What's New in MongoDB 3.2

June 2016

mongoDB

Table of Contents

Introduction

IT teams are under intense pressure to differentiate from competitors by building new classes of applications able to fully exploit the wealth of new and untapped data generated by sensors, mobile, social, and cloud applications. However, the challenges associated with delivering on these needs are immense:

- IDC research states data volumes continue to grow at over 40% per annum. 90% of this data is now unstructured, rather than formated in the neat tabular structures most databases expect.
- The window to analyze and act on new data is shrinking – IBM estimates 60% of data loses its value within milliseconds of generation, and only 0.5% of all data is ever analyzed at all.
- Data governance has never been more important. The University of Texas, Austin calculated a 10% improvement in data usability at a Fortune 1000 company could increase revenues by $2bn per year.
- But as data increases in value, so too does the cost of protecting it. Based on research from PWC, the IT systems of organizations around the world were attacked over 117,000 times a day, every day, in 2014, representing a rise of nearly 50% over the previous year.

Technologies used in the past were never designed for the speed and scale demanded by modern applications. Architects and developers rely on new approaches to database management and analytics, but they must avoid the risks of exploding complexity that would come from using a myriad of niche technologies, each requiring its own tools and workflows.

MongoDB is designed to provide a solution to these challenges. With its Nexus Architecture, MongoDB is the only database that harnesses the innovations of NoSQL – scalability, performance, and data model flexibility – while maintaining the foundation of strong consistency, secondary indexes, and a rich query language that have made relational databases such an enduring technology over the past three decades.

MongoDB 3.2 is a giant step forward in enabling organizations to standardize on a single, modern database for mission-critical applications that could never be delivered on existing technologies. In-memory computing allows you to deliver high throughput at consistent latency for the most demanding workloads. End-to-end data encryption allows you to meet the most stringent regulatory standards. Document Validation gives you the benefits of a flexible data model without sacrificing governance controls for data quality.

To preserve existing investments and reign in TCO, MongoDB supports seamless integration with existing administrative skills, analytics platforms, and operational tooling. Analysts and data scientists can unlock smarter insights faster – from JOINs between different collections, to richer aggregations and search, to graphically exploring and visualizing multi-structured data sets using standard SQL-based BI and visualization platforms. By integrating MongoDB within their existing tool-sets and workflows, DBAs and Operations teams can industrialize new applications with less effort, time and cost, while gaining greater oversight and control over their entire IT estate.

In this whitepaper, you'll learn about what's new in MongoDB 3.2, and how to get started with this latest release.

New Use Cases Served by MongoDB

For developers building increasingly complex data-driven apps, there is no longer a "one size fits all" database storage technology that will perform optimally for every type of application required by the business. Modern applications need to support a variety of services with different access patterns, security requirements and price/performance profiles – from high throughput in-memory operations, to real-time analytics, to managing highly sensitive data.

MongoDB 3.0 introduced a new flexible storage architecture, making it fast and easy for MongoDB and the ecosystem to build new pluggable storage engines that allow the database to be extended with new capabilities, and to be configured for specific workload requirements. Moving beyond the two original storage engines supported with the 3.0 release, MongoDB 3.2 now adds two new options to the mix. The supported storage engines comprise:

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

MongoDB uniquely allows users to mix and match multiple storage engines within a single MongoDB cluster. 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. This approach significantly reduces developer and operational complexity when compared to running multiple distinct database technologies. 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.

New Default MongoDB Storage Engine: WiredTiger

MongoDB 3.2 now uses WiredTiger as its default storage engine. When compared to the original MMAP storage engine used in earlier MongoDB releases, WiredTiger's more granular concurrency control and native compression improve performance by 7-10x, while reducing storage overhead by up to 80%. WiredTiger is ideal for a wide

