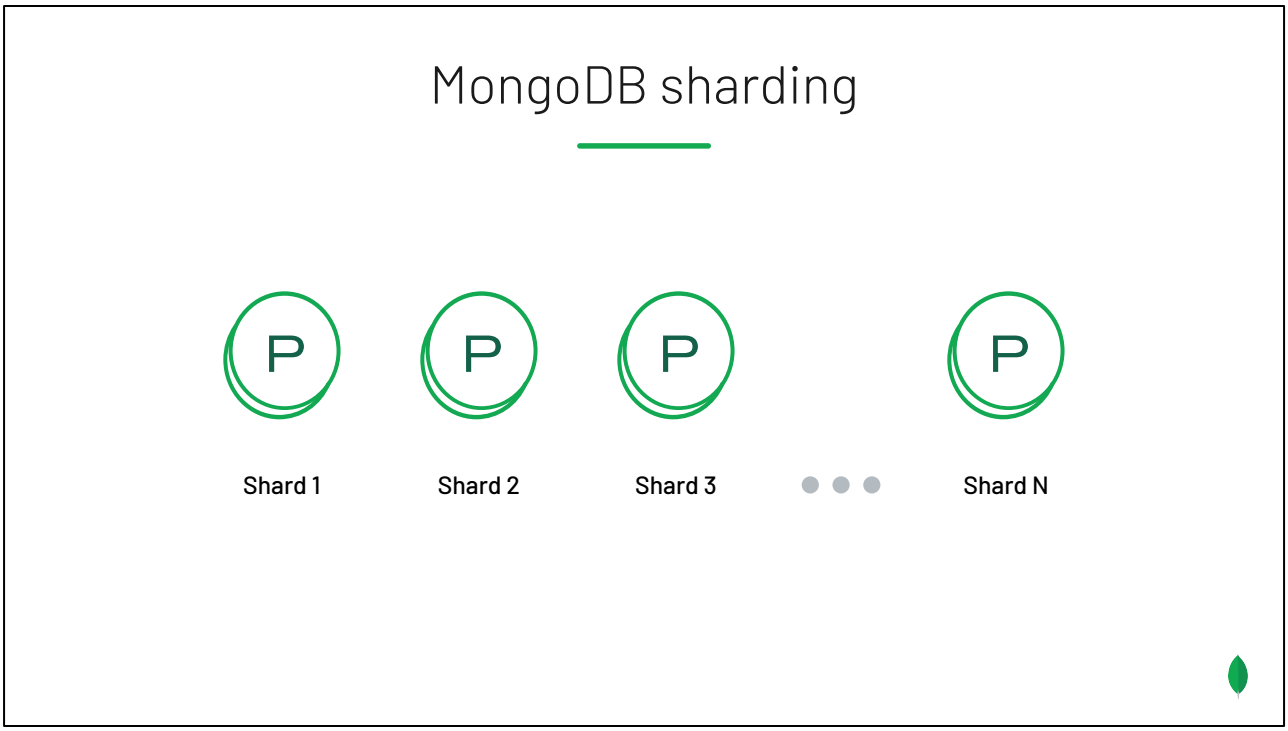


Sharding is how MongoDB and many other non-relational databases partition data to scale horizontally across many machines/instances.

This is done in MongoDB by partitioning the data into key ranges and assigning these key ranges across the machines/instances.

Sharding is how you scale up with MongoDB, it is designed to support very large data sets and high throughput operations.

Let’s look at an overview of sharding and the components behind it in the next slides, we’ll cover sharding itself and how you can use it in your applications in more depth in a later lesson.



Sharding is natively available in MongoDB as an inbuilt feature designed to support the horizontal scale-out of your data.

Sharding helps overcome the constraints in terms of hardware of a single node.

Sharding uses a query layer and provides a transparent (to applications) routing of their requests to the database to the correct shard(s).

Sharding scales and refines the partitioning dynamically in real time and when needed it will also automatically rebalance it.

Sharding also supports distributed transaction, this means that you can design ACID transactions in your MongoDB application and have them safely executed on a sharded cluster.

MongoDB sharding

Native-Sharding for horizontal scale-out

Automatically scale beyond the constraints of a single node

Application transparent

Scale, refine, and rebalance data incrementally, in real time

Distributed transactions

Sharding is natively available in MongoDB as an inbuilt feature designed to support the horizontal scale-out of your data.

