MongoDB Architecture Guide

MongoDB 3.0

Table of Contents

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 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. Consider

- Developers are working with applications that create new, rapidly changing data types — structured, semi-structured, unstructured and polymorphic data — and massive volumes of it.
- 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.

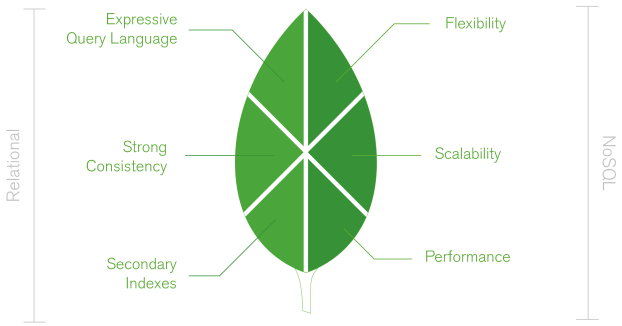

Figure 1: MongoDB Nexus Architecture, blending the best of relational and NoSQL technologies

- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally.
- 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.

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.** Users should be able to access and manipulate their data in sophisticated ways with powerful query, projection, aggregation and update operators, to support both operational and analytical applications.
- **Secondary indexes.** Indexes play a critical role in providing efficient access to data, for both reads and writes, 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, much more complicated to build applications around an eventually consistent model, imposing significant work on the developer, even for the most sophisticated development teams.

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.
- **Elastic Scalability.** NoSQL databases were all built with a focus on scalability, so they all include some form of sharding or partitioning, allowing the database to scale-out on commodity hardware deployed on-premise or in the cloud, allowing for almost unlimited growth.
- **High Performance.** NoSQL databases are designed to deliver great performance, measured in terms of both throughput and latency at any scale.

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 innovations of NoSQL while maintaining the foundation of relational databases.

MongoDB Embraces Modern Applications Through Key Innovations

MongoDB is the database for today's applications: innovative, fast time-to-market, globally scalable, reliable, and inexpensive to operate. With MongoDB, you can build applications that were never possible with traditional relational databases. Here's how.

- **Fast, Iterative Development.** Scope creep and changing business requirements no longer stand between you and successful project delivery. A flexible data model coupled with dynamic schema and idiomatic drivers make it fast for developers to build and evolve applications. Automated provisioning and management enable continuous integration and highly productive operations. Contrast this against static relational schemas and complex operations that have hindered you in the past.
- **Flexible Data Model.** MongoDB's document data model makes it easy for you to store and combine data of any structure, without giving up sophisticated data access and rich indexing functionality. You can dynamically modify the schema without downtime. You spend less time prepping your data for the database, and more time putting your data to work.
- **Multi-Datacenter Scalability.** MongoDB can be scaled within and across geographically distributed data centers, providing new levels of availability and scalability. As your deployments grow in terms of data volume and throughput, MongoDB scales easily with no downtime, and without changing your application. And as your availability and recovery goals evolve, MongoDB lets you adapt flexibly, across data centers, with tunable consistency.
- **Integrated Feature Set.** Analytics, text search, geospatial, in-memory performance and global replication allow you to deliver a wide variety of real-time applications on one technology, reliably and securely. RDBMS systems require additional, complex technologies demanding separate integration overhead and expense to do this well.
- **Lower TCO.** Application development teams are more productive when they use MongoDB. Single click management means operations teams are as well. MongoDB runs on commodity hardware, dramatically lowering costs. Finally, MongoDB offers affordable annual subscriptions, including 24x7x365 global support. Your applications can be one tenth the cost to deliver compared to using a relational database.
- **Long-Term Commitment.** MongoDB Inc and the MongoDB ecosystem stand behind the world's fastest-growing database. 10M+ downloads and 2,000+ customers including over one third of the Fortune 100. Over 1,000 partners and greater investor funding than any other database in history. You can be sure your investment is protected.

