A MongoDB White Paper

MongoDB Operations Best Practices

MongoDB 3.0
February 2015

mongoDB

Table of Contents

Introduction

MongoDB is a high-performance, scalable database designed for a broad array of modern applications. It is used by organizations of all sizes to power online applications where low latency, high throughput and continuous availability are critical requirements of the system.

While some aspects of MongoDB are different from traditional relational databases, the concepts of the system, its operations, policies and procedures will be familiar to staff who have deployed and operated other database systems. Organizations have found that DBAs and operations teams have been able to integrate MongoDB into their production environments without needing to customize operational procedures.

This paper provides guidance on best practices for deploying and managing MongoDB. It assumes familiarity with the architecture of MongoDB and an understanding of concepts related to the deployment of enterprise software.

For the most detailed information on specific topics, please see the online documentation at mongodb.org. Many links are provided throughout this whitepaper to help guide users to the appropriate resources online.

Roles and Responsibilities

As with any database, applications deployed on MongoDB require careful planning and the coordination of a number of roles in an organization's technical teams to ensure successful maintenance and operation. Organizations tend to find that most of the same individuals and their respective roles for traditional database deployments are appropriate for a MongoDB deployment: Data Architects, Database Administrators, System Administrators, Application Developers, and Network Administrators.

In smaller organizations it is common for IT staff to fulfill multiple roles, whereas in larger companies it is more common for each role to be assumed by an individual or team dedicated to those tasks. For example, in a large investment bank there may be a very strong delineation between the functional responsibilities of a DBA and those of a system administrator.

Data Architect

While modeling data for MongoDB is typically simpler than modeling data for a relational database, there tend to be multiple options for a data model, and each has tradeoffs regarding performance, resource utilization, ease of use, and other areas. The data architect can carefully weigh these options with the development team to make informed decisions regarding the design of the schema. Typically the data architect performs tasks that are more proactive in nature, whereas the database administrator may perform tasks that are more reactive.

Database Administrator (DBA)

As with other database systems, many factors should be considered in designing a MongoDB system for a desired performance SLA. The DBA should be involved early in the project regarding discussions of the data model, the types of queries that will be issued to the system, the query volume, the availability goals, the recovery goals, and the desired performance characteristics.

System Administrator (Sysadmin)

Sysadmins typically perform a set of activities similar to those required in managing other applications, including upgrading software and hardware, managing storage, system monitoring, and data migration. MongoDB users have reported that their sysadmins have had no trouble learning to deploy, manage, and monitor MongoDB because no special skills are required.

Application Developer

The application developer works with other members of the project team to ensure the requirements regarding functionality, deployment, security, and availability are clearly understood. The application itself is written in a language such as Java, C#, PHP, or Ruby. Data will be stored, updated, and queried in MongoDB, and language-specific drivers are used to communicate between MongoDB and the application. The application developer works with the data architect to define and evolve the data model and to define the query patterns that should be optimized. The application developer works with the database administrator, sysadmin and network administrator to define the deployment and availability requirements of the application.

Network Administrator

A MongoDB deployment typically involves multiple servers distributed across multiple data centers. Network resources are a critical component of a MongoDB system. While MongoDB does not require any unusual configurations or resources as compared to other database systems, the network administrator should be consulted to ensure the appropriate policies, procedures, configurations, capacity, and security settings are implemented for the project.

Preparing for a MongoDB Deployment

MongoDB Pluggable Storage Engines

MongoDB 3.0 exposes a new storage engine API, enabling the integration of pluggable storage engines that extend MongoDB with new capabilities, and enable optimal use of specific hardware architectures. MongoDB 3.0 ships with two supported storage engines:

- The default MMAPv1 engine, an improved version of the engine used in prior MongoDB releases.
- The new WiredTiger storage engine. For many applications, WiredTiger's more granular concurrency control and native compression will provide significant benefits in the areas of lower storage costs, greater hardware utilization, and more predictable performance.

Both storage engines can coexist within a single MongoDB replica set, making it easy to evaluate and migrate between them. Upgrades to the WiredTiger storage engine are non-disruptive for existing replica set deployments; applications will be 100% compatible, and migrations can be performed with zero downtime through a rolling upgrade of the MongoDB replica set. WiredTiger is enabled by starting the server using the following option: mongod --storageEngine wiredTiger

Review the documentation for a checklist and full instructions on the migration process.

While each storage engine is optimized for different workloads users still leverage the same MongoDB query language, data model, scaling, security and operational tooling independent of the engine they use. As a result most of best practices in this guide apply to both supported storage engines. Any differences in recommendations between the two storage engines are noted.

Schema Design

Developers and data architects should work together to develop the right data model, and they should invest time in this exercise early in the project. The application should drive the data model, updates, and queries of your MongoDB system. Given MongoDB's dynamic schema, developers and data architects can continue to iterate on the data model throughout the development and deployment processes to optimize performance and storage efficiency, as well as support the addition of new application features. All of this can be done without expensive schema migrations.

The topic of schema design is significant, and a full discussion is beyond the scope of this guide. A number of resources are available online, including conference presentations from MongoDB Solutions Architects and users, as well as no-cost, web-based training provided by MongoDB University. MongoDB Global Consulting Services offers a dedicated 3-day Schema Design service.. The key schema design concepts to keep in mind are as follows.

Document Model

MongoDB stores data as documents in a binary representation called BSON. The BSON encoding extends the popular JSON representation to include additional types such as int, long, and floating point. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, sub-documents and binary data. It may be helpful to think of documents as roughly equivalent to rows in a relational database, and fields as roughly equivalent to columns. However, MongoDB documents tend to have all related data for a given record or object in a single document, whereas in a relational database that data is usually spread across rows in many tables. For example, data that belongs to a parent-child relationship in two RDBMS tables would commonly be collapsed (embedded) into a single document in MongoDB. As a result, the document model makes JOINs redundant in many cases.

Dynamic Schema

MongoDB documents can vary in structure. For example, documents that describe users might all contain the user id and the last date they logged into the system, but only some of these documents might contain the user's shipping address, and perhaps some of those contain multiple shipping addresses. MongoDB does not require that all documents conform to the same structure. Furthermore, there is no need to declare the structure of documents to the system – documents are self-describing. MongoDB does not enforce schemas. Schema enforcement should be performed by the application.

Collections

Collections are groupings of documents. Typically all documents in a collection have similar or related purposes for an application. It may be helpful to think of collections as being analogous to tables in a relational database.

Indexes

MongoDB uses B-tree indexes to optimize queries. Indexes are defined in a collection on document fields. MongoDB includes support for many indexes, including compound, geospatial, TTL, text search, sparse, unique, and others. For more information see the section on indexes.

Transactions

Atomicity of updates may influence the schema for your application. MongoDB guarantees ACID compliant updates to data at the document level. It is not possible to update multiple documents in a single atomic operation, however as with JOINs, the ability to embed related data into

MongoDB documents eliminates this requirement in many cases.

For more information on schema design, please see Data Modeling Considerations for MongoDB in the MongoDB Documentation.

Document Size

The maximum BSON document size in MongoDB is 16 MB. Users should avoid certain application patterns that would allow documents to grow unbounded. For example, in an e-commerce application it would be difficult to estimate how many reviews each product might receive from customers. Furthermore, it is typically the case that only a subset of reviews is displayed to a user, such as the most popular or the most recent reviews. Rather than modeling the product and customer reviews as a single document it would be better to model each review or groups of reviews as a separate document with a reference to the product document.

GridFS

For files larger than 16 MB, MongoDB provides a convention called GridFS, which is implemented by all MongoDB drivers. GridFS automatically divides large data into 256 KB pieces called chunks and maintains the metadata for all chunks. GridFS allows for retrieval of individual chunks as well as entire documents. For example, an application could quickly jump to a specific timestamp in a video. GridFS is frequently used to store large binary files such as images and videos in MongoDB.

Space Allocation Tuning (Relevant Only for MMAPv1 Storage Engine)

When a document is updated in the MongoDB MMAPv1 storage engine, the data is updated in-place if there is sufficient space. If the size of the document is greater than the allocated space, then the document may need to be re-written in a new location. The process of moving documents and updating their associated indexes can be I/O-intensive and can unnecessarily impact performance.

To anticipate future growth, the usePowerOf2Sizes attribute is enabled by default on each collection. This setting automatically configures MongoDB to round up allocation sizes to the powers of 2 (e.g., 2, 4, 8, 16, 32, 64, etc). This setting reduces the chances of increased disk I/O at the cost of using some additional storage.