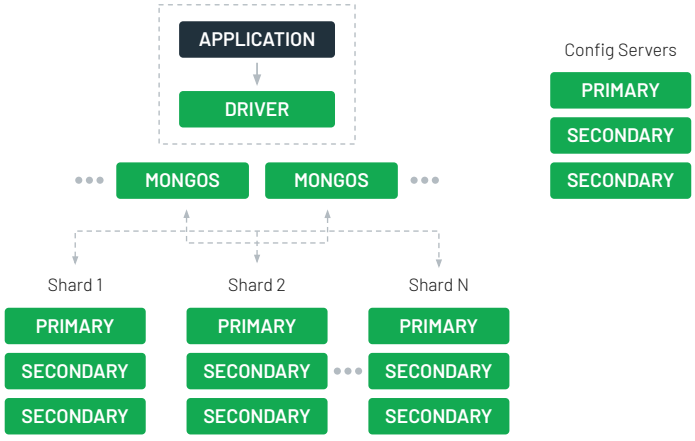
Sharding helps overcome the constraints in terms of hardware of a single node.

Sharding uses a query layer and provides a transparent (to applications) routing of their requests to the database to the correct shard(s).

Sharding scales and refines the partitioning dynamically in real time and when needed it will also automatically rebalance it.

Sharding also supports distributed transaction, this means that you can design ACID transactions in your MongoDB application and have them safely executed on a sharded cluster.

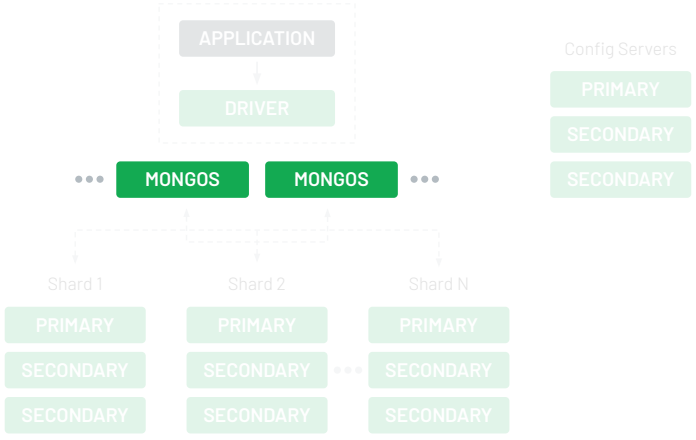
Sharding architecture



MongoDB sharding uses three components:

- shards: Each shard contains a subset of the sharded data. Each shard must be deployed as a replica set.
- mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).

