

MongoDB Architecture Guide

MongoDB 3.4
November 2016

Table of Contents

Introduction	1
How we Build & Run Modern Apps	1
The Nexus Architecture	2
MongoDB Multimodel Architecture	3
MongoDB Data Model	4
MongoDB Query Model	6
MongoDB Data Management	8
Consistency & Durability	9
Availability	10
Performance & Compression	12
Security	12
Operational Management	13
MongoDB Atlas	15
Conclusion	15
We Can Help	16
Resources	16

Introduction

“MongoDB wasn’t designed in a lab. We built MongoDB from our own experiences building large-scale, high availability, robust systems. We didn’t start from scratch, we really tried to figure out what was broken, and tackle that. So the way I think about MongoDB is that if you take MySQL, and change the data model from relational to document-based, you get a lot of great features: embedded docs for speed, manageability, agile development with dynamic schemas, easier horizontal scalability because joins aren’t as important. There are a lot of things that work great in relational databases: indexes, dynamic queries and updates to name a few, and we haven’t changed much there. For example, the way you design your indexes in MongoDB should be exactly the way you do it in MySQL or Oracle, you just have the option of indexing an embedded field.”

— *Eliot Horowitz, MongoDB CTO and Co-Founder*

MongoDB is designed for how we build and run data-driven applications with modern development techniques, programming models, computing resources, and operational automation.

How We Build & Run Modern Applications

Relational databases have a long-standing position in most organizations, and for good reason. Relational databases underpin existing applications that meet current business needs; they are supported by an extensive ecosystem of tools; and there is a large pool of labor qualified to implement and maintain these systems.

But organizations are increasingly considering alternatives to legacy relational infrastructure, driven by challenges presented in building modern applications:

- Developers are working with applications that create massive volumes of new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data.
- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.

- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices on any channel, and scaled globally to millions of users.
- Organizations are now turning to scale-out architectures using open source software, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

The Nexus Architecture

MongoDB's design philosophy is focused on combining the critical capabilities of relational databases with the innovations of NoSQL technologies. Our vision is to leverage the work that Oracle and others have done over the last 40 years to make relational databases what they are today. Rather than discard decades of proven database maturity, MongoDB is picking up where they left off by combining key relational database capabilities with the work that Internet pioneers have done to address the requirements of modern applications.

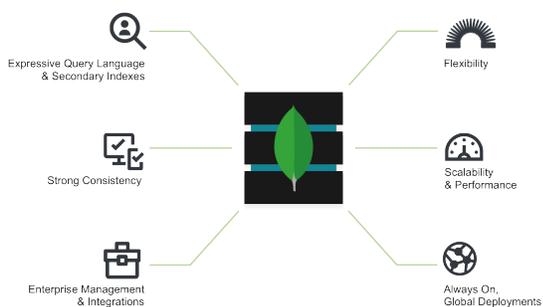


Figure 1: MongoDB Nexus Architecture, blending the best of relational and NoSQL technologies

Relational databases have reliably served applications for many years, and offer features that remain critical today as developers build the next generation of applications:

- **Expressive query language & secondary indexes.** Users should be able to access and manipulate their data in sophisticated ways to support both operational and analytical applications. Indexes play a critical role in providing efficient access to data, supported natively by the database rather than maintained in application code.

- **Strong consistency.** Applications should be able to immediately read what has been written to the database. It is much more complex to build applications around an eventually consistent model, imposing significant work on the developer, even for the most sophisticated engineering teams.
- **Enterprise Management and Integrations.** Databases are just one piece of application infrastructure, and need to fit seamlessly into the enterprise IT stack. Organizations need a database that can be secured, monitored, automated, and integrated with their existing technology infrastructure, processes, and staff, including operations teams, DBAs, and data engineers.

However, modern applications impose requirements not addressed by relational databases, and this has driven the development of NoSQL databases which offer:

- **Flexible Data Model.** NoSQL databases emerged to address the requirements for the data we see dominating modern applications. Whether document, graph, key-value, or wide-column, all of them offer a flexible data model, making it easy to store and combine data of any structure and allow dynamic modification of the schema without downtime or performance impact.
- **Scalability and Performance.** NoSQL databases were all built with a focus on scalability, so they all include some form of sharding or partitioning. This allows the database to be scaled out across commodity hardware deployed on-premises or in the cloud, enabling almost unlimited growth with higher throughput and lower latency than relational databases.
- **Always-On Global Deployments.** NoSQL databases are designed for continuously available systems that provide a consistent, high quality experience for users all over the world. They are designed to run across many nodes, including replication to automatically synchronize data across servers, racks, and geographically-dispersed data centers.

While offering these innovations, NoSQL systems have sacrificed the critical capabilities that people have come to expect and rely upon from relational databases. MongoDB offers a different approach. With its Nexus Architecture, MongoDB is the only database that harnesses the