An additional strategy is to manually pad the documents to provide sufficient space for document growth. If the application will add data to a document in a predictable fashion, the fields can be created in the document before the values are known in order to allocate the appropriate amount of space during document creation. Padding will minimize the relocation of documents and thereby minimize over-allocation, which can be viewed as the paddingFactor field in the output of the db..stats() command. For example, a value of 1 indicates no padding factor, and a value of 1.5 indicates a padding factor of 50%.

The considerations above are not relevant to the MongoDB WiredTiger storage engine which rewrites the document for each update.

Data Lifecycle Management

MongoDB provides features to facilitate the management of data lifecycles, including Time to Live, and capped collections. In addition, by using MongoDB's location-aware sharding, administrators can build highly efficient tiered storage models to support the data lifecycle. With location-aware sharding, administrators can balance query latency with storage density and cost by assigning data sets based on a value such as a timestamp to specific storage devices: Recent, frequently accessed data can be assigned to high performance SSDs with Snappy compression enabled. Older, less frequently accessed data is tagged to lower-throughput hard disk drives where it is compressed with zlib to attain maximum storage density with a lower cost-per-bit. As data ages, MongoDB automatically migrates it between storage tiers, without administrators having to build tools or ETL processes to manage data movement.

You can learn more about using location-aware sharding later in this guide.

Time to Live (TTL)

If documents in a collection should only persist for a pre-defined period of time, the TTL feature can be used to automatically delete documents of a certain age rather than scheduling a process to check the age of all documents and run a series of deletes. For example, if user sessions should only exist for one hour, the TTL can be set for 3600 seconds for a date field called lastActivity that exists in documents used to track user sessions and their last interaction with the system. A background thread will automatically check all these documents and delete those that have been idle for more than 3600 seconds. Another example for TTL is a price quote that should automatically expire after a period of time.

Capped Collections

In some cases a rolling window of data should be maintained in the system based on data size. Capped collections are fixed-size collections that support high-throughput inserts and reads based on insertion order. A capped collection behaves like a circular buffer: data is inserted into the collection, that insertion order is preserved, and when the total size reaches the threshold of the capped collection, the oldest documents are deleted to make room for the newest documents. For example, store log information from a high-volume system in a capped collection to quickly retrieve the most recent log entries.

Dropping a Collection

It is very efficient to drop a collection in MongoDB. If your data lifecycle management requires periodically deleting large volumes of documents, it may be best to model those documents as a single collection. Dropping a collection is much more efficient than removing all documents or a large subset of a collection, just as dropping a table is more efficient than deleting all the rows in a table in a relational database.

When WiredTiger is configured as the MongoDB storage engine, disk space is automatically reclaimed after a collection is dropped. Administrators need to run the compact command to reclaim space when using the MMAPv1 storage engine.

Indexing

Like most database management systems, indexes are a crucial mechanism for optimizing system performance in MongoDB. And while indexes will improve the performance of some operations by one or more orders of magnitude, they incur overhead to updates, disk space, and memory usage. Users should always create indexes to support queries, but should not maintain indexes that queries do not use. This is particularly important for deployments that support insert-heavy workloads.

Query Optimization

Queries are automatically optimized by MongoDB to make evaluation of the query as efficient as possible. Evaluation normally includes the selection of data based on predicates, and the sorting of data based on the sort criteria provided. The query optimizer selects the best index to use by periodically running alternate query plans and selecting the index with the lowest scan count for each query type. The results of this empirical test are stored as a cached query plan and periodically updated.

MongoDB provides an explain plan capability that shows information about how a query was resolved, including:

- The number of documents returned.
- Which index was used.
- Whether the query was covered, meaning no documents needed to be read to return results.
- Whether an in-memory sort was performed, which indicates an index would be beneficial.
- The number of index entries scanned.
- How long the query took to resolve in milliseconds.

The explain plan will show 0 milliseconds if the query was resolved in less than 1 ms, which is not uncommon in well-tuned systems. When explain plan is called, prior cached query plans are abandoned, and the process of testing multiple indexes is evaluated to ensure the best possible plan is used. The query plan can be calculated and returned without first having to run the query. This enables DBAs to review which plan will be used to execute the query, without having to wait for the query to run to completion.

If the application will always use indexes, MongoDB can be configured to throw an error if a query is issued that requires scanning the entire collection.

Profiling

MongoDB provides a profiling capability called Database Profiler, which logs fine-grained information about database operations. The profiler can be enabled to log information for all events or only those events whose duration exceeds a configurable threshold (whose default is 100 ms). Profiling data is stored in a capped collection where it can easily be searched for relevant events. It may be easier to query this collection than parsing the log files. MongoDB Ops Manager and the MongoDB Management Service (discussed later in the guide) can be used to visualize output from the profiler when identifying slow queries.

Primary and Secondary Indexes

A unique index is created for all documents by the _id field. MongoDB will automatically create the _id field and assign a unique value, or the value can be specified when the document is inserted. All user-defined indexes are secondary indexes. MongoDB includes support for many types of secondary indexes that can be declared on any field in the document, including fields within arrays. Index options include:

- Compound indexes
- Geospatial indexes
- Text search indexes
- Unique indexes
- Array indexes
- TTL indexes
- Sparse indexes
- Hash indexes

You can learn more about each of these indexes from the MongoDB Architecture Guide

Index Creation Options

Indexes and data are updated synchronously in MongoDB, thus ensuring queries on indexes never return stale or deleted data. The appropriate indexes should be determined as part of the schema design process. By default creating an index is a blocking operation in MongoDB. Because the creation of indexes can be time and resource intensive, MongoDB provides an option for creating new indexes as a background operation on both the primary and secondary members of a replica set. When the background option is enabled, the total time to create an index will be greater than if the index was created in the foreground, but it will still be possible to query the database while creating indexes. In addition, multiple indexes can be built concurrently in the background. Refer to the Build Index on Replica Sets documentation to learn more about considerations for index creation and on-going maintenance.

Managing Indexes with the MongoDB WiredTiger Storage Engine

Both storage engines fully support MongoDB's rich indexing functionality. If you have configured MongoDB to use the WiredTiger storage engine, then there are some additional optimizations that you can take advantage of:

- By default, WiredTiger uses prefix compression to reduce index footprint on both persistent storage and in RAM. This enables administrators to dedicate more of the working set to manage frequently accessed documents. Compression ratios of around 50% are typical, but users are encouraged to evaluate the actual ratio they can expect by testing their own workloads.
- Administrators can place indexes on their own separate volume, allowing for faster disk paging and lower contention.

Index Limitations

There are a few limitations to indexes that should be observed when deploying MongoDB:

- A collection cannot have more than 64 indexes.
- Index entries cannot exceed 1024 bytes.

- The name of an index must not exceed 125 characters (including its namespace).
- Indexes consume disk space and memory. Use them as necessary.
- Indexes can impact update performance. An update must first locate the data to change, so an index will help in this regard, but index maintenance itself has overhead and this work will reduce update performance.
- In-memory sorting of data without an index is limited to 32MB. This operation is very CPU intensive, and in-memory sorts indicate an index should be created to optimize these queries.

Common Mistakes Regarding Indexes

The following tips may help to avoid some common mistakes regarding indexes:

- **Use a compound index rather than index intersection for best performance:** Index intersection is useful for ad-hoc queries, but for best performance when querying via multiple predicates, compound indexes will generally be more performant.
- **Compound indexes:** Compound indexes are defined and ordered by field. So, if a compound index is defined for last name, first name and city, queries that specify last name or last name and first name will be able to use this index, but queries that try to search based on city will not be able to benefit from this index.
- **Low selectivity indexes:** An index should radically reduce the set of possible documents to select from. For example, an index on a field that indicates male/female is not as beneficial as an index on zip code, or even better, phone number.
- **Regular expressions:** Trailing wildcards work well, but leading wildcards do not because the indexes are ordered.
- **Negation:** Inequality queries are inefficient with respect to indexes.

Working Sets

MongoDB makes extensive use of RAM to speed up database operations. In MongoDB, all data is read and manipulated through in-memory representations of the data. The MMAPv1 storage engine uses memory-mapped files, whereas WiredTiger manages data through its cache. Reading data from memory is measured in nanoseconds and reading data from disk is measured in milliseconds; reading from memory is approximately 100,000 times faster than reading data from disk.

The set of data and indexes that are accessed during normal operations is called the working set. It is best practice that the working set fits in RAM. It may be the case the working set represents a fraction of the entire database, such as in applications where data related to recent events or popular products is accessed most commonly.

