Learn MongoDB 4.x

A guide to understanding MongoDB development and administration for NoSQL developers

Packt>

www.packt.com

Doug Bierer

Learn MongoDB 4.x

A guide to understanding MongoDB development and administration for NoSQL developers

Doug Bierer

BIRMINGHAM - MUMBAI

Learn MongoDB 4.x

Commissioning Editor: Pravin Dhandre
Acquisition Editor: Savia Lobo
Content Development Editor: Pratik Andrade
Senior Editor: Ayaan Hoda
Technical Editor: Sarvesh Jaywant
Copy Editor: Safis Editing
Project Coordinator: Neil Dmello
Proofreader: Safis Editing
Indexer: Rekha Nair
Production Designer: Alishon Mendonca

First published: September 2020

Production reference: 1030920

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78961-938-6

www.packt.com

This year has been extraordinarily tough, not only due to the global pandemic, but also as a result of deteriorating conditions worldwide as a result of climate change and increasing economic inequality across the globe. That having been said, on a personal note, three near and dear friends have died in the period of time it took to write this book. I would like to dedicate this book to those three: Jeff Abel, Daryl Holdridge, and Brad Saunders. Dear friends, I hope you all find yourselves in a better place. May you rest in peace.

– Doug Bierer

1

Introducing MongoDB 4.x

In this book, we cover how to work with a MongoDB 4.x database, starting with the simplest concepts and moving on to more complex ones. The book is divided into parts or sections, each of which looks at a different scenario.

In this chapter, you are given a general introduction to MongoDB 4.x with a focus on new features and a brief high-level overview of the technology. We also discuss security enhancements, along with backward-incompatible changes that might cause an application written for MongoDB 3 to break after an upgrade to MongoDB 4.x.

In the next chapter, a simple scenario is introduced: a fictitious company called *Sweets Complete Inc.* that sells confections online to a small base of international customers. In the next two parts that follow, you are introduced to *BookSomething.com*, another fictitious company with a large database of hotel listings worldwide. Finally, in the last part, you are introduced to *BigLittle Micro Finance Ltd.*, a fictitious company that connects lenders with borrowers and deals with a massive volume of geographically dispersed data.

In this chapter, the following topics are covered:

- A high-level technology overview of MongoDB 4.x
- Significant new features introduced in MongoDB 4.x
- Important security enhancements
- Spotting and avoiding potential problems when migrating from MongoDB 3 to 4

High-level technology overview of MongoDB 4.x

When it was first introduced in 2009, MongoDB took the database world by storm, and since that time it has rapidly gained in popularity. According to the 2019 StackOverflow developer survey (https://insights.stackoverflow.com/survey/2019#technology-_- databases), MongoDB is ranked fifth, with 26% of professional developers and 25.5% of all respondents saying they use MongoDB. DB-Engines (https://db-engines.com/en/ ranking) also ranks MongoDB as the fifth most widely used database, using an algorithm that takes into account the frequency of search, DBA Stack Exchange and StackOverflow references, and the frequency with which MongoDB appears in job postings. What is of even more interest is that the trend graph generated by DB-Engines shows that the score (and therefore ranking) of MongoDB has grown by 200% since 2013. You can refer to https://db-engines.com/en/ranking_trendfor more details. In 2013, MongoDB was not even in the top 10!

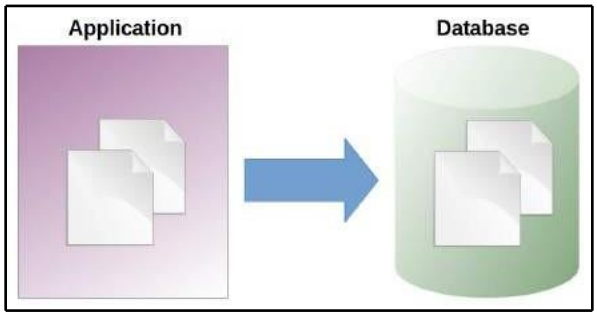
There are many key features of MongoDB that account for its rise in popularity. Subsequent chapters in this book cover the most important of these features in detail. In this section, we present you with a brief, *big-picture* overview of three key aspects of MongoDB.

MongoDB is based upon documents

One of the most important distinctions between MongoDB and the traditional **relational database management systems** (**RDBMS**) is that instead of tables, rows, and columns, the basis for storage in MongoDB is a *document*. In a certain sense, you can think of the traditional RDBMS system as *two dimensional*, whereas MongoDB is *three dimensional*. Documents are typically modeled using JSON formatting and then inserted into the database where they are converted to a binary format for storage (more on that in later chapters!).