MongoDB Pluggable Storage Engines

MongoDB embraces two key trends in modern IT:

- Organizations are expanding the range of applications they deliver to support 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. Through the use of a pluggable storage architecture, MongoDB can be extended with new capabilities, and configured for optimal use of specific hardware architectures. This approach significantly reduces developer and operational complexity compared to running multiple databases to power applications with unique requirements. Users can leverage the same MongoDB query language, data model, scaling, security and operational tooling across different applications, each powered by different pluggable MongoDB storage engines.

MongoDB 3.0 ships with two supported storage engines, both of which can coexist within a single MongoDB replica set, making it easy to evaluate and migrate between them:

- 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.

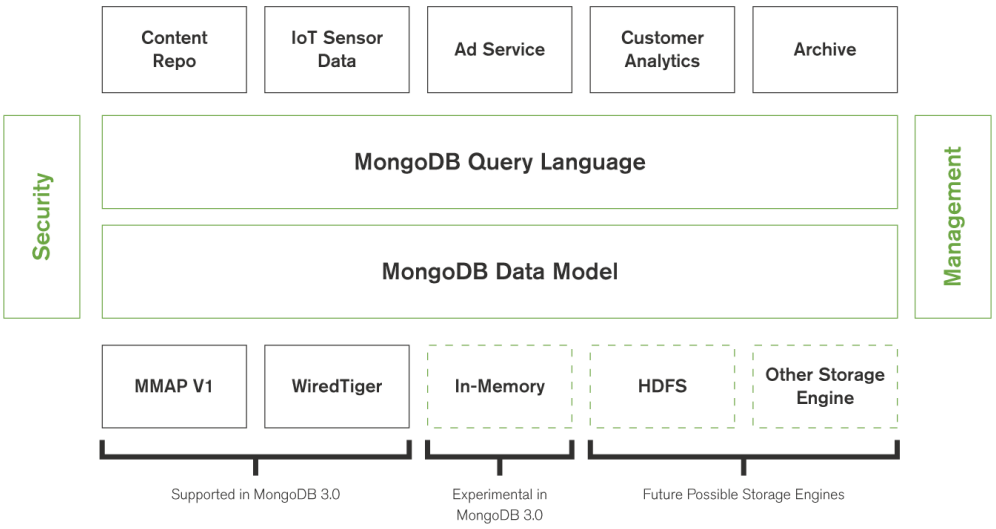

Figure 2: Pluggable storage engines, optimising MongoDB for unique application demands

Other engines are under development by MongoDB and members of the MongoDB ecosystem.

MongoDB Data Model

Data As Documents

MongoDB stores data as documents 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, and floating point. 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.

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.