Page faults occur when MongoDB attempts to access data that has not been loaded in RAM. If there is free memory then the operating system can locate the page on disk and load it into memory directly. However, if there is no free memory, the operating system must write a page that is in memory to disk and then read the requested page into memory. This process can be time consuming and will be significantly slower than accessing data that is already in memory.

Some operations may inadvertently purge a large percentage of the working set from memory, which adversely affects performance. For example, a query that scans all documents in the database, where the database is larger than the RAM on the server, will cause documents to be read into memory and the working set to be written out to disk. Other examples include some maintenance operations such as compacting or repairing a database and rebuilding indexes.

If your database working set size exceeds the available RAM of your system, consider increasing the RAM or adding additional servers to the cluster and sharding your database. For a discussion on this topic, see the section on Sharding Best Practices. It is far easier to implement sharding before the resources of the system become limited, so capacity planning is an important element in successful project delivery.

A useful output included with the serverStatus command is a workingSet document that provides an estimated size of the MongoDB instance's working set. Operations teams can track the number of pages accessed by the instance over a given period, and the elapsed time from the oldest to newest document in the working set. By tracking these metrics, it is possible to detect when the working set is approaching current RAM limits and proactively take action to ensure the system is scaled.

MongoDB Setup and Configuration

Setup

MongoDB provides repositories for .deb and .rpm packages for consistent setup, upgrade, system integration, and configuration. This software uses the same binaries as the tarball packages provided from the MongoDB Downloads Page. The MongoDB Windows package is available via the downloadable binary installed via its MSI.

Database Configuration

User should store configuration options in mongod's configuration file. This allows sysadmins to implement consistent configurations across entire clusters. The configuration files support all options provided as command line options for mongod. Popular tools such as Chef and Puppet can be used to provision MongoDB instances. The provisioning of complex topologies comprising replica sets and sharded clusters can be automated by the MongoDB Management Service (MMS) and Ops Manager, which are discussed later in this guide.

Upgrades

Users should upgrade software as often as possible so that they can take advantage of the latest features as well as any stability updates or bug fixes. Upgrades should be tested in non-production environments to ensure live applications are not adversely affected by new versions of the software.

Customers can deploy rolling upgrades without incurring any downtime, as each member of a replica set can be upgraded individually without impacting database availability. It is possible for each member of a replica set to run under different versions of MongoDB, and with different storage engines. As a precaution, the release notes for the MongoDB release should be consulted to determine if there is a particular order of upgrade steps that needs to be followed and whether there are any incompatibilities between two specific versions. Upgrades can be automated with MMS and Ops Manager.

Data Migration

Users should assess how best to model their data for their applications rather than simply importing the flat file exports of their legacy systems. In a traditional relational database environment, data tends to be moved between systems using delimited flat files such as CSV. While it is possible to ingest data into MongoDB from CSV files, this may in fact only be the first step in a data migration process. It is typically the case that MongoDB's document data model provides advantages and alternatives that do not exist in a relational data model.

The mongoimport and mongoexport tools are provided with MongoDB for simple loading or exporting of data in JSON or CSV format. These tools may be useful in moving data between systems as an initial step. Other tools such as mongodump and mongorestore and MMS or Ops Manager are useful for moving data between two MongoDB systems.

There are many options to migrate data from flat files into rich JSON documents, including mongoimport, custom scripts, ETL tools and from within an application itself which can read from the existing RDBMS and then write a JSON version of the document back to MongoDB.

Hardware

The following recommendations are only intended to provide high-level guidance for hardware for a MongoDB deployment. The specific configuration of your hardware will be dependent on your data, your queries, your performance SLA, your availability requirements, and the capabilities of the underlying hardware components. MongoDB has extensive experience helping customers to select hardware and tune their configurations and we frequently work with customers to plan for and optimize their MongoDB systems. The Healthcheck and Production Readiness consulting packages can be especially valuable in helping select the appropriate hardware for your project.

MongoDB was specifically designed with commodity hardware in mind and has few hardware requirements or limitations. Generally speaking, MongoDB will take advantage of more RAM and faster CPU clock speeds.

Memory

MongoDB makes extensive use of RAM to increase performance. Ideally, the working set fits in RAM. As a general rule of thumb, the more RAM, the better. As workloads begin to access data that is not in RAM, the performance of MongoDB will degrade, as it will for any database. MongoDB delegates the management of RAM to the operating system. MongoDB will use as much RAM as possible until it exhausts what is available. The WiredTiger storage engine gives more control of memory by allowing users to configure how much RAM to allocate to the WiredTiger cache – defaulting to 50% of available memory. WiredTiger's filesystem cache will grow to utilize the remaining memory available.

Storage

MongoDB does not require shared storage (e.g., storage area networks). MongoDB can use local attached storage as well as solid state drives (SSDs). Most disk access patterns in MongoDB do not have sequential properties, and as a result, customers may experience substantial performance gains by using SSDs. Good results and strong price to performance have been observed with SATA SSD and with PCI. Commodity SATA spinning drives are comparable to higher cost spinning drives due to the non-sequential access patterns of MongoDB: rather than spending more on expensive spinning drives, that money may be more effectively spent on more RAM or SSDs. Another benefit of using SSDs is that they provide a more gradual degradation of performance if the working set no longer fits in memory.

While data files benefit from SSDs, MongoDB's journal files are good candidates for fast, conventional disks due to their high sequential write profile. See the section on journaling later in this guide for more information.

Most MongoDB deployments should use RAID-10. RAID-5 and RAID-6 do not provide sufficient performance. RAID-0 provides good write performance, but limited read performance and insufficient fault tolerance. MongoDB's replica sets allow deployments to provide stronger availability for data, and should be considered with RAID and other factors to meet the desired availability SLA.

Compression

MongoDB natively supports compression when using the WiredTiger storage engine. Compression reduces storage footprint by as much as 80%, and enables higher storage I/O scalability as fewer bits are read from disk. As with any compression algorithm administrators trade storage efficiency for CPU overhead, and so it is important to test the impacts of compression in your own environment.

MongoDB offers administrators a range of compression options for documents, indexes and the journal. The default snappy compression algorithm provides a good balance between high document and journal compression ratio (typically around 70%, dependent on the data) with low CPU overhead, while the optional zlib library will achieve higher compression, but incur additional CPU cycles as data is written to and read from disk. Indexes use prefix compression by default, which serves to reduce the in-memory footprint of index storage, freeing up more of the working set for frequently accessed documents. Administrators can modify the default compression settings for all collections and indexes. Compression is also configurable on a per-collection and per-index basis during collection and index creation.

CPU

MongoDB will deliver better performance on faster CPUs. The MongoDB WiredTiger storage engine is better able to saturate multi-core processor resources than the MMAPv1 storage engine.

Process Per Host

For best performance, users should run one mongod process per host. With appropriate sizing and resource allocation using virtualization or container technologies, multiple MongoDB processes can run on a single server without contending for resources. If using the WiredTiger storage engine, administrators will need to calculate the appropriate cache size for each instance by evaluating what portion of total RAM each of them should use, and splitting the default cache_size between each.

For availability, multiple members of the same replica set should not be co-located on the same physical hardware.

Virtualization and IaaS

Customers can deploy MongoDB on bare metal servers, in virtualized environments and in the cloud. Performance will typically be best and most consistent using bare metal, though many MongoDB users leverage infrastructure-as-a-service (IaaS) products like Amazon Web Services' Elastic Compute Cloud (AWS EC2), Rackspace, Google Compute Engine, Microsoft Azure, and others.

Sizing for Mongos and Config Server Processes

For sharded systems, additional processes must be deployed alongside the mongod data storing processes: mongos query routers and config servers. Shards are physical partitions of data spread across multiple servers. For more on sharding, please see the section on horizontal scaling with shards. Queries are routed to the appropriate shards using a query router process called mongos. The metadata used by mongos to determine where to route a query is maintained by the config servers. Both mongos and config server processes are lightweight, but each has somewhat different sizing requirements.

Within a shard, MongoDB further partitions documents into chunks. MongoDB maintains metadata about the relationship of chunks to shards in the config server. Three config servers are maintained in sharded deployments to ensure availability of the metadata at all times. To estimate the total size of the shard metadata, multiply the size of the chunk metadata by the total number of chunks in your database – the default chunk size is 64 MB. For example, a 64 TB database would have 1 million chunks and the total size of the shard metadata managed by the config servers would be 1 million times the size of the chunk metadata, which could range from hundreds of MB to several GB of metadata. Shard metadata access is infrequent: each mongos maintains a cache of this data, which is periodically updated by background processes when chunks are split or migrated to other shards, typically during balancing operations as the cluster expands and contracts. The hardware for a config server should therefore be focused on availability: redundant power supplies, redundant network interfaces, redundant RAID controllers, and redundant storage should be used.