Related to the document basis for storage is the fact that MongoDB documents have *no fixed schema*. The main benefit of this is *vastly reduced overhead*. Database restructuring is a piece of cake, and doesn't cause the massive problems, website crashes, and security breaches seen in applications reliant upon a traditional RDBMS database restructuring.

The really great news for developers is that most modern programming applications are based on classes representing information that needs to be stored. This has spawned the creation of a large number of ***object-relational mapping*** (**ORM**) libraries for the various programming languages. In MongoDB, on the other hand, the need for a complex ORM infrastructure is completely eliminated as programmatic objects can be directly stored in the database as-is:

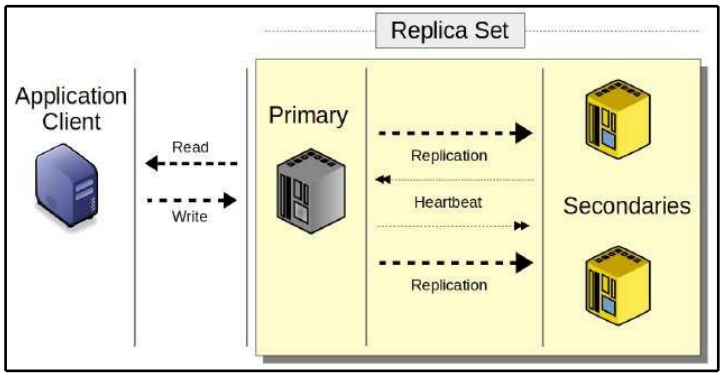


So instead of columns, MongoDB documents have *fields*. Instead of tables, there are *collections* of documents. Let's now have a look at replication in MongoDB.

High availability

Another feature that causes MongoDB to stand out from other database technologies is its ability to ensure *high availability* through a process known as *replication*. A server running MongoDB can have copies of its databases duplicated across two more servers. These copies are known as *replica sets*. Replica sets are organized through an election process whereby the members of the replica *vote* on which server becomes the *primary*. Other servers are then assigned the role of *secondary*.

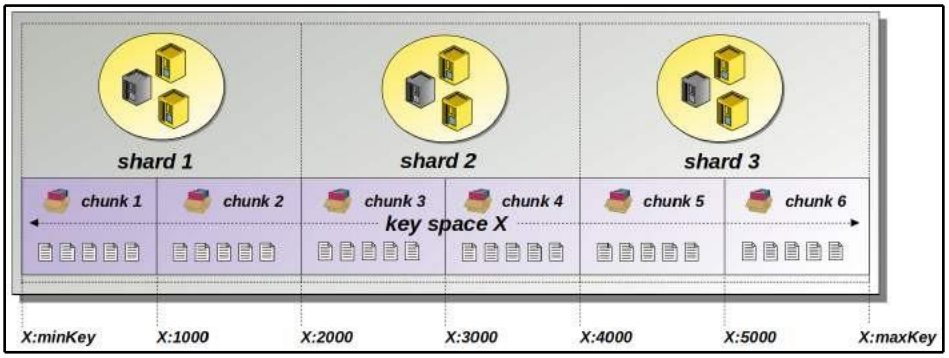
This arrangement not only ensures that the database is continuously available, but that it can also be used by application code by way of *read preferences*. A read preference tells the replica set which servers in the replica set are preferred. If the read preferences are set less restrictively, then the first server in the set to respond might be able to satisfy the request, thereby implementing a form of parallel process that has the potential to greatly enhance performance. This setup is illustrated in the following diagram:



This topic is covered in extensive detail in `Chapter13`, *Deploying a Replica Set*. Lastly, we have a look at *sharding*.

Horizontal scaling

One more feature, among many, is MongoDB's ability to handle a massive amount of data. This is accomplished by splitting up a sizeable collection across multiple servers, creating a *sharded cluster*. In the process of splitting the collection, a *shard key* is chosen from among the fields present with the collection's documents. The shard key is then used by the *sharded cluster balancer* to determine the appropriate distribution of documents. Application program code is then able, by its knowledge of the value of the shard key, to direct queries to specific members of the sharded cluster, achieving potentially enormous performance gains. This setup is illustrated in the following diagram:



This topic is covered in extensive detail in Chapter 15, *Deploying a Sharded Cluster*. In the next section of this chapter, we have a look at the major differences between MongoDB 3 and MongoDB 4.x.

Discovering what's new and different in MongoDB 4.x

What's new and different in the MongoDB 4.x release can be broken down into two main categories: new features and internal enhancements. Let's look at the most significant new features first.

Significant new features

The most significant new features introduced in MongoDB 4.x include the following:

- Multidocument ACID transaction support
- Nonblocking secondary replica reads
- In-progress index build interruption

Transactions, secondary replica reads, and the aggregation pipeline are covered in detail in later chapters of this book. For an excellent brief overview of the major changes from MongoDB 3 to 4, go to https://www.mongodb.com/blog/post/mongodb-40-release-candidate- 0-has-landed.