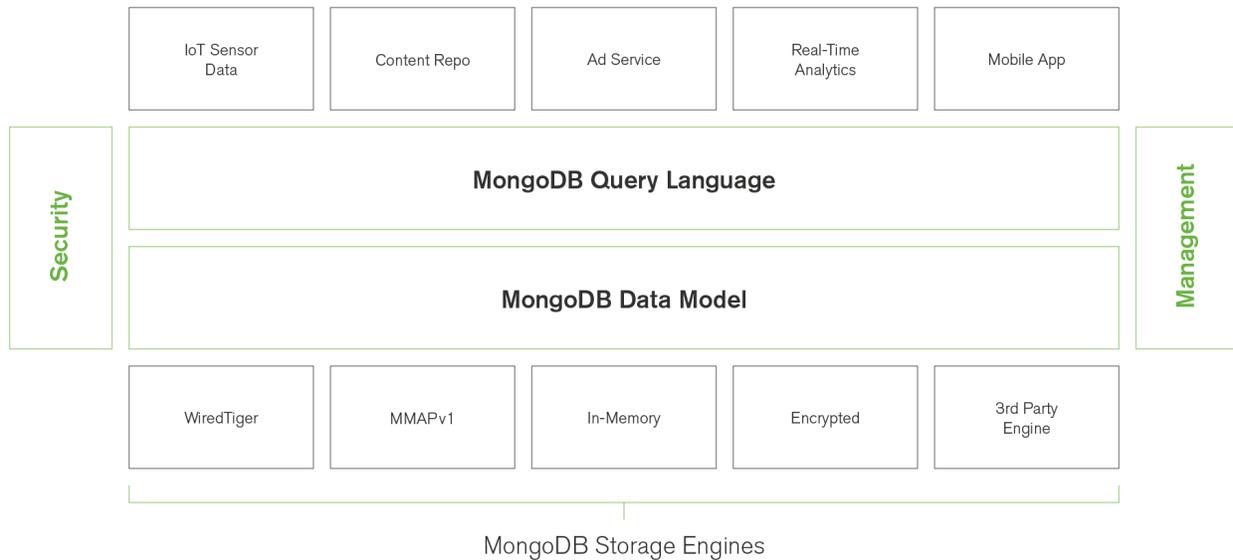


Figure 2: Flexible Storage Architecture, optimising MongoDB for unique application demands

innovations of NoSQL while maintaining the foundation of relational databases.

MongoDB Multimodel Architecture

MongoDB embraces two key trends in modern IT:

- Organizations are rapidly expanding the range of applications they deliver to digitally transform the business.
- CIOs are rationalizing their technology portfolios to a strategic set of vendors they can leverage to more efficiently support their business.

With MongoDB, organizations can address diverse application needs, hardware resources, and deployment designs with a single database technology:

- MongoDB's flexible document data model presents a superset of other database models. It allows data be represented as simple key-value pairs and flat, table-like structures, through to rich documents and objects with deeply nested arrays and sub-documents
- With an expressive query language, documents can be queried in many ways – from simple lookups to creating sophisticated processing pipelines for data analytics

and transformations, through to faceted search, JOINS and graph traversals.

- With a flexible storage architecture, application owners can deploy storage engines optimized for different workload and operational requirements.

MongoDB's multimodel design significantly reduces developer and operational complexity when compared to running multiple distinct database technologies to meet different applications needs. Users can leverage the same MongoDB query language, data model, scaling, security, and operational tooling across different parts of their application, with each powered by the optimal storage engine.

Flexible Storage Architecture

MongoDB uniquely allows users to mix and match multiple storage engines within a single deployment. This flexibility provides a more simple and reliable approach to meeting diverse application needs for data. Traditionally, multiple database technologies would need to be managed to meet these needs, with complex, custom integration code to move data between the technologies, and to ensure consistent, secure access. With MongoDB's flexible storage architecture, the database automatically manages the movement of data between storage engine technologies using native replication.

MongoDB 3.4 ships with four supported storage engines, all of which can coexist within a single MongoDB replica set. This makes it easy to evaluate and migrate between them, and to optimize for specific application requirements – for example combining the in-memory engine for ultra low-latency operations with a disk-based engine for persistence. The supported storage engines include:

- The default WiredTiger storage engine. For many applications, WiredTiger's granular concurrency control and native compression will provide the best all round performance and storage efficiency for the broadest range of applications.
- The Encrypted storage engine protecting highly sensitive data, without the performance or management overhead of separate filesystem encryption. (Requires [MongoDB Enterprise Advanced](#)).
- The In-Memory storage engine delivering the extreme performance coupled with real time analytics for the most demanding, latency-sensitive applications. (Requires [MongoDB Enterprise Advanced](#)).
- The MMAPv1 engine, an improved version of the storage engine used in pre-3.x MongoDB releases.

MongoDB Data Model

Data As Documents

MongoDB stores data in a binary representation called [BSON \(Binary JSON\)](#). The BSON encoding extends the popular JSON (JavaScript Object Notation) representation to include additional types such as int, long, date, floating point, and decimal128. BSON documents contain one or more fields, and each field contains a value of a specific data type, including arrays, binary data and sub-documents.

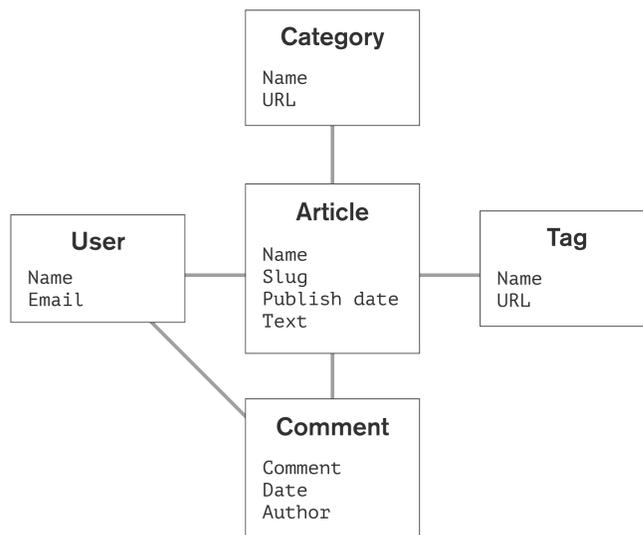


Figure 3: Example relational data model for a blogging application

Documents that tend to share a similar structure are organized as collections. It may be helpful to think of collections as being analogous to a table in a relational database: documents are similar to rows, and fields are similar to columns.

For example, consider the data model for a blogging application. In a relational database the data model would comprise multiple tables. To simplify the example, assume there are tables for Categories, Tags, Users, Comments and Articles.

In MongoDB the data could be modeled as two collections, one for users, and the other for articles. In each blog document there might be multiple comments, multiple tags, and multiple categories, each expressed as an embedded array.