Typically multiple mongos instances are used in a sharded MongoDB system. It is not uncommon for MongoDB users to deploy a mongos instance on each of their application servers. The optimal number of mongos servers will be determined by the specific workload of the application: in some cases mongos simply routes queries to the appropriate shards, and in other cases mongos performs aggregation and other tasks. To estimate the memory requirements for each mongos, consider the following:

- The total size of the shard metadata that is cached by mongos
- 1MB for each connection to applications and to each mongos

The mongos process uses limited RAM and will benefit more from fast CPUs and networks.

Operating System and File System

Configurations for Linux

Only 64-bit versions of operating systems are supported for use with MongoDB. 32-bit builds are available for MongoDB with the MMAPv1 storage engine, but are provided only for backwards compatibility with older development environments. MongoDB WiredTiger builds are not available for 32-bit platforms.

Version 2.6.36 of the Linux kernel or later should be used for MongoDB in production. As MongoDB typically uses very large files, the Ext4 and XFS file systems are recommended:

- If you use the Ext4 file system, use at least version 2.6.23 of the Linux Kernel.
- If you use the XFS file system, use at least version 2.6.25 of the Linux Kernel.

- For MongoDB on Linux use the following recommended configurations:
- Turn off atime for the storage volume with the database files.
- Do not use hugepages virtual memory pages, MongoDB performs better with normal virtual memory pages.
- Disable NUMA in your BIOS or invoke mongod with NUMA disabled.
- Ensure that readahead settings for the block devices that store the database files are relatively small as most access is non-sequential. For example, setting readahead to 32 (16 KB) is a good starting point.
- Synchronize time between your hosts. This is especially important in sharded MongoDB clusters.

Linux provides controls to limit the number of resources and open files on a per-process and per-user basis. The default settings may be insufficient for MongoDB. Generally MongoDB should be the only process on a system to ensure there is no contention with other processes.

While each deployment has unique requirements, the following settings are a good starting point mongod and mongos instances. Use ulimit to apply these settings:

- -f (file size): unlimited
- -t (cpu time): unlimited
- -v (virtual memory): unlimited
- -n (open files): above 20,000
- -m (memory size): unlimited
- -u (processes/threads): above 20,000

For more on using ulimit to set the resource limits for MongoDB, see the MongoDB Documentation page on (Linux ulimit Settings)[http://docs.mongodb.org/manual/reference/ulimit/].

Networking

Always run MongoDB in a trusted environment with network rules that prevent access from all unknown entities. There are a finite number of predefined processes that communicate with a MongoDB system: application servers, monitoring processes, and MongoDB processes.

By default MongoDB processes will bind to all available network interfaces on a system. If your system has more than one network interface, bind MongoDB processes to the private or internal network interface.

Detailed information on default port numbers for MongoDB, configuring firewalls for MongoDB, VPN, and other topics is available in the MongoDB Security Tutorials. Review the Security section later in this guide for more information on best practices on securing your deployment.

Production-Proven Recommendations

The latest recommendations on specific configurations for operating systems, file systems, storage devices and other system-related topics are maintained in the MongoDB Production Notes documentation.

Continuous Availability

Under normal operating conditions, a MongoDB system will perform according to the performance and functional goals of the system. However, from time to time certain inevitable failures or unintended actions can affect a system in adverse ways. Hard drives, network cards, power supplies, and other hardware components will fail. These risks can be mitigated with redundant hardware components. Similarly, a MongoDB system provides configurable redundancy throughout its software components as well as configurable data redundancy.

Journaling

MongoDB implements write-ahead journaling of operations to enable fast crash recovery and durability in the storage engine. In the case of a server crash, journal entries are recovered automatically.

The behavior of the journal is dependent on the configured storage engine:

- MMAPv1 journal commits to disk are issued at least as often as every 100 ms by default. In addition to providing durability, the journal also prevents corruption in the case of an unclean shutdown of the system. By default, journaling is enabled for MongoDB with MMAPv1. No production deployment should run without the journal configured.

- The WiredTiger journal ensures that writes are persisted to disk between checkpoints. WiredTiger uses checkpoints to flush data to disk by default every 60 seconds or after 2GB of data has been written. Thus, by default, WiredTiger can lose up to 60 seconds of writes if running without journaling – though the risk of this loss will typically be much less if using replication for durability. The WiredTiger transaction log is not necessary to keep the data files in a consistent state in the event of an unclean shutdown, and so it is safe to run without journaling enabled, though to ensure durability the "replica safe" write concern should be configured (see the Write Availability section later in the guide for more information). Another feature of the WiredTiger storage engine is the ability to compress the journal on disk, thereby reducing storage space.

For additional guarantees, the administrator can configure the journaled write concern for both storage engines, whereby MongoDB acknowledges the write operation only after committing the data to the journal.

Locating MongoDB's journal files and data files on separate storage arrays may help performance. The I/O patterns for the journal are very sequential in nature and are well suited for storage devices that are optimized for fast sequential writes, whereas the data files are well suited for storage devices that are optimized for random reads and writes. Simply placing the journal files on a separate storage device normally provides some performance enhancements by reducing disk contention.

Learn more about journaling from the documentation.

Data Redundancy

MongoDB maintains multiple copies of data, called replica sets, using native replication. Users should use replica sets to help prevent database downtime. Replica failover is fully automated in MongoDB, so it is not necessary to manually intervene to recover in the event of a failure.

A replica set consists of multiple replicas. At any given time, one member acts as the primary replica and the other members act as secondary replicas. If the primary member fails for any reason (e.g., a failure of the host system), one of the secondary members is automatically elected to primary and begins to process all writes.

Sophisticated algorithms control the election process, ensuring only the most suitable secondary member is promoted to primary, and reducing the risk of unnecessary failovers (also known as "false positives"). The election algorithms process a range of parameters including analysis of timestamps to identify those replica set members that have applied the most recent updates from the primary, heartbeat and connectivity status and user-defined priorities assigned to replica set members. For example, administrators can configure all replicas located in a secondary data center to be candidates for election only if the primary data center fails. Once the new primary replica set member has been elected, remaining secondary members are automatically reconfigured to receive updates from the new primary. If the original primary comes back online, it will recognize that it is no longer the primary and by default will reconfigure itself to become a secondary replica set member.

The number of replica nodes in a MongoDB replica set is configurable, and a larger number of replica nodes provides increased protection against database downtime in case of multiple machine failures. While a node is down MongoDB will continue to function. When a node is down, MongoDB has less resiliency and the DBA or sysadmin should work to recover the failed replica in order to mitigate the temporarily reduced resilience of the system.

Replica sets also provide operational flexibility by providing sysadmins with an option for performing hardware and software maintenance without taking down the entire system. Using a rolling upgrade, secondary members of the replica set can be upgraded in turn, before the administrator demotes the master to complete the upgrade. This process is fully automated when using MMS or Ops Manager discussed later in this guide.

Consider the following factors when developing the architecture for your replica set:

- Ensure that the members of the replica set will always be able to elect a primary. Run an odd number of members or run an arbiter (a replica that exists solely for participating in election of the primary) on one of your application servers if you have an even number of members. There should be at least three replicas with copies of the data in a replica set, or two replicas with an arbiter.
- With geographically distributed members, know where the majority of members will be in the case of any network partitions. Attempt to ensure that the set can elect a primary among the members in the primary data center.
- Consider including a hidden member in the replica set. Hidden members can never become a primary and are typically used for backups or to run applications such as analytics and reporting that require isolation from regular operational workloads. Delayed replica set members can also be deployed that apply changes on a fixed time delay to provide recovery from unintentional operations.

More information on replica sets can be found on the Replication MongoDB documentation page.

Multi-Data Center Replication

MongoDB replica sets allow for flexible deployment designs both within and across data centers that account for failure at the server, rack, and regional levels. In the case of a natural or human-induced disaster, the failure of a single datacenter can be accommodated with no downtime when MongoDB replica sets are deployed across datacenters.