Multidocument ACID transaction support

In the database world, a *transaction* is a block of database operations that should be treated as if the entire block of commands was just a single command. An example would be where your application is performing end-of-the-month payroll processing. In order to maintain the integrity of the database, and your accounting files, you would need this set of operations to be safeguarded in the event of a failure. ACID is an acronym that stands for *atomicity*, *consistency*, *isolation*, and *durability*. It represents a set of principles that the database needs to follow in order to safeguard a block of database updates. For more information on ACID, you can refer to https://en.wikipedia.org/wiki/ACID.

In MongoDB 3, a write operation on a single document, even a document containing other embedded documents, was considered *atomic*. In MongoDB 4.x, multiple documents can be included in a single atomic transaction. Although invoking this support negatively impacts performance, the gain in database integrity might prove attractive. It's also worth noting that the lack of such support prior to MongoDB 4.x was a major criticism leveled against MongoDB, and slowed its adoption at the corporate level.

Invoking transaction support impacts *read preferences* and *write concerns*. For more information, you can refer to https://docs.mongodb.com/manual/core/transactions/ #transaction-options-read-concern-write-concern-read-preference. Although these topics are covered in detail later in the book, we can briefly summarize them by stating that read preferences allow you to direct operations to specific members of a replica set. For example, you might want to indicate a preference for the *primary* member server in a replica set rather than allowing any member, including *secondaries*, to be used in a read operation. Write concerns allow you to adjust the level of acknowledgement when writing to the database, thereby ensuring that data integrity is maintained. In MongoDB 4.x, you are able to set read preferences and write concerns at the transaction level, that in turn influences individual document operations.

In MongoDB version 4.2 and above, the 16 MB limit on transaction size is removed. Also, as of MongoDB 4.2, full support for multidocument transactions is added for sharded clusters. In addition, full transaction support is extended to replica sets whose secondary members are using the *in-memory* storage engine (https://docs.mongodb.com/manual/core/ inmemory/).

Replica sets are discussed in Chapter 13, *Deploying a Replica Set*. Sharded clusters are covered in Chapter 15, *Deploying a Sharded Cluster*.

Nonblocking secondary reads

MongoDB developers have often included read concerns (as mentioned previously) in their operations in order to shift the burden of response from the primary server in a replica set to its secondaries instead. This frees up the primary to process write operations. Traditionally, MongoDB, prior to version 4, blocked such secondary reads while an update from the primary was in progress. The block ensured that any data read from the secondary would appear exactly the same as data read from the primary.

The downside to this approach, however, was that while the block was in place, the secondary read operation had to wait, which in turn negatively impacted read performance. Likewise, if a read operation was requested prior to a write, it would hold up update operations between the primary and secondary, negatively impacting write performance.

Because of internal changes introduced in MongoDB 4.x to support multidocument transactions, storage engine timestamps and snapshots are now used, which has the side effect of eliminating the need to block secondary reads. The net effect is an overall improvement in consistency and lower latency in terms of reads and writes. Another way to view this enhancement is that it allows an application to read from a secondary at the same time writes are being applied without delay.

In-progress index build interruption

A big problem with versions of MongoDB prior to 4.4 is that the following commands error out if an *index build* (https://docs.mongodb.com/master/core/index-creation/#index- builds-on-populated-collections) operation is in progress:

- db.dropDatabase() db.collection.drop()
- db.collection.dropIndexes()
-

In MongoDB 4.4, when this happens, an attempt to force the in-progress index build operation is made. If successful, the index build is halted, and the drop*()operation continues without error. In the case of a drop*()performed on a replica set, the abort attempt is made on the primary. Once the primary commits to the abort, it then synchronizes to the secondaries.

Other noteworthy new features

There are a number of other new features that do not represent a massive paradigm shift, but are extremely useful nonetheless. These include improvements to the aggregation pipeline, field-level encryption, password()prompt, and wildcard indexes. Let's first have a look at aggregation pipeline improvements.

Aggregation pipeline type conversions

Another major new feature we discuss here involves the introduction of $convert, a new aggregation pipeline operator. This new operator allows the developer to change the data type of a document field while being processed in the pipeline. Target data types include *double*, *string*, *Boolean*, *date*, *integer*, *long*, and *decimal*. In addition, you can convert a field in the pipeline to the data type objectId, which is critically useful when you need direct access to the autogenerated unique identification field _id. For more information on the aggregation pipeline operator and $convert, go to https://docs.mongodb.com/master/core/aggregation-pipeline/#aggregation-pipeline.