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

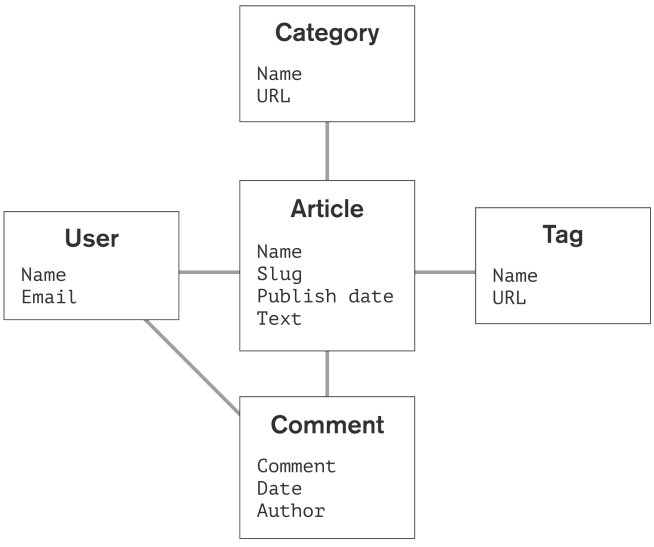

Figure 3: Example relational data model for a blogging application

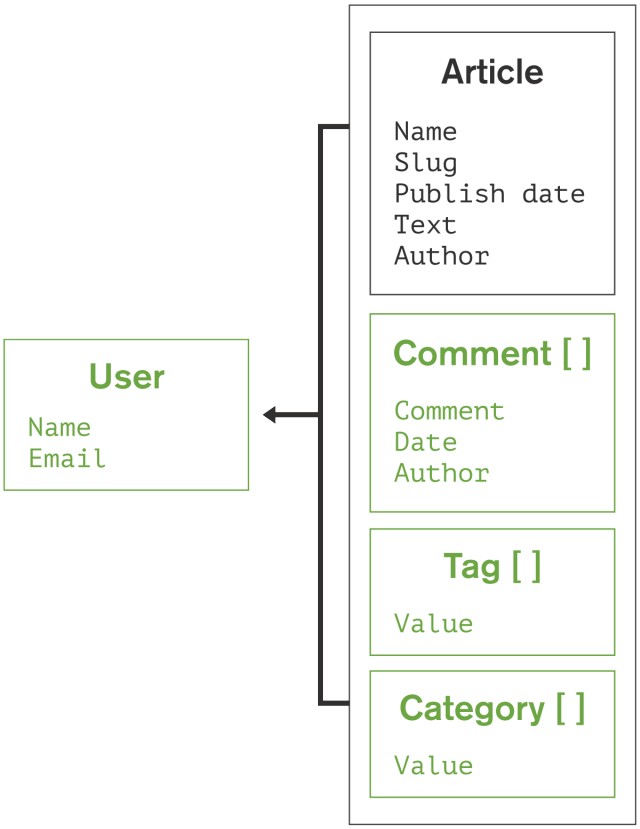

Figure 4: Data as documents: simpler for developers, faster for users.

the database can retrieve the entire document containing all related data.

In addition, MongoDB BSON documents are more 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

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 for performing costly ALTER_TABLE operations, or worse - having to re-design the schema from scratch.

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, 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, .NET, Ruby, PHP, JavaScript, node.js, Python, Perl, PHP, Scala and others. MongoDB drivers are designed to be idiomatic for the given language.

One fundamental difference as compared to 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.

Mongo Shell

The mongo shell is a rich, interactive JavaScript shell that is included with all MongoDB distributions. Nearly all commands supported by MongoDB can be issued through the shell, including administrative operations. The mongo shell is a popular way to interact with MongoDB for ad hoc operations. All examples in the MongoDB Manual are based on the shell. For more on the mongo shell, see the corresponding page in the MongoDB Manual.

Query Types

Unlike NoSQL databases, MongoDB is not limited to simple Key-Value operations. Developers can build rich applications using complex queries 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 against 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.
- **Text Search** queries return results in relevance order based on text arguments using Boolean operators (e.g., AND, OR, NOT).
- **Aggregation Framework** queries return aggregations of values returned by the query (e.g., count, min, max, average, similar to a SQL GROUP BY statement).
- **MapReduce queries** execute complex data processing that is expressed in JavaScript and executed across data in the database.

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.
- **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 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 pre-review and optimize plans using the powerful explain method and index filters.

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 to manage process and data distribution.

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-based Sharding.** Documents are partitioned across shards according to the shard key value. Documents with shard key values “close” to one another

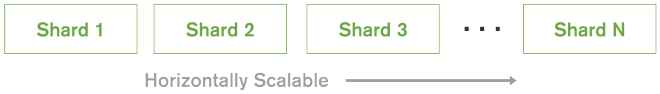

Figure 5: Automatic sharding provides horizontal scalability in MongoDB.

are likely to be co-located on the same shard. This approach is well suited for applications that need to optimize rangebased 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, but is less optimal for range-based queries.
- **Location-aware Sharding.** Documents are partitioned according to a user-specified configuration that associates shard key ranges with specific shards and hardware. Users can continuously refine the physical location of documents for application requirements such as locating data in specific data centers or on multi-temperature storage (i.e. SSDs for the most recent data, and HDDs for older data).