Availability of Writes

MongoDB allows administrators to specify the level of availability when issuing writes to the database, which is called the write concern. The following options can be configured on a per connection, per database, per collection, or even per operation basis. Starting with the lowest level of guarantees, the options are as follows:

- **Write Acknowledged:** This is the default global write concern. The mongod will confirm the receipt of the write operation, allowing the client to catch network, duplicate key, and other exceptions.
- **Replica Safe:** It is also possible to wait for acknowledgement of writes to other replica set members. MongoDB supports writing to a specific number of replicas, or to a majority of replica set members. Because replicas can be deployed across racks within data centers and across multiple data centers, ensuring writes propagate to additional replicas can provide extremely robust durability.
- **Journal Safe (journaled):** The mongod will confirm the write operation only after it has flushed the operation to the journal on the primary. This confirms that the write operation can survive a mongod crash and ensures that the write operation is durable on disk.
- **Data Center Awareness:** Using tag sets, sophisticated policies can be created to ensure data is written to specific combinations of replica sets prior to acknowledgement of success. For example, you can create a policy that requires writes to be written to at least three data centers on two continents, or two servers across two racks in a specific data center. For more information see the MongoDB Documentation on Data Center Awareness.

For more on the subject of configurable availability of writes see the MongoDB Documentation on Write Concern for Replica Sets.

Read Preferences

Reading from the primary replica is the default configuration. If higher read throughput is required, it is recommended to take advantage of MongoDB's auto-sharding to distribute read operations across multiple primary members.

There are applications where replica sets can improve scalability of the MongoDB deployment. For example, Business Intelligence (BI) applications can execute queries against a secondary replica, thereby reducing overhead on the primary and enabling MongoDB to serve operational and analytical workloads from a single deployment. Backups can be taken against the secondary replica to further reduce overhead. Another configuration option directs reads to the replica closest to the user based on ping distance, which can significantly decrease the latency of read operations in globally distributed applications.

A very useful option is primaryPreferred, which issues reads to a secondary replica only if the primary is unavailable. This configuration allows for the continuous availability of reads during the failover process.

For more on the subject of configurable reads, see the MongoDB Documentation page on replica set Read Preference.

Scaling a MongoDB System

Horizontal Scaling with Sharding

MongoDB provides horizontal scale-out for databases using a technique called sharding, which is transparent to applications. MongoDB distributes data across multiple physical partitions called shards. With automatic balancing, MongoDB ensures data is equally distributed across shards as data volumes grow or the size of the cluster increases or decreases. Sharding allows MongoDB deployments to scale beyond the limitations of a single server, such as bottlenecks in RAM or disk I/O, without adding complexity to the application.

MongoDB supports three types of sharding which enables administrators to accommodate diverse query patterns:

- **Range-based sharding:** Documents are partitioned across shards according to the shard key value. Documents with shard key values close to one another are likely to be co-located on the same shard. This approach is well suited for applications that need to optimize range-based queries.

- **Hash-based sharding:** Documents are uniformly distributed according to an MD5 hash of the shard key value. Documents with shard key values close to one another are unlikely to be co-located on the same shard. This approach guarantees a uniform distribution of writes across shards, making it optimal for write-intensive workloads.

- **Location-aware sharding:** Documents are partitioned according to a user-specified configuration that “tags” shard key ranges to physical shards residing on specific hardware. Users can optimize the physical location of documents for application requirements such as locating data in specific data centers, or for separating hot and cold data onto different tiers of storage.

While sharding is very powerful, it can add operational complexity to a MongoDB deployment and it has its own infrastructure requirements. As a result, users should shard as necessary and when indicated by actual operational requirements.

Users should consider deploying a sharded cluster in the following situations:

- **RAM Limitation:** The size of the system's active working set plus indexes is expected to exceed the capacity of the maximum amount of RAM in the system.

- **Disk I/O Limitation:** The system will have a large amount of write activity, and the operating system will not be able to write data fast enough to meet demand, or I/O bandwidth will limit how fast the writes can be flushed to disk.

- **Storage Limitation:** The data set will grow to exceed the storage capacity of a single node in the system.

- **Location-aware requirements:** The data set needs to be assigned to a specific data center for compliance, or to support low latency local reads and writes. Alternatively, to create multi-temperature storage infrastructures that separate hot and cold data onto specific volumes. You can learn more about using location-aware sharding for this deployment model by reading the Tiered Storage Models in MongoDB post.

Applications that meet these criteria, or that are likely to do so in the future, should be designed for sharding in advance rather than waiting until they run out of capacity. Applications that will eventually benefit from sharding should consider which collections they will want to shard and the corresponding shard keys when designing their data models. If a system has already reached or exceeded its capacity, it will be challenging to deploy sharding without impacting the application's performance.

Sharding Best Practices

Users who choose to shard should consider the following best practices:

Select a good shard key. When selecting fields to use as a shard key, there are at least three key criteria to consider:

1. Cardinality: Data partitioning is managed in 64MB chunks by default. Low cardinality (e.g., the attribute size) will tend to group documents together on a small number of shards, which in turn will require frequent rebalancing of the chunks. Instead, a shard key should exhibit high cardinality.
2. Insert Scaling: Writes should be evenly distributed across all shards based on the shard key. If the shard key is monotonically increasing, for example, all inserts will go to the same shard even if they exhibit high cardinality, thereby creating an insert hotspot. Instead, the key should be evenly distributed.
3. Query Isolation: Queries should be targeted to a specific shard to maximize scalability. If queries cannot be isolated to a specific shard, all shards will be queried in a pattern called scatter/gather, which is less efficient than querying a single shard.

For more on selecting a shard key, see Considerations for Selecting Shard Keys.

Add capacity before it is needed. Cluster maintenance is lower risk and more simple to manage if capacity is added before the system is over utilized. Run three configuration servers to provide redundancy. Production deployments must use three config servers. Config servers should be deployed in a topology that is robust and resilient to a variety of failures.

Use replica sets. Sharding and replica sets are absolutely compatible. Replica sets should be used in all deployments, and sharding should be used when appropriate. Sharding allows a database to make use of multiple servers for data capacity and system throughput. Replica sets maintain redundant copies of the data across servers, server racks, and even data centers.

Use multiple mongos instances.

Apply best practices for bulk inserts. Pre-split data into multiple chunks so that no balancing is required during the insert process. Alternately, disable the balancer. Also, use multiple mongos instances to load in parallel for greater throughput. For more information see Create Chunks in a Sharded Cluster in the MongoDB Documentation.

More information on sharding can be found in the MongoDB Documentation under Sharding Concepts

Dynamic Data Balancing

As data is loaded into MongoDB, the system may need to dynamically rebalance chunks across shards in the cluster using a process called the balancer. The balancing operations attempt to minimize the impact to the performance of the cluster by only moving one chunk of documents at a time, and by only migrating chunks when a distribution threshold is exceeded. It is possible to disable the balancer or to configure when balancing is performed to further minimize the impact on performance. For more information on the balancer and scheduling the balancing process, see the MongoDB Documentation page on Sharded Collection Balancing.

Geographic Distribution

Shards can be configured such that specific ranges of shard key values are mapped to a physical shard location. Location-aware sharding allows a MongoDB administrator to control the physical location of documents in a MongoDB cluster, even when the deployment spans multiple data centers in different regions.

It is possible to combine the features of replica sets, location-aware sharding, read preferences and write concern in order to provide a deployment that is geographically distributed, enabling users to read and write to their local data centers. It can also fulfil regulatory requirements around data locality. One can restrict sharded collections to a select set of shards, effectively federating those shards for different uses. For example, one can tag all USA data and assign it to shards located in the United States.

To learn more, download the MongoDB Multi-Datacenter Deployments Guide.

Managing MongoDB: Provisioning, Monitoring and Disaster Recovery

Ops Manager is the simplest way to run MongoDB, making it easy for operations teams to deploy, monitor, backup, and scale MongoDB. Ops Manager was created by the engineers who develop the database and is available as part of MongoDB Enterprise Advanced. Many of the capabilities of Ops Manager are also available in MMS hosted in the cloud. Today, MMS supports thousands of deployments, including systems from one to hundreds of servers.

Ops Manager and MMS incorporate best practices to help keep managed databases healthy and optimized. They ensures operational continuity by converting complex manual tasks into reliable, automated procedures with the click of a button or via an API call.

- **Deploy.** Any topology, at any scale;
- **Upgrade.** In minutes, with no downtime;
- **Scale.** Add capacity, without taking the application offline;
- **Point-in-time, Scheduled Backups.** Restore to any point in time, because disasters aren't predictable;
- **Performance Alerts.** Monitor 100+ system metrics and get custom alerts before the system degrades.