Client-side field-level encryption

The official programming language drivers for MongoDB 4.2 now support *client-side field-level encryption*. The implications for security improvements are enormous. This enhancement means that your applications can now provide end-to-end encryption for transmitted data down to the field level. So you could have a transmission of data from your application to MongoDB that includes, for example, an encrypted national identification number mixed in with otherwise plain-text data.

You can refer to https://docs.mongodb.com/master/core/security- client-side-encryption/#driver-compatibility-table to access the drivers for MongoDB 4.2. For client-side field-level encryption, you can refer to https://docs.mongodb.com/master/core/security-client- side-encryption/#client-side-field-level-encryption.

Password prompt

In many cases, it is highly undesirable to include a hard-coded password in a *Mongo* script. Starting with MongoDB 4.2, in place of a hard-coded password value, you can substitute a built-in JavaScript function passwordPrompt(). You can refer to https://docs.mongodb.com/master/reference/method/passwordPrompt/#passwordPrompt for more details on the function. This causes the Mongo shell to pause and wait for manual user input before proceeding. The password that is entered is then used as the password value.

Wildcard indexes

Starting with MongoDB 4.2, support has been added for *wildcard indexes* (https://docs.mongodb.com/master/core/index-wildcard/#wildcard-indexes). This feature is useful for situations where the index is either not yet available or is unknown. For situations where the field is known and well established, it's best to create a normal index. There are cases, however, where you have a subset of documents that contain a particular field otherwise lacking in other documents in the collection. You might also be in a situation where a field is added later, and where the DBA has not yet had a chance to create an index on the new field. In these cases, adding a wildcard index allows MongoDB to perform a query more efficiently.

Extended JSON v2 support

Starting with MongoDB 4.2, support for the Extended JSON v2 (https://docs.mongodb.com/master/reference/mongodb-extended-json/#mongodb-extended-json-v2) specification has been enabled for the following utilities:

- bsondump
- mongodump
- mongoexport
- mongoimport

Improved logging and diagnostics

Starting with MongoDB 4.2, there are now five verbosity log levels, each revealing increasing amounts of information. In MongoDB 4.0.6, you can now set a threshold on the maximum time it should take for data to replicate between members of a replica set. It's now possible to get the information from the MongoDB log file if that time is exceeded.

A further enhancement to diagnostics capabilities includes additional fields that are added to the output of the `db.serverStatus()` command that can be issued from a Mongo shell.

Hedged reads

MongoDB 4.4 adds the ability to perform a *hedged read* (https://docs.mongodb.com/master/core/sharded-cluster-query-router/#hedged-reads) on a sharded cluster. By setting a *hedged read preference* option, applications are able to direct read requests to servers in replica sets other than the primary. The advantage of this approach is that the application simply takes the first result response, improving performance. The potential cost, of course, is that if a secondary responds, the data might be slightly out of date.

TCP fast open support

MongoDB version 4.4 introduces support for **TCP Fast Open** (**TCO**) connections. For this to work, it must be supported by the operating system hosting MongoDB. The following configuration file (and command line) parameters have been added to enable and control support under the setParameter configuration option: tcpFastOpenServer, tcpFastOpenClient, and tcpFastQueueSize. In addition, four new TCO-related information counters have been added to the output of the serverStatus() database command.

You can refer to https://tools.ietf.org/html/rfc7413for more details on TCO. For more information on the parameter, you can refer to https:/ /docs.mongodb.com/master/reference/parameters/#param. tcpFastOpenServer. Refer to https://docs.mongodb.com/master/ reference/command/serverStatus/#serverstatus for more information on serverStatus.

Natural sort

MongoDB version 4.4 introduces a new operator, $natural, which is used in a cursor.hint()operation. This operator causes the results of a sort operation to return a list in natural (also called human-readable) order. As an example, take these values:

```
['file19.txt','file10.txt','file20.txt','file8.txt']
```

An ordinary sort would return the list in this order:

```
['file10.txt','file19.txt','file20.txt','file8.txt']
```

Whereas a *natural* sort would return the following:

```
['file8.txt','file10.txt','file19.txt','file20.txt']
```

Internal enhancements

The first enhancement we examine is related to nonblocking secondary reads (as mentioned earlier). After that, we cover shard migration, authentication, and stream enhancements.

Timestamps in the storage engine

One of the major new features introduced in MongoDB version 3 was the integration of the *WiredTiger* storage engine. Prior to December 2014, *WiredTiger Inc*. was a company that specialized in database storage engine technology. Its impressive list of customers included *Amazon Inc*. In December 2014, *WiredTiger* was acquired by MongoDB after partnering with them on multiple projects.