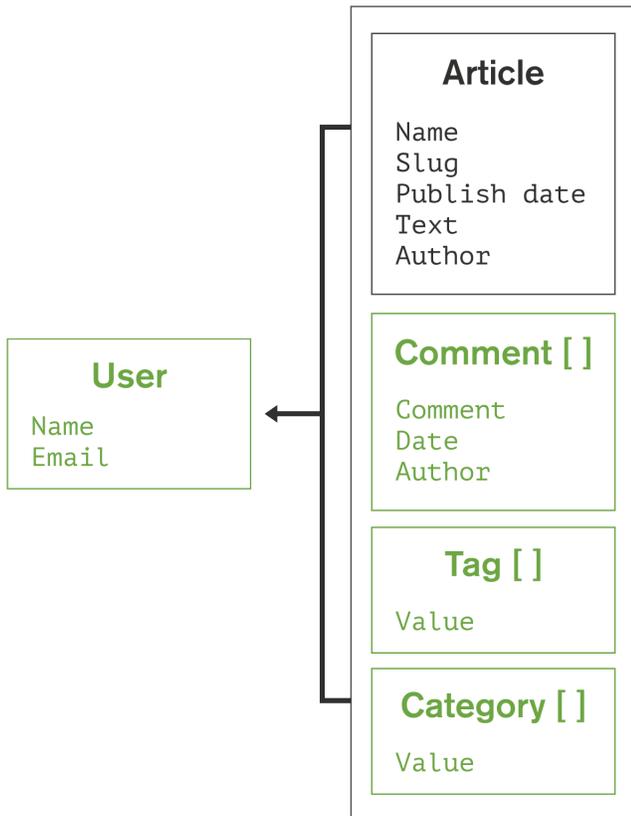


Figure 4: Data as documents: simpler for developers, faster for users.

As this example illustrates, MongoDB documents tend to have all data for a given record in a single document, whereas in a relational database information for a given record is usually spread across many tables. With the MongoDB document model, data is more localized, which significantly reduces the need to JOIN separate tables. The result is dramatically higher performance and scalability across commodity hardware as a single read to the database can retrieve the entire document containing all related data. Unlike many NoSQL databases, users don't need to give up JOINS entirely. For additional flexibility, MongoDB provides the ability to perform left-outer JOINS that combine data from multiple collections.

MongoDB BSON documents are closely aligned to the structure of objects in the programming language. This makes it simpler and faster for developers to model how data in the application will map to data stored in the database.

Dynamic Schema without Compromising Data Governance

MongoDB documents can vary in structure. For example, all documents that describe users might contain the user id and the last date they logged into the system, but only some of these documents might contain the user's identity for one or more third party applications. Fields can vary from document to document; there is no need to declare the structure of documents to the system – documents are self describing. If a new field needs to be added to a document then the field can be created without affecting all other documents in the system, without updating a central system catalog, and without taking the system offline.

Developers can start writing code and persist the objects as they are created. And when developers add more features, MongoDB continues to store the updated objects without the need to perform costly `ALTER TABLE` operations, or worse – having to re-design the schema from scratch.

Document Validation

Dynamic schemas bring great agility, but it is also important that controls can be implemented to maintain data quality, especially if the database is shared by multiple applications. Unlike NoSQL databases that push enforcement of these controls back into application code, MongoDB provides document validation within the database. Users can enforce checks on document structure, data types, data ranges and the presence of mandatory fields. As a result, DBAs can apply data governance standards, while developers maintain the benefits of a flexible document model.

Schema Design

Although MongoDB provides schema flexibility, schema design is still important. Developers and DBAs should consider a number of topics, including the types of queries the application will need to perform, relationships between data, how objects are managed in the application code, and how documents will change over time. Schema design is an extensive topic that is beyond the scope of this document.

For more information, please see [Data Modeling Considerations](#).

MongoDB Query Model

Idiomatic Drivers

MongoDB provides native drivers for all popular programming languages and frameworks to make development natural. Supported drivers include Java, Javascript, .NET, Python, Perl, PHP, Scala and others, in addition to 30+ community-developed drivers. MongoDB drivers are designed to be idiomatic for the given language.

One fundamental difference with relational databases is that the MongoDB query model is implemented as methods or functions within the API of a specific programming language, as opposed to a completely separate language like SQL. This, coupled with the affinity between MongoDB's JSON document model and the data structures used in object-oriented programming, makes integration with applications simple. For a complete list of drivers see the [MongoDB Drivers](#) page.

Interacting with the Database

MongoDB offers developers and administrators a range of tools for interacting with the database, independent of the drivers.

The [mongo shell](#) is a rich, interactive JavaScript shell that is included with all MongoDB distributions. Additionally [MongoDB Compass](#) is a sophisticated and intuitive GUI for MongoDB. Offering rich schema exploration and management, Compass allows DBAs to modify documents, create validation rules, and efficiently optimize query performance by visualizing explain plans and index usage. Sophisticated queries can be built and executed by simply selecting document elements from the user interface, with the results viewed both graphically, and as a set of JSON documents. All of these tasks can be accomplished from a point and click interface, and all with zero knowledge of MongoDB's query language.

MongoDB Compass is included with both [MongoDB Professional](#) and [MongoDB Enterprise Advanced](#)

subscriptions used with your self-managed instances, or hosted [MongoDB Atlas](#) instances. MongoDB Compass is free to use for evaluation and in development environments.