The Ops Optimization Service assists you in every stage of planning and implementing your operations strategy for MongoDB, including the production of a MongoDB playbook for your deployment. MMS is available for those operations teams who do not want to maintain their own management and backup infrastructure in-house.

Deployments and Upgrades

Ops Manager (and MMS) coordinate critical operational tasks across the servers in a MongoDB system. It communicates with the infrastructure through agents installed on each server. The servers can reside in the public cloud or a private data center. Ops Manager reliably orchestrates the tasks that administrators have traditionally performed manually – deploying a new cluster, upgrades, creating point in time backups, and many other operational tasks.

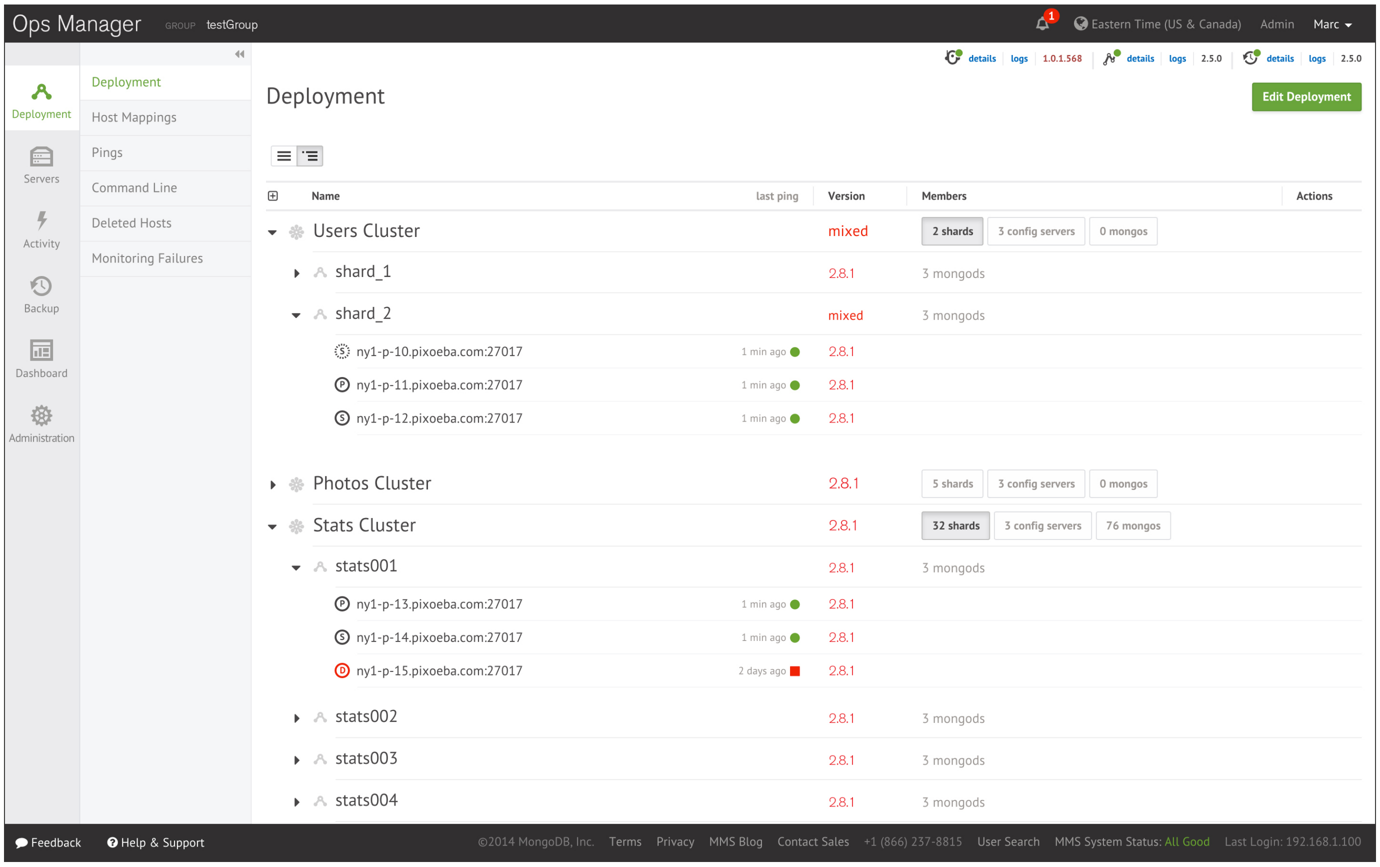
Ops Manager is designed to adapt to problems as they arise by continuously assessing state and making adjustments as needed. Here's how:

- Ops Manager agents are installed on servers (where MongoDB will be deployed), either through provisioning tools such as Chef or Puppet, or by an administrator.
- The administrator creates a new design goal for the system, either as a modification to an existing deployment (e.g., upgrade, oplog resize, new shard), or as a new system.
- The agents periodically check in with the Ops Manager central server and receive the new design instructions.
- Agents create and follow a plan for implementing the design. Using a sophisticated rules engine, agents continuously adjust their individual plans as conditions change. In the face of many failure scenarios – such as server failures and network partitions – agents will revise their plans to reach a safe state.
- Minutes later, the system is deployed, safely and reliably.

Ops Manager and MMS can deploy MongoDB on any connected server, but on AWS, MMS does even more. Users can input their AWS keys into MMS, which allows MMS to provision virtual machines on Amazon AWS and deploy MongoDB on them at the same time. This integration removes a step and makes it even easier to get started. MMS provisions your AWS virtual machines with an optimal configuration for MongoDB.

In addition to initial deployment, Ops Manager and MMS make it possible to dynamically resize capacity by adding shards and replica set members. Other maintenance tasks such as upgrading MongoDB or resizing the oplog can be reduced from dozens or hundreds of manual steps to the click of a button, all with zero downtime.

Administrators can use the Ops Manager interface directly, or invoke the Ops Manager RESTful API from existing enterprise tools, including popular monitoring and orchestration frameworks.



Name	last ping	Version	Members	Actions
Users Cluster		mixed	2 shards, 3 config servers, 0 mongos	
shard_1		2.8.1	3 mongods	
shard_2		mixed	3 mongods	
ny1-p-10.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-11.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-12.pixoeba.com:27017	1 min ago ●	2.8.1		
Photos Cluster		2.8.1	5 shards, 3 config servers, 0 mongos	
Stats Cluster		2.8.1	32 shards, 3 config servers, 76 mongos	
stats001		2.8.1	3 mongods	
ny1-p-13.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-14.pixoeba.com:27017	1 min ago ●	2.8.1		
ny1-p-15.pixoeba.com:27017	2 days ago 🟥	2.8.1		
stats002		2.8.1	3 mongods	
stats003		2.8.1	3 mongods	
stats004		2.8.1	3 mongods	

Figure 1: Ops Manager: simple, intuitive and powerful. Deploy and upgrade entire clusters with a single click.

Monitoring & Capacity Planning

System performance and capacity planning are two important topics that should be addressed as part of any MongoDB deployment. Part of your planning should involve establishing baselines on data volume, system load, performance, and system capacity utilization. These baselines should reflect the workloads you expect the system to perform in production, and they should be revisited periodically as the number of users, application features, performance SLA, or other factors change.

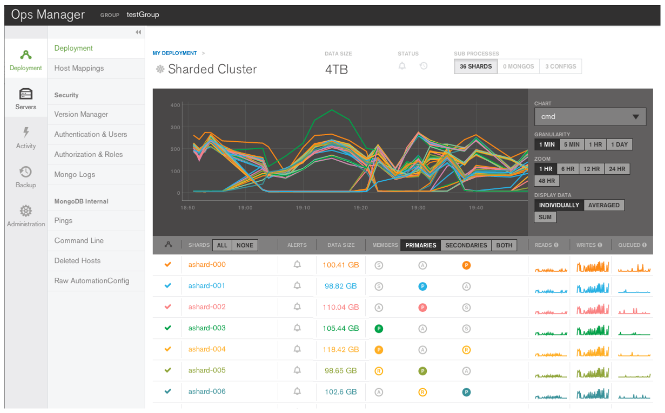
Baselines will help you understand when the system is operating as designed, and when issues begin to emerge that may affect the quality of the user experience or other factors critical to the system. It is important to monitor your MongoDB system for unusual behavior so that actions can be taken to address issues pro-actively. The following represents the most popular tools for monitoring MongoDB, and also describes different aspects of the system that should be monitored.

Monitoring with Ops Manager and MMS

Featuring charts, custom dashboards, and automated alerting, Ops Manager tracks 100+ key database and systems health metrics including operations counters, memory and CPU utilization, replication status, open connections, queues and any node status.