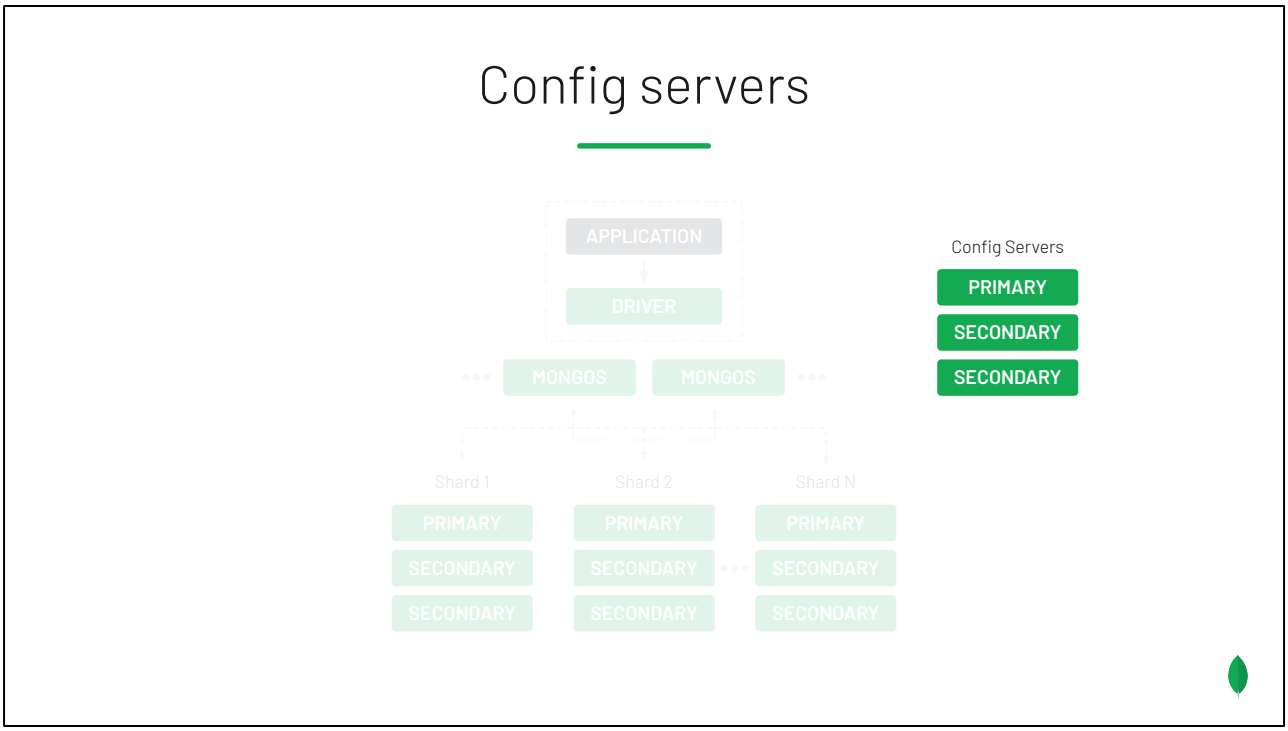
Mongos



MongoDB sharding adds a routing layer with one or more routers called ‘mongos’ these are what your driver/application connect to. Any query or command issued against a shard cluster will go through a ‘mongos’ which will check to determine on which shard(s) the data is and return it back to your application or process the command on the appropriate shard(s).

The key points for a mongos are:

- Caches where the data is located
- Reads it from the config server
- Has no persistent storage
- Speaks the database server protocol

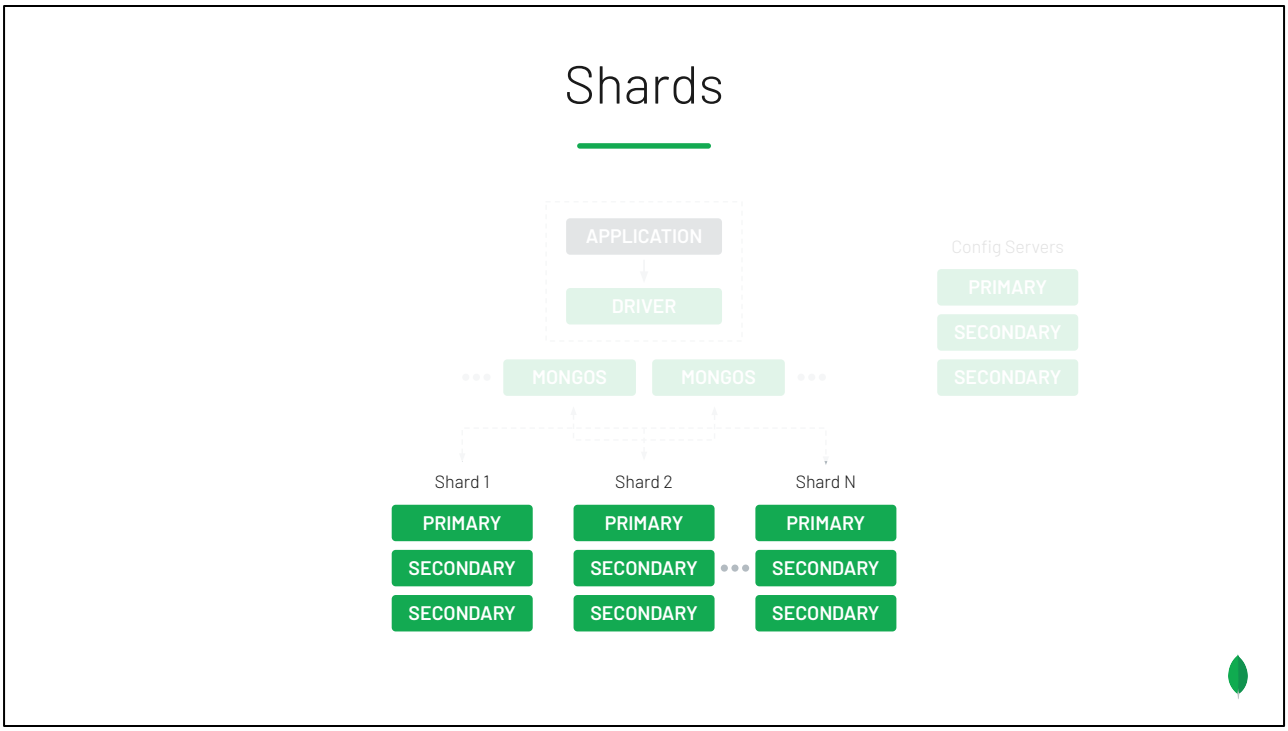


Config servers hold the meta data for all of the shards in your sharded cluster. The metadata reflects state and organization for all data and components within the sharded cluster. The metadata includes the list of chunks on every shard and the ranges that define the chunks. The config servers are all part of a single replica set.

The mongos instances cache this data and use it to route read and write operations to the correct shards. mongos updates the cache when there are metadata changes for the cluster.

In summary for config servers, they:

- Durably persists where data is located
- Uses replication for durability and high availability
- Does not contain any user data, only metadata



A shard contains a subset of sharded data for a sharded cluster. Together, the cluster's shards hold the entire data set for the cluster.
Each shard is itself a replica set.