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 and aggregate and sort 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.

Consistency & Durability

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.

Concurrency Control

MongoDB enforces concurrency control for multiple clients accessing the database by coordinating multi-threaded access to shared data structures and objects. The granularity of concurrency control is dependent on the storage engine configured for MongoDB. WiredTiger enforces control at the document level while the MMAPv1

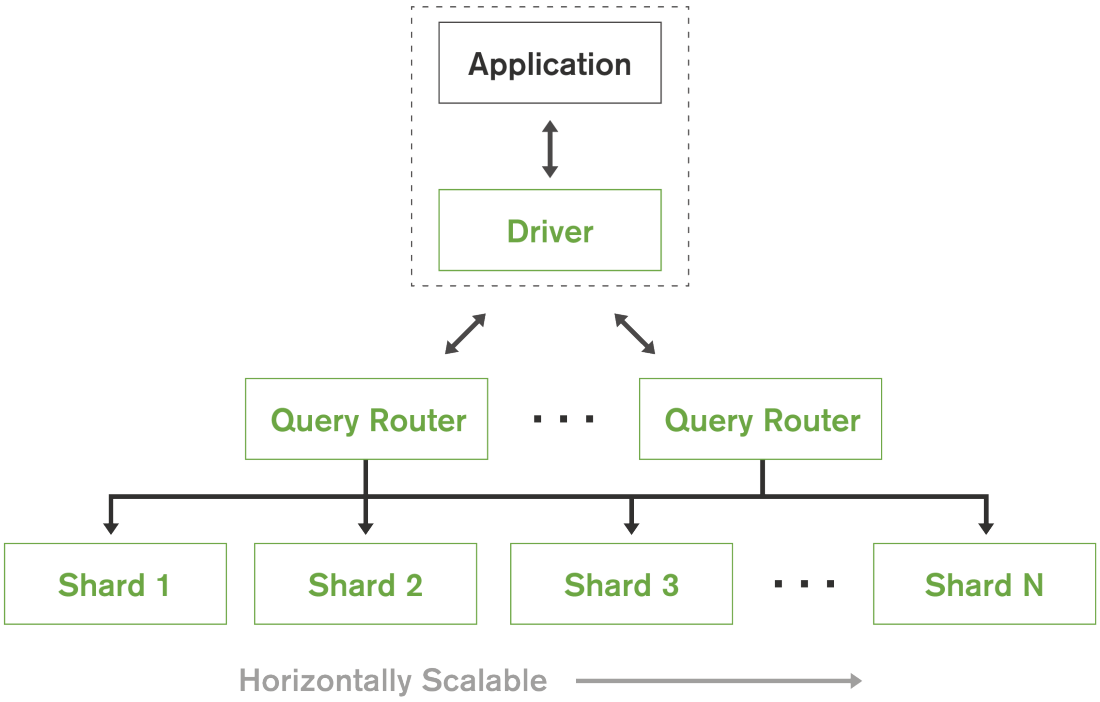

Figure 6: Sharding is transparent to applications.

storage engine implements collection-level concurrency control. For many applications, WiredTiger will provide benefits in greater hardware utilization and more predictable performance by supporting simultaneous write access to multiple documents in a collection from multiple sessions. Both storage engines support an unlimited number of simultaneous readers on a document.

Review the documentation for more information on concurrency control.

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 are issued at least as often as every 100ms by default. In addition to durability, the journal also prevents corruption in the event of an unclean system shutdown. 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 availability the “replica safe” write concern should be configured. An added feature of the WiredTiger storage engine is the ability to compress the journal on disk, 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.

Learn more about journaling from the documentation.