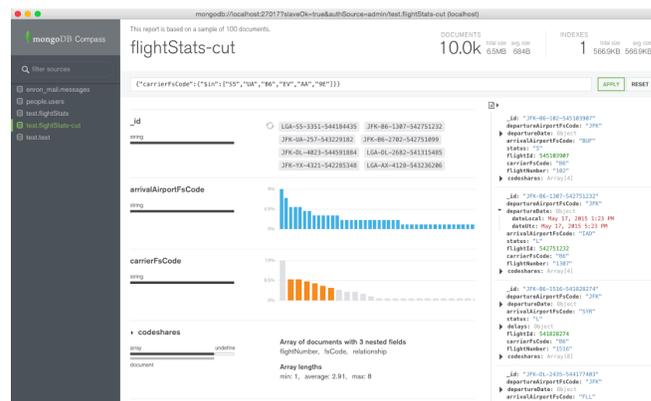


Figure 5: Interactively build and execute database queries with MongoDB Compass

Querying and Visualizing Data

Unlike NoSQL databases, MongoDB is not limited to simple Key-Value operations. Developers can build rich applications using complex queries, aggregations and secondary indexes that unlock the value in structured, semi-structured and unstructured data.

A key element of this flexibility is MongoDB's support for many types of queries. A query may return a document, a subset of specific fields within the document or complex aggregations and transformation of many documents:

- **Key-value queries** return results based on any field in the document, often the primary key.
- **Range queries** return results based on values defined as inequalities (e.g, greater than, less than or equal to, between).
- **Geospatial queries** return results based on proximity criteria, intersection and inclusion as specified by a point, line, circle or polygon.
- **Search queries** return results in relevance order and in faceted groups, based on text arguments using Boolean operators (e.g., AND, OR, NOT), and through bucketing, grouping and counting of query results. With support for collations, data comparison and sorting order can be defined for over 100 different languages and locales.

- **Aggregation Framework** queries return aggregations and transformations of values returned by the query (e.g., count, min, max, average, similar to a SQL GROUP BY statement).
- **JOINS and graph traversals.** Through the `$lookup` stage of the aggregation pipeline, documents from separate collections can be combined through a left outer JOIN operation. `$graphLookup` brings native graph processing within MongoDB, enabling efficient traversals across trees, graphs and hierarchical data to uncover patterns and surface previously unidentified connections.
- **MapReduce queries** execute complex data processing that is expressed in JavaScript and executed across data in the database.

Additionally the [MongoDB Connector for Apache Spark](#) exposes Spark's Scala, Java, Python, and R libraries. MongoDB data is materialized as DataFrames and Datasets for analysis through machine learning, graph, streaming, and SQL APIs.

Data Visualization with BI Tools

With the [MongoDB Connector for BI](#) included in MongoDB Enterprise Advanced, modern application data can be easily analyzed with industry-standard SQL-based BI and analytics platforms. Business analysts and data scientists can seamlessly analyze semi and unstructured data managed in MongoDB, alongside traditional data in their SQL databases using the same BI tools deployed within millions of enterprises.

Indexing

Indexes are a crucial mechanism for optimizing system performance and scalability while providing flexible access to the data. Like most database management systems, while indexes will improve the performance of some operations by orders of magnitude, they incur associated overhead in write operations, disk usage, and memory consumption. By default, the WiredTiger storage engine compresses indexes in RAM, freeing up more of the working set for documents.

MongoDB includes support for many types of secondary indexes that can be declared on any field in the document, including fields within arrays:

- **Unique Indexes.** By specifying an index as unique, MongoDB will reject inserts of new documents or the update of a document with an existing value for the field for which the unique index has been created. By default all indexes are not set as unique. If a compound index is specified as unique, the combination of values must be unique.
- **Compound Indexes.** It can be useful to create compound indexes for queries that specify multiple predicates. For example, consider an application that stores data about customers. The application may need to find customers based on last name, first name, and city of residence. With a compound index on last name, first name, and city of residence, queries could efficiently locate people with all three of these values specified. An additional benefit of a compound index is that any leading field within the index can be used, so fewer indexes on single fields may be necessary; this compound index would also optimize queries looking for customers by last name.
- **Array Indexes.** For fields that contain an array, each array value is stored as a separate index entry. For example, documents that describe products might include a field for components. If there is an index on the component field, each component is indexed and queries on the component field can be optimized by this index. There is no special syntax required for creating array indexes – if the field contains an array, it will be indexed as a array index.
- **TTL Indexes.** In some cases data should expire out of the system automatically. Time to Live (TTL) indexes allow the user to specify a period of time after which the data will automatically be deleted from the database. A common use of TTL indexes is applications that maintain a rolling window of history (e.g., most recent 100 days) for user actions such as clickstreams.
- **Geospatial Indexes.** MongoDB provides geospatial indexes to optimize queries related to location within a two dimensional space, such as projection systems for the earth. These indexes allow MongoDB to optimize queries for documents that contain points or a polygon

that are closest to a given point or line; that are within a circle, rectangle, or polygon; or that intersect with a circle, rectangle, or polygon.

- **Partial Indexes.** By specifying a filtering expression during index creation, a user can instruct MongoDB to include only documents that meet the desired conditions, for example by only indexing active customers. Partial indexes balance delivering low latency query performance while reducing system overhead.
- **Sparse Indexes.** Sparse indexes only contain entries for documents that contain the specified field. Because the document data model of MongoDB allows for flexibility in the data model from document to document, it is common for some fields to be present only in a subset of all documents. Sparse indexes allow for smaller, more efficient indexes when fields are not present in all documents.
- **Text Search Indexes.** MongoDB provides a specialized index for text search that uses advanced, language-specific linguistic rules for stemming, tokenization, case sensitivity and stop words. Queries that use the text search index will return documents in relevance order. One or more fields can be included in the text index.

Query Optimization

MongoDB automatically optimizes queries to make evaluation as efficient as possible. Evaluation normally includes selecting data based on predicates, and sorting 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 best response time for each query type. The results of this empirical test are stored as a cached query plan and are updated periodically. Developers can review and optimize plans using the powerful explain method and index filters. Using MongoDB Compass, DBAs can visualize index coverage, enabling them to determine which specific fields are indexed, their type, size, and how often they are used. Compass also provides the ability to visualize explain plans, presenting key information on how a query performed – for example the number of documents returned, execution time, index usage, and more. Each

stage of the execution pipeline is represented as a node in a tree, making it simple to view explain plans from queries distributed across multiple nodes.