The metrics are securely reported to Ops Manager and MMS where they are processed, aggregated, alerted and visualized in a browser, letting administrators easily determine the health of MongoDB in real-time. Views can be based on explicit permissions, so project team visibility can be restricted to their own applications, while systems administrators can monitor all the MongoDB deployments in the organization.

Historic performance can be reviewed in order to create operational baselines and to support capacity planning. Integration with existing monitoring tools is also straightforward via the Ops Manager RESTful API, making the deep insights from Ops Manager part of a consolidated view across your operations.



SHARDS	DATA SIZE	MEMBERS		
✓ ashard-000	100.41 GB	S	A	P
✓ ashard-001	98.82 GB	S	P	A
✓ ashard-002	110.04 GB	A	P	S
✓ ashard-003	105.44 GB	P	A	S
✓ ashard-004	118.42 GB	P	A	R
✓ ashard-005	98.65 GB	R	P	A
✓ ashard-006	102.6 GB	A	R	P

Figure 2: Ops Manager provides real time & historic visibility into the MongoDB deployment.

Ops Manager and MMS allow administrators to set custom alerts when key metrics are out of range. Alerts can be configured for a range of parameters affecting individual hosts, replica sets, agents and backup. Alerts can be sent via SMS and email or integrated into existing incident management systems such as PagerDuty and HipChat to proactively warn of potential issues, before they escalate to costly outages.

If using MMS, access to monitoring data can also be shared with MongoDB support engineers, providing fast issue resolution by eliminating the need to ship logs between different teams.

Hardware Monitoring

Munin node is an open-source software program that monitors hardware and reports on metrics like disk and RAM usage. Ops Manager and MMS can collect this data from Munin node and provide it along with other data available in the Ops Manager dashboard. While each application and deployment is unique, users should create alerts for spikes in disk utilization, major changes in network activity, and increases in average query length/ response times.

mongotop

mongotop is a utility that ships with MongoDB. It tracks and reports the current read and write activity of a MongoDB cluster. mongotop provides collection-level stats.

mongostat

mongostat is a utility that ships with MongoDB. It shows real-time statistics about all servers in your MongoDB system. mongostat provides a comprehensive overview of all operations, including counts of updates, inserts, page faults, index misses, and many other important measures of the system health. mongostat is similar to the linux tool vmstat.

Other Popular Tools

There are a number of popular open-source monitoring tools for which MongoDB plugins are available. If MongoDB is configured with the WiredTiger storage engine, ensure the tool is using a WiredTiger-compatible driver:

- Nagios
- Ganglia
- Cacti
- Scout
- Munin
- Zabbix

Linux Utilities

Other common utilities that should be used to monitor different aspects of a MongoDB system:

- iostat: Provides usage statistics for the storage subsystem.
- vmstat: Provides usage statistics for virtual memory.
- netstat: Provide usage statistics for the network.
- sar: Captures a variety of system statistics periodically and stores them for analysis.

Windows Utilities

Performance Monitor, a Microsoft Management Console snap-in, is a useful tool for measuring a variety of stats in a Windows environment.

Things to Monitor

Ops Manager and MMS can be used to monitor database-specific metrics, including page faults, ops counters, queues, connections and replica set status. Alerts can be configured against each monitored metric to proactively warn administrators of potential issues before users experience a problem.

Application Logs And Database Logs

Application and database logs should be monitored for errors and other system information. It is important to correlate your application and database logs in order to determine whether activity in the application is ultimately responsible for other issues in the system. For example, a spike in user writes may increase the volume of writes to MongoDB, which in turn may overwhelm the underlying storage system. Without the correlation of application and database logs, it might take more time than necessary to establish that the application is responsible for the increase in writes rather than some process running in MongoDB.

In the event of errors, exceptions or unexpected behavior, the logs should be saved and uploaded to MongoDB when opening a support case. Logs for mongod processes running on primary and secondary replica set members, as well as mongos and config processes will enable the support team to more quickly root cause any issues.

Page Faults

When a working set ceases to fit in memory, or other operations have moved other data into memory, the volume of page faults may spike in your MongoDB system. Page faults are part of the normal operation of a MongoDB system, but the volume of page faults should be monitored in order to determine if the working set is growing to the level that it no longer fits in memory and if alternatives such as more memory or sharding across multiple servers is appropriate. In most cases, the underlying issue for problems in a MongoDB system tends to be page faults. Also use the working set estimator discussed earlier in the guide.

Disk

Beyond memory, disk I/O is also a key performance consideration for a MongoDB system because writes are journaled and regularly flushed to disk. Under heavy write load the underlying disk subsystem may become overwhelmed, or other processes could be contending with MongoDB, or the RAID configuration may be inadequate for the volume of writes. Other potential issues could be the root cause, but the symptom is typically visible through iostat as showing high disk utilization and high queuing for writes.

CPU

A variety of issues could trigger high CPU utilization. This may be normal under most circumstances, but if high CPU utilization is observed without other issues such as disk saturation or pagefaults, there may be an unusual issue in the system. For example, a MapReduce job with an infinite loop, or a query that sorts and filters a large number of documents from working set without good index coverage, might cause a spike in CPU without triggering issues in the disk system or pagefaults.

Connections

MongoDB drivers implement connection pooling to facilitate efficient use of resources. Each connection consumes 1MB of RAM, so be careful to monitor the total number of connections so they do not overwhelm the available RAM and reduce the available memory for the working set. This typically happens when client applications do not properly close their connections, or with Java in particular, that relies on garbage collection to close the connections.

Op Counters

The utilization baselines for your application will help you determine a normal count of operations. If these counts start to substantially deviate from your baselines it may be an indicator that something has changed in the application, or that a malicious attack is underway.

Queues

If MongoDB is unable to complete all requests in a timely fashion, requests will begin to queue up. A healthy deployment will exhibit very low queues. If things start to deviate from baseline performance, caused by a high degree of page faults or a long-running query for example, requests from applications will begin to queue up. The queue is therefore a good first place to look to determine if there are issues that will affect user experience.

System Configuration

It is not uncommon to make changes to hardware and software in the course of a MongoDB deployment. For example, a disk subsystem may be replaced to provide better performance or increased capacity. When components are changed it is important to ensure their configurations are appropriate for the deployment. MongoDB is very sensitive to the performance of the operating system and underlying hardware, and in some cases the default values for system configurations are not ideal. For example, the default readahead for the file system could be several MB whereas MongoDB is optimized for readahead values closer to 32 KB. If the new storage system is installed without making the change to the readahead from the default to the appropriate setting, the application's performance is likely to degrade substantially.

Shard Balancing

One of the goals of sharding is to uniformly distribute data across multiple servers. If the utilization of server resources is not approximately equal across servers there may be an underlying issue that is problematic for the deployment. For example, a poorly selected shard key can result in uneven data distribution. In this case, most if not all of the queries will be directed to the single mongod that is managing the data. Furthermore, MongoDB may be attempting to redistribute the documents to achieve a more ideal balance across the servers. While redistribution will eventually result in a more desirable distribution of documents, there is substantial work associated with rebalancing the data and this activity itself may interfere with achieving the desired performance SLA. By running db.currentOp() you will be able to determine what work is currently being performed by the cluster, including rebalancing of documents across the shards.

In order to ensure data is evenly distributed across all shards in a cluster, it is important to select a good shard key. If in the course of a deployment it is determined that a new shard key should be used, it will be necessary to reload the data with a new shard key because shard keys and shard values are immutable. To support the use of a new shard key, it is possible to write a script that reads each document, updates the shard key, and writes it back to the database.

Replication Lag

Replication lag is the amount of time it takes a write operation on the primary replica set member to replicate to a secondary member. A small amount of delay is normal, but as replication lag grows, significant issues may arise. Typical causes of replication lag include network latency or connectivity issues, and disk latencies such as the throughput of the secondaries being inferior to that of the primary.

Config Server Availability

In sharded environments it is required to run three config servers. Config servers are critical to the system for understanding the location of documents across shards. If one config server goes down then the other two will go into read-only mode. The database will remain operational in this case, but the balancer will be unable to move chunks until all three config servers are available.

Disaster Recovery: Backup & Recovery

A backup and recovery strategy is necessary to protect your mission-critical data against catastrophic failure, such as a fire or flood in a data center, or human error such as code errors or accidentally dropping collections. With a backup and recovery strategy in place, administrators can restore business operations without data loss, and the organization can meet regulatory and compliance requirements. Taking regular backups offers other advantages, as well. The backups can be used to seed new environments for development, staging, or QA without impacting production systems.