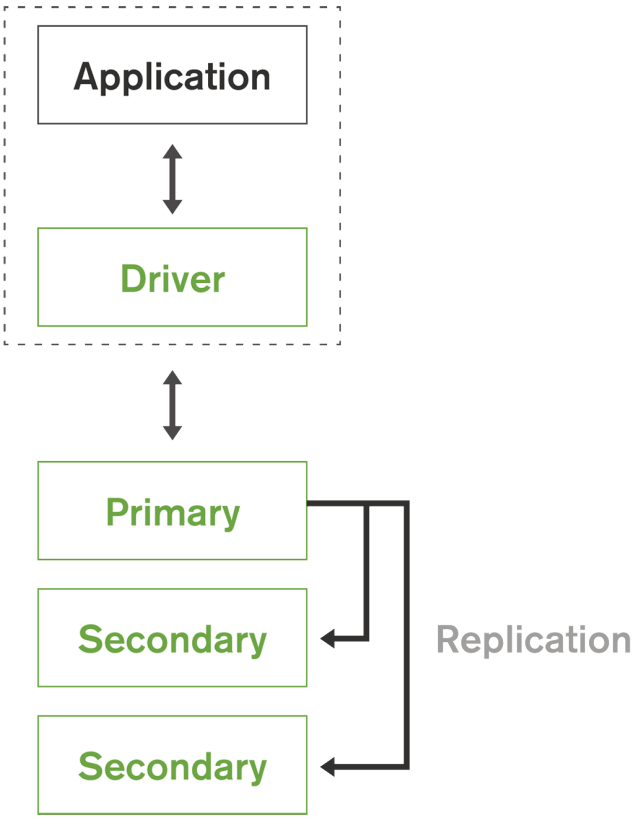

Figure 7: Self-Healing MongoDB Replica Sets for High Availability

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. 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 and begins to process all writes.

The number of replicas in a MongoDB replica set is configurable, and 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.

To further increase durability, administrators can take advantage of replica sets by configuring operations to write to multiple replica members before returning to the application – thereby providing functionality that is similar to synchronous replication.

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 applications. Applications can also read from the closest copy of the data as measured by ping distance when geographic latency is more important than consistency. For more on reading from secondaries see the entry on Read Preference.

In addition to increased resilience and broader data distribution, replica sets can also be configured with a members performing specialized tasks:

- **Hidden replica set members** can be provisioned to run applications such as analytics and reporting that require isolation from regular operational workloads.
- **Delayed replica set members** can be deployed to provide “historical” snapshots of data at different intervals in time for use in recovery from certain errors, such as “fat-finger” mistakes dropping databases or collections.

Replica sets also provide operational flexibility by providing a way to upgrade hardware and software without requiring the database to go 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. For more on replica sets, see the entry on Replication.

Replica Set Oplog

Operations that modify a database on the primary replica set member are replicated to the secondary members with