Index intersection provides additional flexibility by enabling MongoDB to use more than one index to optimize an ad-hoc query at run-time.

Covered Queries

Queries that return results containing only indexed fields are called covered queries. These results can be returned without reading from the source documents. With the appropriate indexes, workloads can be optimized to use predominantly covered queries.

MongoDB Data Management

Auto-Sharding

MongoDB provides horizontal scale-out for databases on low cost, commodity hardware or cloud infrastructure using a technique called **sharding**, which is transparent to applications. Sharding distributes data across multiple physical partitions called shards. Sharding allows MongoDB deployments to address the hardware limitations of a single server, such as bottlenecks in RAM or disk I/O, without adding complexity to the application. MongoDB automatically balances the data in the sharded cluster as the data grows or the size of the cluster increases or decreases.

Unlike relational databases, sharding is automatic and built into the database. Developers don't face the complexity of building sharding logic into their application code, which then needs to be updated as shards are migrated. Operations teams don't need to deploy additional clustering software or expensive shared-disk infrastructure to manage process and data distribution or failure recovery.



Figure 6: Automatic sharding provides horizontal scalability in MongoDB.

Unlike other distributed databases, multiple sharding policies are available that enable developers and administrators to distribute data across a cluster according to query patterns or data locality. As a result, MongoDB delivers much higher scalability across a diverse set of workloads:

- **Range 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 Sharding.** Documents are distributed according to an MD5 hash of the shard key value. This approach guarantees a uniform distribution of writes across shards, but is less optimal for range-based queries.
- **Zone Sharding.** Provides the the ability for DBAs and operations teams to define specific rules governing data placement in a sharded cluster. Zones accommodate a range of deployment scenarios – for example locating data by geographic region, by hardware configuration for tiered storage architectures, or by application feature. Administrators can continuously refine data placement rules by modifying shard key ranges, and MongoDB will automatically migrate the data to its new zone.

Tens of thousands of organizations use MongoDB to build high-performance systems at scale. You can read more about them on the [MongoDB scaling page](#).

Query Router

Sharding is transparent to applications; whether there is one or one hundred shards, the application code for querying MongoDB is the same. Applications issue queries to a query router that dispatches the query to the appropriate shards.

For key-value queries that are based on the shard key, the query router will dispatch the query to the shard that manages the document with the requested key. When using range-based sharding, queries that specify ranges on the shard key are only dispatched to shards that contain documents with values within the range. For queries that don't use the shard key, the query router will broadcast the

query to all shards, aggregating and sorting the results as appropriate. Multiple query routers can be used with a MongoDB system, and the appropriate number is determined based on performance and availability requirements of the application.

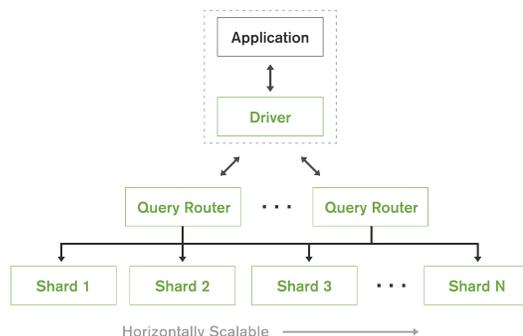


Figure 7: Sharding is transparent to applications.

Consistency

Transaction Model & Configurable Write Availability

MongoDB is ACID compliant at the document level. One or more fields may be written in a single operation, including updates to multiple sub-documents and elements of an array. The ACID guarantees provided by MongoDB ensures complete isolation as a document is updated; any errors cause the operation to roll back and clients receive a consistent view of the document.

MongoDB also allows users to specify write availability in the system using an option called the write concern. The default write concern acknowledges writes from the application, allowing the client to catch network exceptions and duplicate key errors. Developers can use MongoDB's Write Concerns to configure operations to commit to the application only after specific policies have been fulfilled – for example only after the operation has been flushed to the journal on disk. This is the same mode used by many traditional relational databases to provide durability guarantees. As a distributed system, MongoDB presents additional flexibility in enabling users to achieve their desired durability goals, such as writing to at least two replicas in one data center and one replica in a second data center. Each query can specify the appropriate write concern, ranging from unacknowledged to

acknowledgement that writes have been committed to all replicas.

Availability

Replication

MongoDB maintains multiple copies of data called replica sets using native [replication](#). A replica set is a fully self-healing shard that helps prevent database downtime and can be used to scale read operations. Replica failover is fully automated, eliminating the need for administrators to intervene manually.

A replica set consists of multiple replicas. At any given time one member acts as the primary replica set member and the other members act as secondary replica set members. MongoDB is strongly consistent by default: reads and writes are issued to a primary copy of the data. If the primary member fails for any reason (e.g., hardware failure, network partition) one of the secondary members is automatically elected to primary, typically within several seconds. As discussed below, sophisticated rules govern which secondary replicas are evaluated for promotion to the primary member.