Ops Manager and MMS backups are maintained continuously, just a few seconds behind the operational system. If the MongoDB cluster experiences a failure, the most recent backup is only moments behind, minimizing exposure to data loss. Ops Manager and MMS are the only MongoDB solutions that offer point-in-time backup of replica sets and cluster-wide snapshots of sharded clusters. You can restore to precisely the moment you need, quickly and safely.

Because Ops Manager and MMS only read the oplog, the ongoing performance impact is minimal – similar to that of adding an additional replica to a replica set.

By using MongoDB Enterprise Advanced you can deploy Ops Manager to control backups in your local data center, or use the MMS cloud service which offers a fully managed backup solution with a pay-as-you-go model. Dedicated MongoDB engineers monitor user backups on a 24x365 basis, alerting operations teams if problems arise.

Ops Manager and MMS is not the only mechanism for backing up MongoDB. Other options include:
* File system copies
* The mongodump tool packaged with MongoDB.

File System Backups

File system backups, such as that provided by Linux LVM, quickly and efficiently create a consistent snapshot of the file system that can be copied for backup and restore purposes. For databases with a single replica set it is possible to stop operations temporarily so that a consistent snapshot can be created by issuing the db.fsyncLock() command. This will flush all pending writes to disk and lock the entire mongod instance to prevent additional writes until the lock is released with db.fsyncUnlock(). Note, for MongoDB instances configured with the WiredTiger storage engine, this will only work if the journal is co-located on the same volume as the data files.

For more on how to use file system snapshots to create a backup of MongoDB, please see Backup and Restore with Filesystem Snapshots in the MongoDB Documentation.

Only Ops Manager and MMS provide an automated method for locking all shards in a cluster for backup purposes. If you are not using these platforms, the process for creating a backup follows these approximate steps:

- Stop the balancer so that chunks are consistent across shards in the cluster.
- Stop one of the config servers to prevent all metadata changes.
- Lock one replica of each of the shards using db.fsyncLock().
- Create a backup of one of the config servers.
- Create the file system snapshot for each of the locked replicas.
- Unlock all the replicas.
- Start the config server.
- Start the balancer.

For more on backup and restore in sharded environments, see the MongoDB Documentation page on Backup and Restore Sharded Clusters and the tutorial on Backup a Sharded Cluster with Filesystem Snapshots.

mongodump

mongodump is a tool bundled with MongoDB that performs a live backup of the data in MongoDB. mongodump may be used to dump an entire database, collection, or result of a query. mongodump can produce a dump of the data that reflects a single moment in time by dumping the oplog and then replaying it during mongorestore, a tool that imports content from BSON database dumps produced by mongodump. mongodump can also work against an inactive set of database files.

Integrating MongoDB with External Monitoring Solutions

The Ops Manager and MMS API provides integration with external management frameworks through programmatic access to automation features and monitoring data.

In addition to Ops Manager and MMS, MongoDB Enterprise Advanced can report system information to SNMP traps, supporting centralized data collection and aggregation via external monitoring solutions. Review the documentation to learn more about SNMP integration.

Security

As with all software, MongoDB administrators must consider security and risk exposure for a MongoDB deployment. There are no magic solutions for risk mitigation, and maintaining a secure MongoDB deployment is an ongoing process.

Defense in Depth

A Defense in Depth approach is recommended for securing MongoDB deployments, and it addresses a number of different methods for managing risk and reducing risk exposure.

The intention of a Defense in Depth approach is to layer your environment to ensure there are no exploitable single points of failure that could allow an intruder or un-trusted party to access the data stored in the MongoDB database. The most effective way to reduce the risk of exploitation is to run MongoDB in a trusted environment, to limit access, to follow a system of least privileges, to institute a secure development lifecycle and to follow deployment best practices.

MongoDB Enterprise Advanced features extensive capabilities to defend, detect and control access to MongoDB, offering among the most complete security controls of any modern database.

- **User Rights Management.** Control access to sensitive data using industry standard mechanisms for authentication and authorization to the database, collection, and down to the level of individual fields within a document.
- **Auditing.** Ensure regulatory and internal compliance.
- **Encryption.** Protect data in motion over the network and at rest in persistent storage.
- **Administrative Controls.** Identify potential exploits faster and reduce their impact.

Review the MongoDB Security Reference Architecture to learn more about each of the security features discussed below.

Authentication

Authentication can be managed from within the database itself or via MongoDB Enterprise Advanced integration with external security mechanisms including LDAP, Windows Active Directory, Kerberos, and x.509 certificates.

Authorization

MongoDB allows administrators to define permissions for a user or application, and what data it can access when querying the database. MongoDB provides the ability to configure granular user-defined roles, making it possible to realize a separation of duties between different entities accessing and managing the database.

Additionally, MongoDB's Aggregation Pipeline includes a stage to implement Field-Level Redaction, providing a method to restrict the content of a returned document on a per-field level, based on user permissions. The application must pass the redaction logic to the database on each request. It therefore relies on trusted middleware running in the application to ensure the redaction pipeline stage is appended to any query that requires the redaction logic.

Auditing

MongoDB Enterprise Advanced enables security administrators to construct and filter audit trails for any operation against MongoDB, whether DML, DCL or DDL. For example, it is possible to log and audit the identities of users who retrieved specific documents, and any changes made to the database during their session. The audit log can be written to multiple destinations in a variety of formats including to the console and syslog (in JSON format), and to a file (JSON or BSON), which can then be loaded to MongoDB and analyzed to identify relevant events

Encryption

MongoDB data can be encrypted on the network and on disk.

Support for SSL allows clients to connect to MongoDB over an encrypted channel. MongoDB supports FIPS 140-2 encryption when run in FIPS Mode with a FIPS validated Cryptographic module.

Data at rest can be protected using either certified database encryption solutions from MongoDB partners such as IBM and Vormetric, or within the application itself.

Data encryption software should ensure that the cryptographic keys remain safe and enable compliance with standards such as HIPAA, PCI-DSS and FERPA.

Monitoring

Database monitoring is critical in identifying and protecting against potential exploits, reducing the impact of any attempted breach. Ops Manager and MMS users can visualize database performance and set custom alerts that notify when particular metrics are out of normal range.

Query Injection

As a client program assembles a query in MongoDB, it builds a BSON object, not a string. Thus traditional SQL injection attacks should not pose a risk to the system for queries submitted as BSON objects.

However, several MongoDB operations permit the evaluation of arbitrary Javascript expressions and care should be taken to avoid malicious expressions. Fortunately most queries can be expressed in BSON and for cases where Javascript is required, it is possible to mix Javascript and BSON so that user-specified values are evaluated as values and not as code.

MongoDB can be configured to prevent the execution of Javascript scripts. This will prevent MapReduce jobs from running, but the aggregation framework can be used as an alternative in many use cases.

Conclusion

MongoDB is the next-generation database used by the world's most sophisticated organizations, from cutting-edge startups to the largest companies, to create applications never before possible at a fraction of the cost of legacy databases. MongoDB is the fastest-growing database ecosystem, with over 9 million downloads, thousands of customers, and over 700 technology and service partners. MongoDB users rely on the best practices discussed in this guide to maintain the highly available, secure and scalable operations demanded by organizations today.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than a third of the Fortune 100. We offer software and services to make your life easier:

MongoDB Enterprise Advanced is the best way to run MongoDB in your data center. It's a finely-tuned package of advanced software, support, certifications, and other services designed for the way you do business.

MongoDB Management Service (MMS) is the easiest way to run MongoDB in the cloud. It makes MongoDB the system you worry about the least and like managing the most.

Production Support helps keep your system up and running and gives you peace of mind. MongoDB engineers help you with production issues and any aspect of your project.

Development Support helps you get up and running quickly. It gives you a complete package of software and services for the early stages of your project.

MongoDB Consulting packages get you to production faster, help you tune performance in production, help you scale, and free you up to focus on your next release.

MongoDB Training helps you become a MongoDB expert, from design to operating mission-critical systems at scale. Whether you're a developer, DBA, or architect, we can make you better at MongoDB.

Resources

For more information, please visit mongodb.com or contact us at sales@mongodb.com.

Case Studies (mongodb.com/customers)
Presentations (mongodb.com/presentations)
Free Online Training (university.mongodb.com)
Webinars and Events (mongodb.com/events)
Documentation (docs.mongodb.org)
MongoDB Enterprise Download (mongodb.com/download)