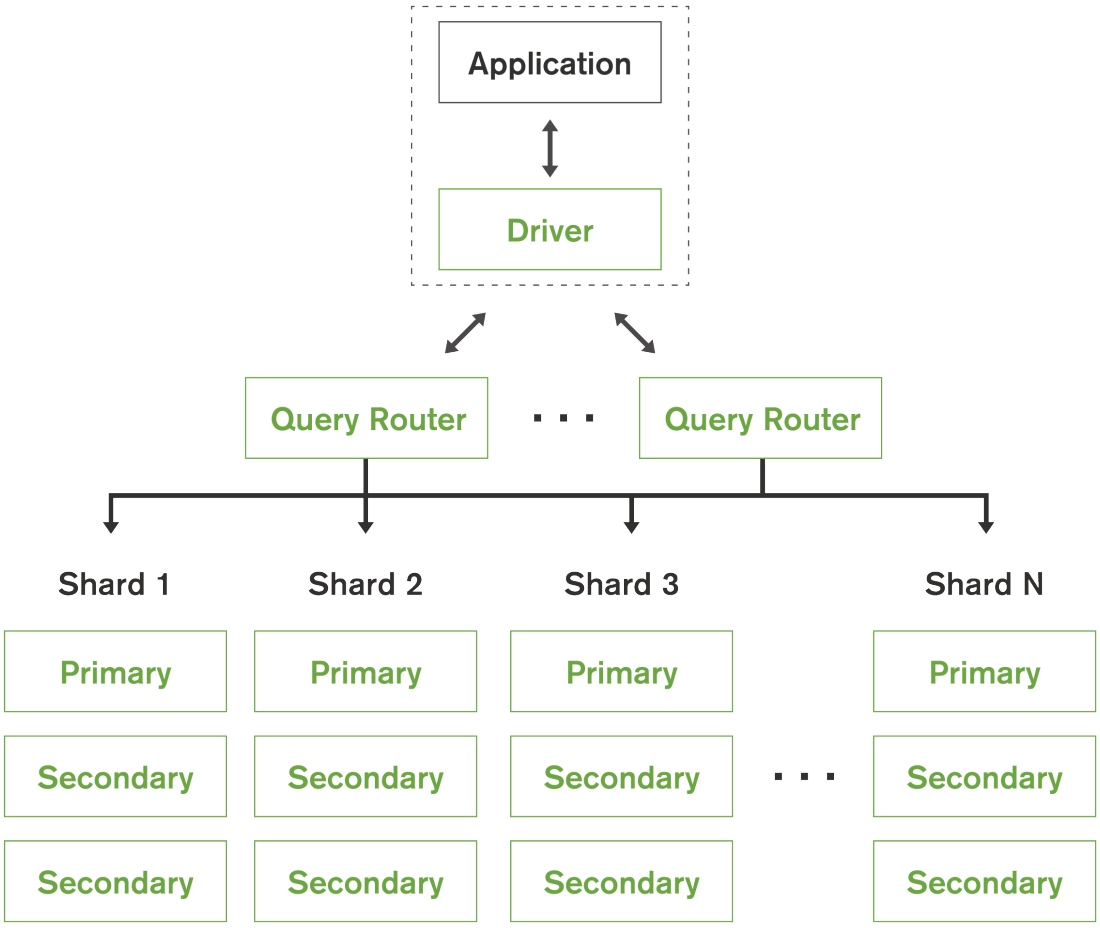

Figure 8: Sharding and replica sets; automatic sharding provides horizontal scalability; replica sets help prevent database downtime.

a log called the oplog. 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.

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 and their collections are copied from the primary or another replica to the secondary as well as the oplog, then the indexes are built. Initial synchronization is also performed when adding a new member to a replica set. 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 become the new primary in a process called an election. Once the new primary has been determined, remaining secondaries are configured 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 will configure itself to become 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"). 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. 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

MongoDB makes extensive use of RAM to speed up database operations. Reading data from memory is measured in nanoseconds, whereas reading data from spinning disk is measured in milliseconds; reading from memory is approximately 100,000 times faster than reading data from disk. While it is not required that all data fit in RAM, it should be the goal that indexes and all data that is frequently accessed is accommodated in RAM. For example it may be the case that a fraction of the entire database is most frequently accessed by the application, such as data related to recent events or popular products. If the volume of data that is frequently accessed exceeds the capacity of a single machine, MongoDB can scale horizontally across multiple servers using automatic sharding.

Because MongoDB provides in-memory performance, for most applications there is no need for a separate caching layer.

Learn more from the MongoDB Performance Best Practices whitepaper

Storage Efficiency with Compression

The WiredTiger storage engine supports 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.

By using the compression algorithms available in MongoDB, operations teams get higher performance per node and reduced storage costs.