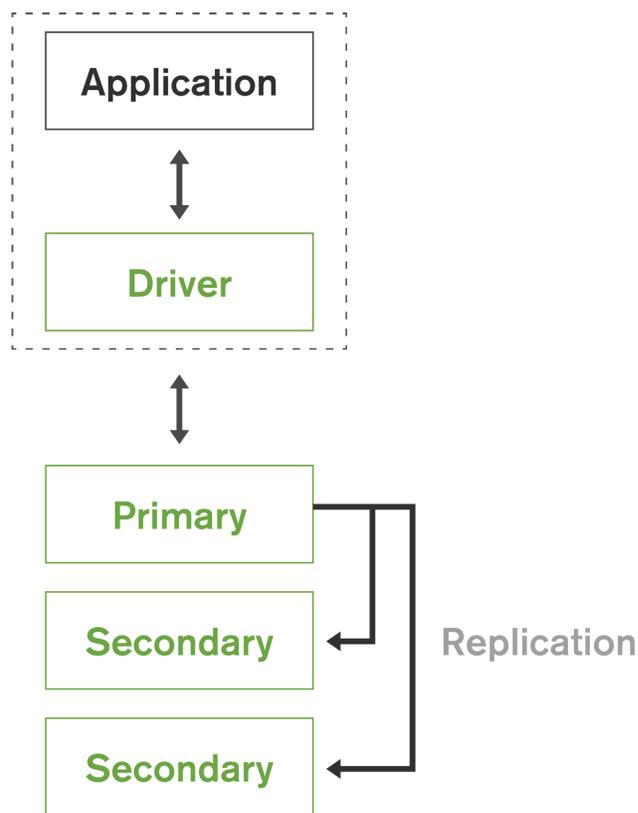


Figure 8: Self-Healing MongoDB Replica Sets for High Availability

The number of replicas in a MongoDB replica set is configurable: a larger number of replicas provides increased data durability and protection against database downtime (e.g., in case of multiple machine failures, rack failures, data center failures, or network partitions). Up to 50 members can be provisioned per replica set.

Enabling tunable consistency, applications can optionally read from secondary replicas, where data is eventually consistent by default. Reads from secondaries can be useful in scenarios where it is acceptable for data to be slightly out of date, such as some reporting and analytical applications. Administrators can control which secondary members service a query, based on a consistency window defined in the driver. For data-center aware reads, applications can also read from the closest copy of the data as measured by ping distance to reduce the effects of geographic latency. For more on reading from secondaries see the entry on [Read Preference](#).

Replica sets also provide operational flexibility by providing a way to upgrade hardware and software without requiring

the database to be taken offline. This is an important feature as these types of operations can account for as much as one third of all downtime in traditional systems.

Replica Set Oplog

Operations that modify a database on the primary replica set member are replicated to the secondary members using the oplog (operations log). The oplog contains an ordered set of idempotent operations that are replayed on the secondaries. The size of the oplog is configurable and by default is 5% of the available free disk space. For most applications, this size represents many hours of operations and defines the recovery window for a secondary, should this replica go offline for some period of time and need to catch up to the primary when it recovers.

If a secondary replica set member is down for a period longer than is maintained by the oplog, it must be recovered from the primary replica using a process called initial synchronization. During this process all databases with their collections and indexes are copied from the primary or another replica to the secondary. Initial synchronization is also performed when adding a new member to a replica set, or migrating between MongoDB storage engines. For more information see the page on [Replica Set Data Synchronization](#).

Elections And Failover

Replica sets reduce operational overhead and improve system availability. If the primary replica for a shard fails, secondary replicas together determine which replica should be elected as the new primary using an extended implementation of the Raft consensus algorithm. Once the election process has determined the new primary, the secondary members automatically start replicating from it. If the original primary comes back online, it will recognize its change in state and automatically assume the role of a secondary.

Election Priority

Sophisticated algorithms control the replica set 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"). In a typical deployment, a new primary replica set member is promoted within several seconds of the original primary failing. During this time, queries configured with the appropriate read preference can continue to be serviced by secondary replica set members. The election algorithms evaluate a range of parameters including analysis of election identifiers and 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. In an election, the replica set elects an eligible member with the highest priority value as primary. By default, all members have a priority of 1 and have an equal chance of becoming primary; however, it is possible to set priority values that affect the likelihood of a replica becoming primary.

In some deployments, there may be operational requirements that can be addressed with election priorities. For instance, all replicas located in a secondary data center could be configured with a priority so that one of them would only become primary if the main data center fails.

Performance & Compression

In-Memory Performance With On-Disk Capacity

With the In-Memory storage engine, MongoDB users can realize the performance advantages of in-memory computing for operational and real-time analytics workloads. The In-Memory storage engine delivers the extreme throughput and predictable latency demanded by the most performance-intensive applications in AdTech, finance, telecoms, IoT, eCommerce and more, eliminating the need for separate caching layers.

MongoDB replica sets allow for hybrid in-memory and on-disk database deployments. Data managed by the In-Memory engine can be processed and analyzed in real time, before being automatically replicated to MongoDB instances configured with the persistent disk-based WiredTiger storage engine. Lengthy ETL cycles typical when moving data between different databases is avoided,

and users no longer have to trade away the scalable capacity or durability guarantees offered by disk storage.

Storage & Network Efficiency with Compression

The WiredTiger and Encrypted storage engines support native compression, reducing physical storage footprint by as much as 80%. In addition to reduced storage space, compression enables much higher storage I/O scalability as fewer bits are read from disk.

Administrators have the flexibility to configure specific compression algorithms for collections, indexes and the journal, choosing between:

- **Snappy** (the default library for documents and the journal), provides the optimum balance between high document compression ratio – typically around 70%, dependent on data types – with low CPU overhead.
- **zlib**, providing higher document compression ratios for storage-intensive applications at the expense of extra CPU overhead.
- **Prefix compression** for indexes reducing the in-memory footprint of index storage by around 50%, freeing up more of the working set in RAM for frequently accessed documents. As with snappy, the actual compression ratio will be dependent on workload.

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.

As a distributed database, MongoDB relies on efficient network transport during query routing and inter-node replication. In addition to storage, MongoDB also offers intra-cluster network compression. Based on the snappy compression algorithm, network traffic can be compressed by up to 70%, providing major performance benefits in bandwidth-constrained environments, and reducing networking costs.

Security

The frequency and severity of data breaches continues to escalate. Industry analysts predict cybercrime will cost the global economy \$6 trillion annually by 2021. Organizations face an onslaught of new threat classes and threat actors with phishing, ransomware and intellectual property theft growing more than 50% year on year, and key infrastructure subject to increased disruption. With databases storing an organization's most important information assets, securing them is top of mind for administrators.