In MongoDB version 3, multidocument transactions were not supported. Furthermore, in the replication process (https://docs.mongodb.com/manual/replication/#replication), changes accepted by the primary server in a replica set were pushed out to the secondary servers in the set, which were largely controlled through *oplogs* (https://docs.mongodb. com/manual/core/replica-set-oplog/#replica-set-oplog) and programming logic outside of the storage engine. Simply stated, the oplog represents changes made to the database. When a secondary synchronizes with a primary, it creates a copy of the oplog and then applies the changes to its own local copy of the database. The logic in place that controlled this process in MongoDB 3 was quite complicated and consumed resources that could otherwise have been used to satisfy user requests.

In MongoDB 4, the internal update mechanism of *WiredTiger,* the storage engine, was rewritten to include a timestamp in each update document. The main reason for this change was to provide support for multidocument transactions. It was soon discovered, however, that this seemingly simple change could potentially revolutionize the entire replication process.

In MongoDB 4, much of the logic required to ensure data integrity during replication synchronization has now been shifted to the storage engine itself, which in turn frees up resources to service user requests. The net effect is threefold: improved data integrity, read operations producing a more up-to-date set of documents, and improved performance.

For an excellent in-depth explanation of how timestamps work in the WiredTiger storage engine, have a look at the video at https://www. mongodb.com/presentations/wiredtiger-timestamps-enforcing- correctness-in-operation-ordering-across-the-distributed- storage-layer, which features Dr. Michael Cahill, formerly of *WiredTiger, Inc*., now Director of Engineering at MongoDB

Shard migration

Typically, DevOps engineers distribute the database into shards to support a massive amount of data. There comes a time, however, when the data needs to be moved. For example, let's say a host server needs to be upgraded or replaced. In MongoDB version 3.2 and earlier, this process could be quite daunting. In one documented case, a 500 GB shard took *13 days* to migrate. In MongoDB 3.4, parallelism support was provided that sped up the migration process. Part of the reason for the improvement was that the *chunk balancer* logic was moved to the *config server* (https://docs.mongodb.com/manual/core/sharded- cluster-config-servers/#config-servers), which must be configured as part of a replica set.

Another improvement, available with MongoDB 4.0.3, allows the *sharded cluster balancer* (https://docs.mongodb.com/manual/core/sharding-balancer-administration/ #sharded-cluster-balancer) to preallocate *chunks* if *zones and ranges* have been defined, which facilitates rapid capacity expansion. DevOps engineers are able to add and remove nodes from a sharded cluster in real time. The sharded cluster balancer handles the work of rebalancing data between the nodes, thereby alleviating the need for manual intervention.
This gives DevOps engineers the ability to scale database capacity up or down on demand. This feature is especially needed in environments that experience seasonal shifts in demand. An example would be a retail outlet that needs to scale up its database capacity to support increased consumer spending during holidays.

For more information on chunks, refer to https://docs.mongodb.com/manual/core/sharding-data-partitioning/#data-partitioning-with- chunks, and for more information on zones and ranges, refer to https://docs.mongodb.com/manual/core/zone-sharding/.

Change streams

As the database is updated, changes are recorded in the *oplog* maintained by the primary server in the replica set, which is then used to replicate changes to the secondaries. Trying to read a list of changes via the oplog is a tedious and resource-intensive process, so many developers choose to use *change streams* (https://docs.mongodb.com/manual/changeStreams/?jmp=blog_ga=2.5574835.1698487790.1546401611-137143613.1528093145#change-streams) to subscribe to all changes on a collection. For those of you who are familiar with software design patterns, this is a form of the publish/subscribe pattern.

Aside from their obvious use in troubleshooting and diagnostics, changing streams can also be used to give an indicator of whether or not data changes are durable.

What is new and different in MongoDB 4.x is the introduction of a startAtOperationTime parameter that allows you to specify the timestamp at which you wish to tap into the change stream. This timestamp can also be in the past, but cannot extend beyond what is recorded in the current oplog.

If you enter 4.0 as a value of another parameter, featureCompatibilityVersion, then the streams return token data, used to restart the stream, in the form of a hex-encoded string, which gives you greater flexibility when comparing blocks of token data. An interesting side effect of this is that a replica set based on MongoDB 4.x could theoretically make use of a change stream token opened on a replica set based on MongoDB 3.6. Another new feature in MongoDB 4 is that change streams that are opened on multidocument transactions include the transaction number.

Important new security enhancements

There were many security improvements introduced in MongoDB 4, but here, we highlight the two most significant changes: support for SHA-256 and **transport layer security** (**TLS**) handling.