Security

Data security and privacy is a critical concern in today's connected world. Data analyzed from new sources such as social media, logs, mobile devices and sensor networks has become as sensitive as traditional transactional data generated by back-office systems.

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

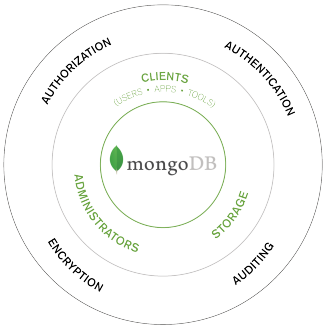

Figure 9: MongoDB security architecture: comprehensive protection of your most valuable data assets.

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. Additionally, field-level redaction can work with trusted middleware to manage access to individual fields within a document, making it possible to colocate data with multiple security levels in a single document.
- **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 and on disk. Support for SSL allows clients to connect to MongoDB over an encrypted channel.

To learn more, download the MongoDB Security Reference Architecture Whitepaper.

Managing MongoDB - Provisioning, Monitoring and Disaster Recovery

MongoDB 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 the MongoDB Cloud Manager service hosted in the cloud. Today, Cloud Manager supports thousands of deployments, including systems from one to hundreds of servers.

Ops Manager and Cloud Manager 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.

- **Deployment.** 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.

Deployments and Upgrades

Ops Manager (and Cloud 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. 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.