MongoDB Enterprise Advanced features extensive capabilities to defend, detect, and control access to data:

- **Authentication.** Simplifying access control to the database, MongoDB offers integration with external security mechanisms including LDAP, Windows Active Directory, Kerberos, and x.509 certificates.
- **Authorization.** User-defined roles enable administrators to configure granular permissions for a user or an application based on the privileges they need to do their job. These can be defined in MongoDB, or centrally within an LDAP server. Additionally, administrators can define views that expose only a subset of data from an underlying collection, i.e. a view that filters or masks specific fields, such as Personally Identifiable Information (PII) from customer data or health records.
- **Auditing.** For regulatory compliance, security administrators can use MongoDB's native audit log to track any operation taken against the database – whether DML, DCL or DDL.
- **Encryption.** MongoDB data can be encrypted on the network, on disk and in backups. With the Encrypted storage engine, protection of data at-rest is an integral feature within the database. By natively encrypting database files on disk, administrators eliminate both the management and performance overhead of external encryption mechanisms. Only those staff who have the appropriate database authorization credentials can access the encrypted data, providing additional levels of defence.

To learn more, download the [MongoDB Security Reference Architecture Whitepaper](#).

Managing MongoDB - Provisioning, Monitoring and Disaster Recovery

Created by the engineers who develop the database, [MongoDB Ops Manager](#) is the simplest way to run MongoDB in your own environment, making it easy for operations teams to deploy, monitor, backup and scale MongoDB. The capabilities of Ops Manager are also available in the [MongoDB Cloud Manager](#) tool hosted in the cloud. Organizations who run with MongoDB Enterprise Advanced can choose between Ops Manager and Cloud Manager for their deployments.

Ops Manager incorporates 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.

- **Deployment.** Any topology, at any scale;
- **Upgrade.** In minutes, with no downtime;
- **Scale.** Add capacity, without taking the application offline;
- **Visualize.** Graphically display query performance to identify and fix slow running operations;
- **Point-in-time, Scheduled Backups.** Restore complete running clusters to any point in time with just a few clicks, because disasters aren't predictable
- **Performance Alerts.** Monitor 100+ system metrics and get custom alerts before the system degrades.

Deployments and Upgrades

Ops Manager coordinates 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 activities.

Ops Manager is designed to adapt to problems as they arise by continuously assessing state and making adjustments as needed. Using a sophisticated rules engine, agents 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.

In addition to initial deployment, Ops Manager makes it possible to dynamically resize capacity by adding shards and replica set members. Other maintenance tasks such as upgrading MongoDB, building new indexes across replica sets 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.

Monitoring

High-performance distributed systems benefit from comprehensive monitoring. Ops Manager and Cloud Manager have been developed to give administrators the insights needed to ensure smooth operations and a great experience for end users.

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 where they are processed, aggregated, alerted and visualized in a browser, letting administrators easily determine the health of MongoDB in real-time. Historic performance can be reviewed in order to create operational baselines and to support capacity planning. The Visual Query Profiler provides a quick and convenient way for DBAs to analyze the performance of specific queries or query families. It can also provide recommendations on the addition of indexes that would improve performance of common operations.