SHA-256 support

SHA stands for ***secure hash algorithm***. SHA-256 is a hash function (https://csrc.nist.gov/Projects/Hash-Functions) derivative of the SHA-2 family. The significance of offering SHA-256 support is based on the difference between the SHA-1, which MongoDB supports, and SHA-2 families of hash algorithms. SHA-1, introduced in 1995, used algorithms similar to an older family of hash functions: MD2, MD4, and MD5. SHA-1, however, produces a hash value of 160 bits compared with 128 for the MDx series. SHA-256, introduced in 2012, increases the hash value size to 256, which makes it exponentially more difficult to crack. Attack vectors that could compromise communications based upon SHA-1 and SHA-2 include the preimage attack, the collision attack, and the length-extension attack.

The first attack relies upon *brute-force* attack methods to reverse the hash. In the past, this required computational power beyond the reach of anyone other than a well-funded organization (for example, a government agency or a large corporation). Today, a normal desktop computer could have multiple cores, plenty of memory, and a graphics processing unit (GPU) that are easily capable of such attacks. To launch the attack, the attacker would need to be in a place where access to the database itself is possible, which means that other layers of security (such as the firewall) would have first been breached.

A collision attack uses two different messages that produce the same hash. Once the match has been found, it is mathematically possible to interfere with TLS communications. The attacker could, for example, start forging signatures, which would wreak havoc on systems dependent on digitally signed documents. The danger of this form of attack is that it can theoretically be successfully launched in half the number of iterations compared with a preimage attack.

At the time of writing, the SHA-256 hash function is immune to both preimage and collision attacks; however, both the SHA-1 and SHA-2 family of hash functions, including SHA-256, are vulnerable to length-extension attacks. This attack involves adding to the message, thereby extending its length and then recalculating the hash. The modified message is then seen as valid, allowing the attacker a way into the communication stream. Unfortunately, even though SHA-256 is *resistant* to this form of attack, it is still vulnerable.

It might be of interest to note that Bitcoin uses SHA-256 for the verification of transactions.

TLS handling

Transport layer security (**TLS**) was introduced in 1999 to address serious vulnerabilities inherent in all versions of the Secure Sockets Layer (SSL). It is highly recommended that you secure your MongoDB installations with TLS 1.1 or above (covered later in this book). Once you have configured your *mongod* instances to use TLS, all communications are affected. These include communications between clients, drivers, and the server, as well as internal communications between members of a replica set and between nodes in a sharded cluster.

TLS security depends on which block cipher algorithm and mode are selected. For example, the *3DES* (Data Encryption Standard 3) algorithm with the **Cipher Block Chaining** (**CBC**) mode are considered vulnerable to attack even in TLS version 1.2! The **Advanced Encryption Standard** (**AES**) algorithm and **Galois Counter Mode** (**GCM**) are considered a secure combination, but are only supported in TLS versions 1.2 and 1.3 (ratified in 2018). It should be noted, however, that the AES-256 and GCM combination is not supported when running the MongoDB Enterprise edition on a Windows server.

Using any form of SSL with MongoDB is now deprecated. TLS 1.0 support is also disabled in MongoDB 4.x and above. Ultimately, the version of TLS you end up using in your MongoDB installation completely depends on what cryptographic libraries are available for the server's operating system. This means that as you upgrade your OS and refresh your MongoDB 4+ installation, TLS support is also automatically upgraded. Currently, MongoDB 4+ uses *OpenSSL* on Linux hosts, *Secure Channel* on Windows, and *Secure Transport* on the Mac.

As of MongoDB 4.4, the *Mongo* shell now issues a warning if the x.509 certificate is due to expire within the next 30 days. Likewise, you now see log file messages if there is a pending certificate expiration between *mongod* instances in a sharded cluster or replica set.

See https://www.mongodb.com/blog/post/exciting-new-security- features-in-mongodb-40for a good discussion of security features that were introduced with MongoDB 4.0.

Avoiding problems when upgrading from MongoDB 3.x to 4.x

If you are not familiar with the concept of *backward incompatibilities*, then you have probably not yet survived a major upgrade! To give you an idea of how important it is to be aware of this when reviewing the *change log* for MongoDB, this concept is also referred to as *code breaks*...as in *things that can break your code*.

The article at https://docs.mongodb.com/manual/release-notes/4.0- compatibility/ covers the full list of compatibility changes in MongoDB 4.0. For information on MongoDB 4.2 compatibility changes, go to https://docs.mongodb.com/master/release-notes/4.2-compatibility/#compatibility-changes-in-mongodb-4-2. For information on MongoDB 4.4 compatibility changes, go to https://docs.mongodb.com/master/release-notes/4.4/#changes-affecting- compatibility.

MMAPv1 storage engine

MMAPv1 (https://docs.mongodb.com/manual/storage/#mmapv1-storage-engine), the original MongoDB storage engine, has been deprecated in MongoDB 4.0 and removed as of MongoDB 4.2. It has been replaced by *WiredTiger*, which has been available since MongoDB version 3.0. WiredTiger has been the default since MongoDB version 3.2, so there is a good chance that this backward-incompatible change does not affect your applications nor installation.