The key points for a shard are:

- Caches what data it owns (specified by the key range for the shard)
- Reads it from the config server
- Durably persists the actual data of the collection

In our later lesson on sharding, we'll look at how you can use sharding in your application to scale out when and as required.

Hopefully this lesson has given you an understand of the mechanics and mechanisms used in sharding.

Quiz

Quiz

Which of the following are true for MongoDB Architecture?

- A. The database can use different storage engines
- B. Sharding provides high-availability of data
- C. Sharding provides a means of distributing data across multiple machines
- D. Replication provides redundancy for the database both for it as an application and for the data it stores

Quiz

Which of the following are true for MongoDB Architecture?

- ✓ A. The database can use different storage engines
- ✗ B. Sharding provides high-availability of data
- ✓ C. Sharding provides a means of distributing data across multiple machines
- ✓ D. Replication provides redundancy for the database both for it as an application and for the data it stores

CORRECT: The database can use different storage engines - MongoDB can use a variety of underlying storage engines, typically the default storage engine is WiredTiger but you can use others.
INCORRECT: Sharding provides high-availability of data - Replication is how high available of data is provided, sharding is a mechanism to scale data by partitioning it across machines.
CORRECT: Sharding provides a means of distributing data across multiple machines - Sharding partitions data across multiple machines, in MongoDB each partition is called a shard and is backed by a replica set.
CORRECT: Replication provides redundancy for the database both for it as an application and for the data it stores - Replication provides multiple copies of the data for redundancy, it further ensures that in the case of the failure of the primary node that an election will occur ensuring a new primary is elected which provides redundancy for your application.

Quiz

Which of the following are true for MongoDB Architecture?

- ✓ A. The database can use different storage engines
- ✗ B. Sharding provides high-availability of data
- ✓ C. Sharding allows for distributing data across multiple machines
- ✓ D. Replication provides database redundancy, both for the application and for the data it stores

This is correct. MongoDB can use a variety of underlying storage engines, typically the default storage engine is WiredTiger but you can use others.

CORRECT: The database can use different storage engines - This is correct. MongoDB can use a variety of underlying storage engines, typically the default storage engine is WiredTiger but you can use others.

Quiz

Which of the following are true for MongoDB Architecture?

- ✓ A. The database can use different storage engines
- ✗ B. Sharding provides high-availability of data
- ✓ C. Sharding allows for distributing data across multiple machines
- ✓ D. Replication provides database redundancy, both for the application and for the data it stores

This incorrect. Replication is how high available of data is provided, sharding is a mechanism to scale data by partitioning it across machines.

INCORRECT: Sharding provides high-availability of data - This is incorrect. Replication is how high available of data is provided, sharding is a mechanism to scale data by partitioning it across machines.

Quiz

Which of the following are true for MongoDB Architecture?

- ✓ A. The database can use different storage engines
- ✗ B. Sharding provides high-availability of data
- ✓ C. **Sharding allows for distributing data across multiple machines**
- ✓ D. Replication provides database redundancy, both for the application and for the data it stores

This is correct. Sharding partitions data across multiple machines, in MongoDB each partition is called a shard and is backed by a replica set.

CORRECT: Sharding provides a means of distributing data across multiple machines - This is correct. Sharding partitions data across multiple machines, in MongoDB each partition is called a shard and is backed by a replica set.

Quiz

Which of the following are true for MongoDB Architecture?

- ✓ A. The database can use different storage engines
- ✗ B. Sharding provides high-availability of data
- ✓ C. Sharding allows for distributing data across multiple machines
- ✓ D. Replication provides database redundancy, both for the application and for the data it stores

This is correct. Replication provides multiple copies of the data for redundancy, it further that in the case of the failure of the primary node that an election will occur ensuring a new primary to provide redundancy.

CORRECT: Replication provides redundancy for the database both for it as an application and for the data it stores - This is correct. Replication provides multiple copies of the data for redundancy, it further that in the case of the failure of the primary node that an election will occur ensuring a new primary to provide redundancy.

Continue Learning!

MongoDB University has free self-paced courses and labs ranging from beginner to advanced levels.

Github Student Developer Pack

Sign up for the MongoDB Student Pack to receive $50 in Atlas credits and free certification!

This concludes the material for this lesson. However, there are many more ways to learn about MongoDB and non-relational databases, and they are all free! Check out MongoDB's University page to find free courses that go into more depth about everything MongoDB and non-relational. For students and educators alike, MongoDB for Academia is here to offer support in many forms. Check out our educator resources and join the Educator Community. Students can receive $50 in Atlas credits and free certification through the Github Student Developer Pack.