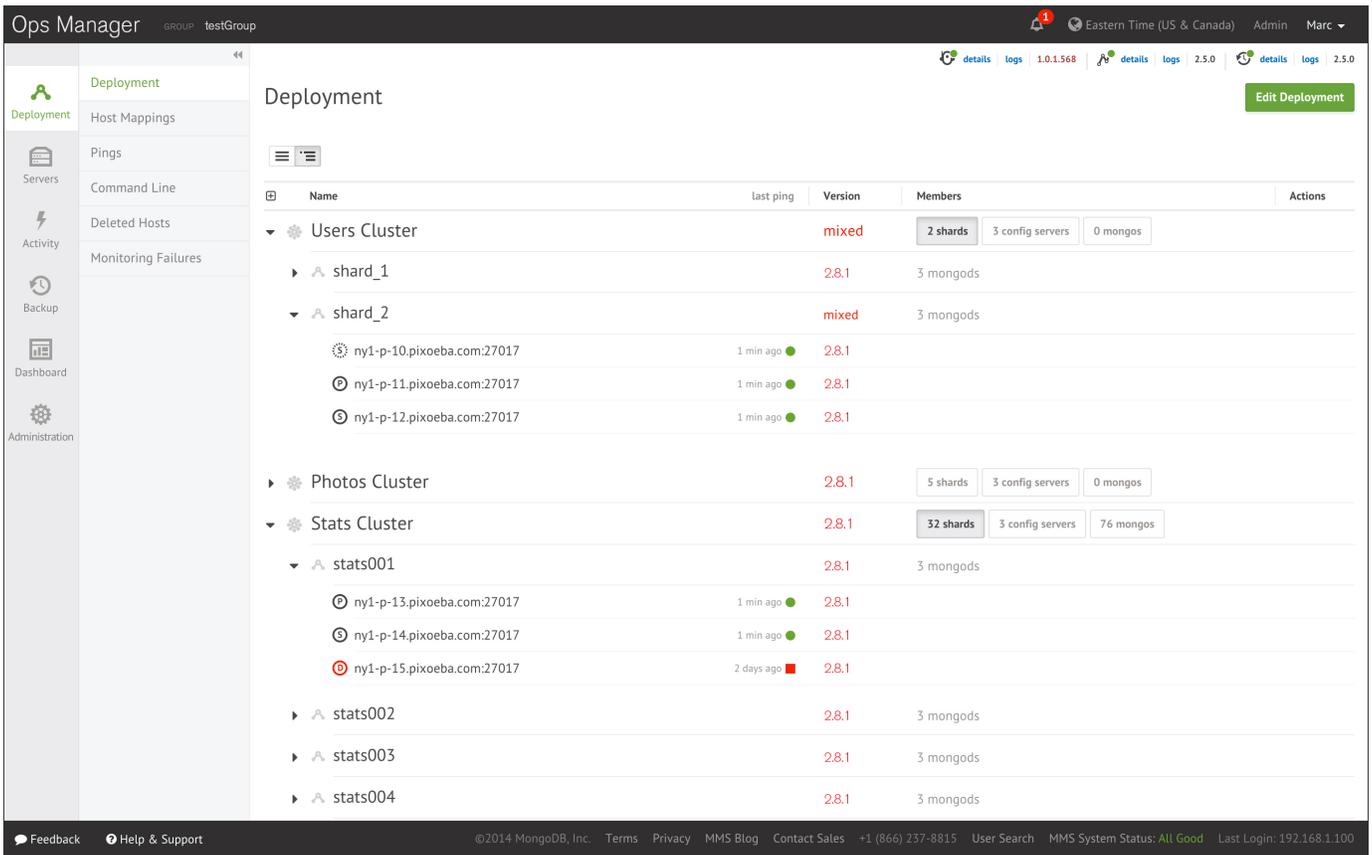


Figure 9: Ops Manager self-service portal: simple, intuitive and powerful. Deploy and upgrade entire clusters with a single click.

Integration with existing monitoring tools is also straightforward via the Ops Manager and Cloud Manager RESTful API, and with packaged integrations to leading Application Performance Management (APM) platforms such as New Relic. This integration allows MongoDB status to be consolidated and monitored alongside the rest

of your application infrastructure, all from a single pane of glass.

Ops Manager allows 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, Slack, HipChat and others to proactively warn of potential issues, before they escalate to costly outages.

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

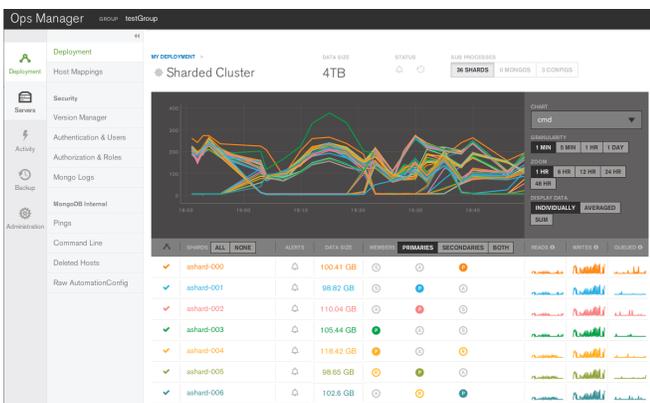


Figure 10: Ops Manager provides real time & historic visibility into the MongoDB deployment.

Disaster Recovery: Backups & Point-in-Time 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 create new environments for development, staging, or QA without impacting production.

Ops Manager and Cloud Manager backups are maintained continuously, just a few seconds behind the operational system. Because Ops Manager only reads the oplog, the ongoing performance impact is minimal – similar to that of adding an additional replica to a replica set. If the MongoDB cluster experiences a failure, the most recent backup is only moments behind, minimizing exposure to data loss. Ops Manager and Cloud Manager 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. Automation-driven restores allows fully a configured cluster to be re-deployed directly from the database snapshots in a just few clicks.

By using MongoDB Enterprise Advanced you can deploy Ops Manager to control backups in your local data center, or use the Cloud Manager 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.

SNMP: Integrating MongoDB with External Monitoring Solutions

In addition to Ops Manager and Cloud Manager, 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.

MongoDB Atlas: Database as a Service For MongoDB

[MongoDB Atlas](#) provides all of the features of MongoDB, without the operational heavy lifting required for any new application. MongoDB Atlas is available on-demand through a pay-as-you-go model and billed on an hourly basis, letting you focus on what you do best.

It's easy to get started – use a simple GUI to select the instance size, region, and features you need. MongoDB Atlas provides:

- Security features to protect access to your data
- Built in replication for always-on availability, tolerating complete data center failure
- Backups and point in time recovery to protect against data corruption
- Fine-grained monitoring to let you know when to scale. Additional instances can be provisioned with the push of a button
- Automated patching and one-click upgrades for new major versions of the database, enabling you to take advantage of the latest and greatest MongoDB features
- A choice of cloud providers, regions, and billing options

MongoDB Atlas is versatile. It's great for everything from a quick Proof of Concept, to test/QA environments, to complete production clusters. If you decide you want to bring operations back under your control, it is easy to move your databases onto your own infrastructure and manage them using MongoDB Ops Manager or MongoDB Cloud Manager. The user experience across MongoDB Atlas, Cloud Manager, and Ops Manager is consistent, ensuring that disruption is minimal if you decide to migrate to your own infrastructure.

MongoDB Atlas is automated, it's easy, and it's from the creators of MongoDB. [Learn more](#) and take it for a spin.

Conclusion

MongoDB is the database for today's applications: innovative, fast time-to-market, globally scalable, reliable,

and inexpensive to operate. In this guide we have explored the fundamental concepts and assumptions that underly the architecture of MongoDB. Other guides on topics such as Operations Best Practices can be found at mongodb.com.

We Can Help

We are the MongoDB experts. Over 2,000 organizations rely on our commercial products, including startups and more than 50% 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 Atlas is a database as a service for MongoDB, letting you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. With the click of a button, you can scale up and down when you need to, with no downtime, full security, and high performance.

MongoDB Cloud Manager is a cloud-based tool that helps you manage MongoDB on your own infrastructure. With automated provisioning, fine-grained monitoring, and continuous backups, you get a full management suite that reduces operational overhead, while maintaining full control over your databases.

MongoDB Professional helps you manage your deployment and keep it running smoothly. It includes support from MongoDB engineers, as well as access to MongoDB Cloud Manager.

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.com)
MongoDB Enterprise Download (mongodb.com/download)
MongoDB Atlas database as a service for MongoDB (mongodb.com/cloud)