If your installation was using the MMAPv1 storage engine before the upgrade, then you immediately notice more efficient memory and disk-space allocation. Simply put, MMAPv1 grabbed as much free memory as it could, and would allocate additional disk space with an insatiable appetite. This made a MongoDB 3 installation using MMAPv1 a *bad neighbor* on a server that is also doing other things!

Another difference that DevOps engineers appreciate is that embedded documents no longer continue to grow in size after being created. This was an unwanted side effect produced by the MMAPv1 storage engine, which could have potentially affected write performance and lead to data fragmentation.

If, as a result of an upgrade from MongoDB 3 to 4, you are in the unfortunate position of having to update a database that is stored using the MMAPv1 storage engine, then you *must* manually back up the data on each server prior to the MongoDB 4.x upgrade. After the upgrade has completed, you then need to perform a manual restore.

A detailed discussion on backup and restore is given in Chapter3, *Essential MongoDB Administration Techniques*.

For a good nonbiased comparison of MMAPv1 and WiredTiger, go to https://stackoverflow.com/questions/37985134/how-to-choose- from-mmapv1-wiredtiger-or-in-memory-storageengine-for-mongodb.

Replica set protocol version

Communications between members of a replica set are governed by an internal protocol simply referred to as pv0or pv1. pv0was the original protocol. Prior to MongoDB 3.2, the only version available was pv0. MongoDB 4.x dropped support for pv0and only supports pv1. Accordingly, before you perform an upgrade to MongoDB 4, you must reconfigure all replica sets to pv1 (https://docs.mongodb.com/manual/reference/replica-set- protocol-versions/#modify-replica-set-protocol-version).

Fortunately, this process is quite easy, and can be accomplished by this simple procedure. These steps must then be repeated for each replica set: verify oplog entry replication and upgrade to pv1. Let's go into more detail regarding these two steps.

Verifying that at least one oplog entry has replicated

These steps must be performed on each *secondary* in the replica set:

1. Use the mongoshell to connect to each *secondary* in the replica set:

```
mongo --host <address of secondary>
```

2. Once connected to the *secondary*, run the rs.status();command and check the values of the optimes::appliedOpTime::tkey.
3. Repeat this for each *secondary* and confirm that the tvalue is greater than -1. This tells us that at least one oplog entry has replicated from the *primary* to all secondaries.

Upgrading the primary to protocol version 1

You can now upgrade the replica set protocol version to pv1:

1. Use the mongo(https://docs.mongodb.com/manual/mongo/#the-mongo-shell) shell to connect to the primary in the replica set:

```
mongo --host <address of primary>
```

2. Once connected to the *primary*, run these commands to update the protocol version for the replica set:

```
cfg = rs.conf();
  cfg.protocolVersion=1;
  rs.reconfig(cfg);
```

Feature compatibility

A number of the new features available in MongoDB 4.x only work if you update the setFeatureCompatibilityVersion parameter (https://docs.mongodb.com/master/reference/command/setFeatureCompatibilityVersion/ #setfeaturecompatibilityversion) in the admin database. The new features affected include the following:

- SCRAM-SHA-256
- New type conversion operators and enhancements
- Multidocument transactions
- $dateToStringoption changes New
- change stream methods
- Change stream resume token data type changes

To view the current featureCompatibilitysetting, go through the following steps:

1. Use the mongoshell to connect to your database as a user who has the rights to modify the admin database:

```
mongo --username <name of user> --password
```

2. Use this command to view the current setting:

```
db.adminCommand({getParameter:1, featureCompatibilityVersion:1})
```

To perform the update, go through the following steps:

1. Use the mongoshell to connect to your database as a user who has the rights to modify the admin database:

```
mongo --username <name of user> --password
```

2. You can then update this parameter as follows, substituting 4.0, 4.2, 4.4, and so on in place of <VERSION>:

```
db.adminCommand( { setFeatureCompatibilityVersion: "<VERSION>" } )
```

It is important to note that the mongos binary crashes where the binary version and/or feature compatibility version of mongosis lower than the connected mongod instances.

User authentication

When establishing security for database users, you have a choice of several different approaches. One of the most popular approaches is *challenge-response*. Simply put: the database *challenges* the user to prove their identity. The response (in most cases), is a username and password combination. In MongoDB 3, this popular approach was implemented by default using MONGODB-CR (MongoDB Challenge Response). As of MongoDB 4, this mechanism is no longer available. This means that when you upgrade from MongoDB 3 to MongoDB 4, you *must* implement at least its replacement, **Salted Challenge Response Authentication Method** (**SCRAM**).