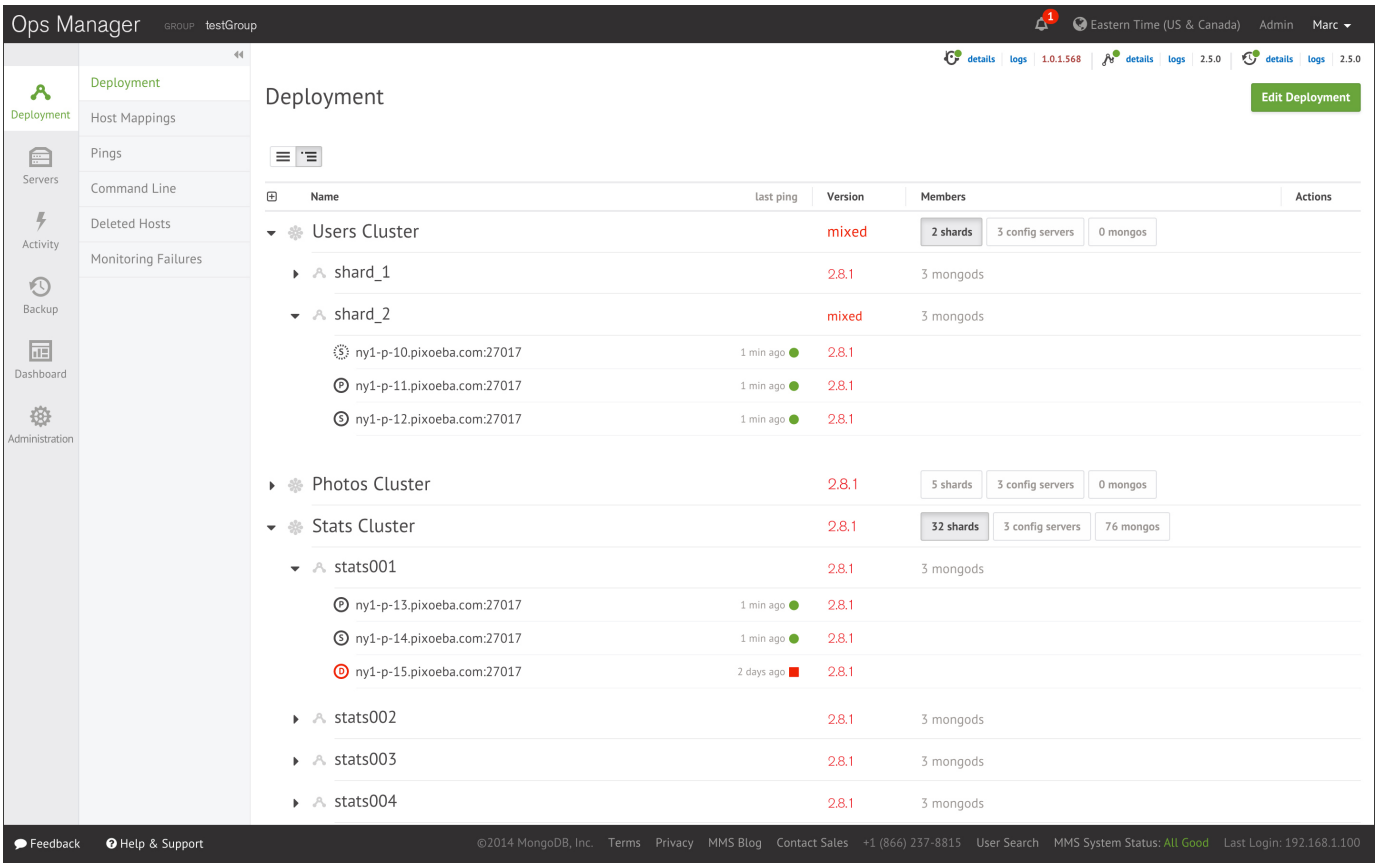

Figure 10: Ops Manager self-service portal: simple, intuitive and powerful. Deploy and upgrade entire clusters with a single click.

- 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 can deploy MongoDB on any connected server, but on AWS, Cloud Manager does even more. Once the AWS keys are provided, Cloud Manager can provision virtual machines on Amazon AWS at the time MongoDB is deployed. This integration removes a step and makes it even easier to get started. Cloud Manager provisions your AWS virtual machines with an optimal configuration for MongoDB.

In addition to initial deployment, Ops Manager and Cloud Manager 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.

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 and Cloud Manager 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.

Ops Manager and Cloud Manager 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 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.

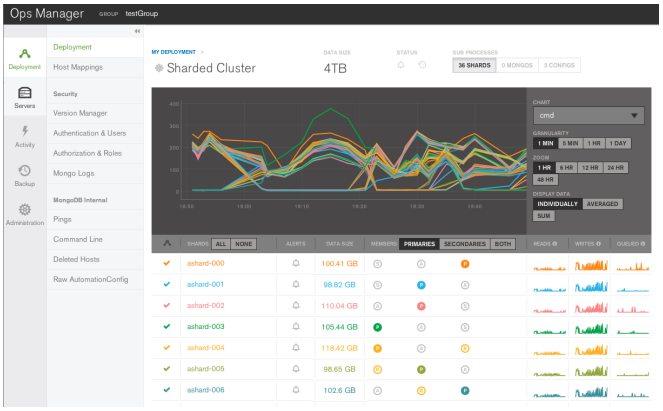

Figure 11: Ops Manager provides real time & historic visibility into the MongoDB deployment.

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. 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 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 Cloud Manager 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 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.

Integrating MongoDB with External Monitoring Solutions

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

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.

Running at 1/10th the Cost of Your Relational Database

MongoDB can be 1/10th the cost to build and run, compared to a relational database. The cost advantage is driven by:

- MongoDB's increased ease of use and developer flexibility, which reduces the cost of developing and operating an application;
- MongoDB's ability to scale on commodity server hardware and storage;
- MongoDB's substantially lower prices for commercial licensing, advanced features and support.

Furthermore, MongoDB's technical and cost-related benefits translate to topline advantages as well, such as faster time-to-market and time-to-scale.

To learn more, download our TCO comparison of Oracle and MongoDB.

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 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 Cloud Manager 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)

New York • Palo Alto • Washington, D.C. • London • Dublin • Barcelona • Sydney • Tel Aviv
US 866-237-8815 • INTL +1-650-440-4474 • info@mongodb.com