An alternative would be to use x.509 certificates, covered in Chapter 11, *Administering MongoDB Security*.

If your user credentials are in MONGODB-CR format, then you must use the following command to upgrade to *SCRAM* format:

```
db.adminCommand({authSchemaUpgrade: 1});
```

It is critical that you perform this upgrade *while still running MongoDB 3*. The reason for this is that the authSchemaUpgrade parameter has been removed in MongoDB 4! Another side effect of upgrading to SCRAM is that your application driver might also be affected. Have a look at the SCRAM Driver Support table in the documentation at https://docs.mongodb.com/manual/core/security-scram/#driver-support to be sure. The minimum programming language driver for Python that supports SCRAM authentication, for example, is version 2.8.

It is *extremely important* to note that when upgrading to SCRAM from MONGODB-CR, the old credentials are *discarded*, which means that this process is *irreversible*. A good overview of the upgrade from MONGODB-CR to SCRAM can be found at https://docs.mongodb.com/manual/ release-notes/3.0-scram/index.html#upgrade-to-scram. Please note that this upgrade process *only works* on MongoDB 3. You need to perform this upgrade *before* you upgrade the version of MongoDB.

For even more security when implementing SCRAM, you now have the option of using SHA-256 instead of SHA-1 (see the previous information).

Removed and deprecated items

Any command or executable binary that is *removed* can potentially cause problems if you are relying on these as part of your application or automated maintenance procedures. Any application that relies upon an item that is been *deprecated* should be examined and scheduled to be rewritten in a timely manner.

Removed items

The following table shows the removed items:

Item	Type	Notes
mongoperf	Binary	Used to measure disk I/O performance without having to enter a mongoshell or otherwise access MongoDB.
$isolated	Operator	This operator was used in previous versions of MongoDB during update operations to prevent multidocument updates from being read until all changes took place. MongoDB 4.x uses transaction support instead. Any commands that include this operator need to be rewritten.

Significant removed items

The following table shows the removed items:

Item	Notes	Type
copyDb clone	Command	Copies an entire database. Although this command is still available, you cannot use it to copy a database managed by a mongod version 4 instance to one managed by a mongod version 3.4 or earlier instance. Use the external binaries mongodumpand mongorestoreinstead after upgrading to MongoDB 4.
db.copyDatabase()	Mongo shell command	This is a wrapper for the copyDbcommand. The same notes for copyDb cloneapply.
db.cloneDatabase()	Mongo shell command	This is a wrapper for the clonecommand. The same notes for copyDb cloneapply.
geoNear	Command	Reads geospatial information (that is, latitude and longitude) and returns documents in order, based on their proximity to the source point. Instead of this command, in MongoDB 4.x, you would use the $geoNear aggregation stage operator or the $nearor $nearSpherequery operators, depending on the nature of the query you wish to construct.

These commands were *deprecated* in MongoDB 4.0 and *removed* as of MongoDB 4.2.

Deprecated SSL configuration options

Another compatibility issue comes from the TLS/SSL configuration options. In MongoDB 3 and MongoDB 4.0, in the configuration files for both *mongod* (MongoDB database daemon) and *mongos* (which is used to control a sharded cluster), you could add a series of options under the net.ssl (https://docs.mongodb.com/v4.0/reference/configuration- options/#net-ssl-options) key. As of MongoDB 4.2, these options are deprecated in favor of the net.tlsoptions. The net.tls options have enhanced functionality compared with the net.ssloptions. These are covered in detail in Chapter11, *Administering MongoDB Security*.

Summary

In this chapter, you learned about the most important features that were added to MongoDB version 4. These were broken down into three categories: new features, security enhancements, and things to avoid during an upgrade from MongoDB 3 to 4.

One important new feature was adding timestamps to the WiredTiger storage engine, which opened the doors for multidocument transaction support and nonblocking secondary read enhancements. Other internal enhancements include improvements in the shard-migration process, which significantly cuts down the time required for this operation to complete.

In the realm of security, you learned about how SHA-256 support gives you greater security when communicating with the MongoDB database, and also with communications between servers within a replica set or sharded cluster. You also learned that TLS 1.0 support has been removed, and that the new default is TLS 1.1. MongoDB 4.x even provides support for the latest version of TLS, version 1.3, but only if the underlying operating system libraries provide support.

Finally, as the original MongoDB storage engine, MMAPv1, has been removed in favor of WiredTiger, if your original MongoDB 3 installation had data that used MMAPv1, you need to back up while still running MongoDB 3 and then restore after the MongoDB 4.x upgrade has occurred. You were also presented with a list of the most significant items, including binary executables, parameters, and commands, which have been removed, and those which have been deprecated.

In the next chapter, you learn how to install MongoDB 4.x and its Python programming language driver.