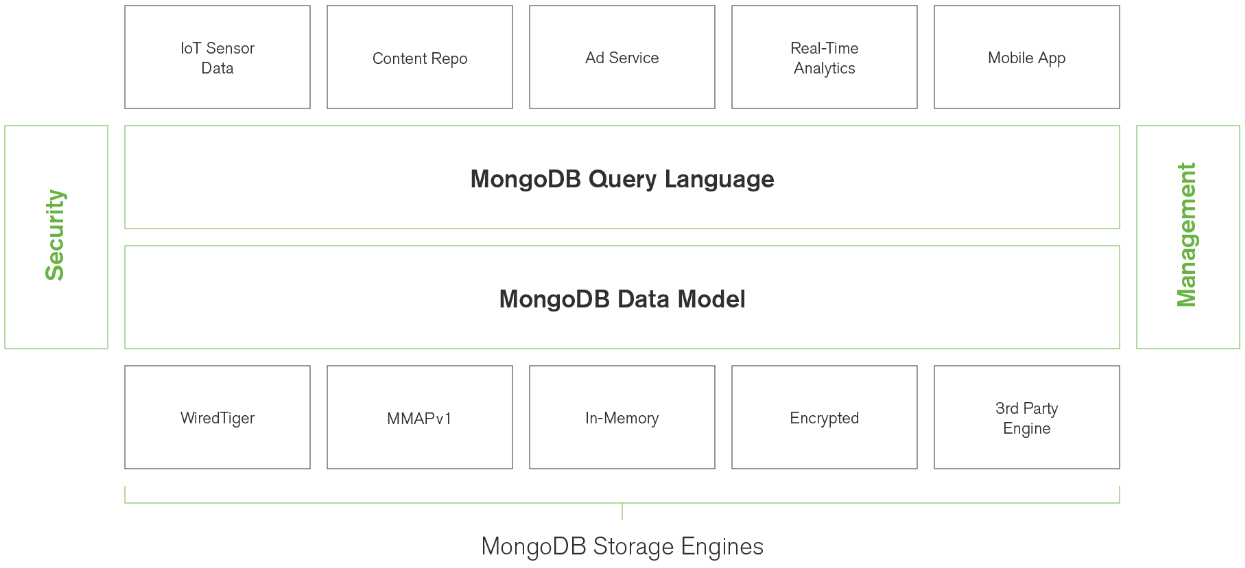


Figure 1: Mix and match storage engines within a single MongoDB replica set

range of operational applications, and is therefore the default storage engine.

New MongoDB Encrypted Storage Engine

The frequency and severity of data breaches continues to escalate year on year. Research from PWC identified over 117,000 attacks against information systems every day in 2014, representing an increase of 48% over the previous year. Companies reporting losses from security breaches totaling $20m or more doubled over the same period. Updated European Union General Data Protection regulations will increase penalties on those organizations proven to have not taken sufficient measures to protect their customers' data against unauthorized disclosure. With databases storing an organization's most important information assets, securing them is top of mind for administrators.

With advanced authentication, authorization, auditing and network encryption security controls, MongoDB is widely used in regulated industries such as finance, retail, healthcare, education and government. However, protecting data stored "at-rest" on persistent storage required encryption to be implemented either at the application level, or via external filesystem and disk encryption solutions. By introducing additional technology into the stack, both of these approaches can add cost and complexity.

With the introduction of the Encrypted storage engine, protection of data at-rest now becomes an integral feature of the database. By natively encrypting database files on disk, administrators eliminate both the management and performance overhead of external encryption mechanisms.

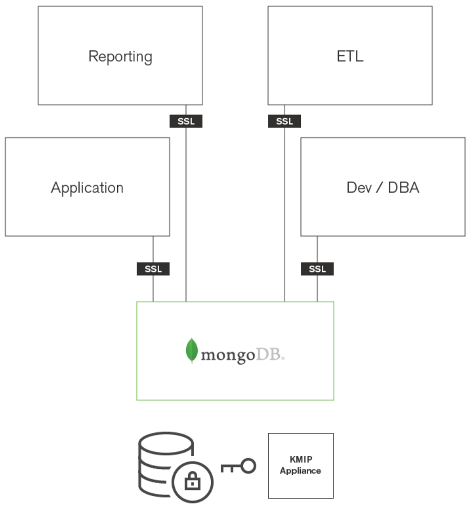


Figure 2: End to End Encryption – Data In-Flight and Data At-Rest

This new storage engine provides an additional level of defense, allowing only those staff with the appropriate database credentials access to encrypted data.

Using the Encrypted storage engine, the raw database "plaintext" content is encrypted using an algorithm that takes a random encryption key as input and generates ciphertext that can only be read if decrypted with the decryption key. The process is entirely transparent to the application. MongoDB supports a variety of encryption schema, with AES-256 (256 bit encryption) in CBC mode being the default. AES-256 in GCM mode is also supported. The encryption schema can be configured for FIPS 140-2 compliance.

The storage engine encrypts each database with a separate key. The key-wrapping scheme in MongoDB wraps all of the individual internal database keys with one external master key for each server. The Encrypted storage engine supports two key management options – in both cases, the only key being managed outside of MongoDB is the master key:

- Local key management via a keyfile.
- Integration with a third party key management appliance via the KMIP protocol (recommended).

Most regulatory requirements mandate that the encryption keys must be rotated and replaced with a new key at least once annually. MongoDB can achieve key rotation without incurring downtime by performing rolling restarts of the replica set. When using a KMIP appliance, the database files themselves do not need to be re-encrypted, thereby avoiding the significant performance overhead imposed by key rotation in other databases. Only the master key is rotated, and the internal database keystore is re-encrypted.

The Encrypted storage engine is based on WiredTiger, and so is designed for operational efficiency and performance:

- Document level concurrency control and compression.
- Support for Intel's AES-NI equipped CPUs for acceleration of the encryption/decryption process.
- As documents are modified, only updated storage blocks need to be encrypted, rather than the entire database.

Based on user testing, the Encrypted storage engine minimizes performance overhead to around 15% (this can vary, based on data types being encrypted), which can be much less than the observed overhead imposed by some filesystem encryption solutions.

The Encrypted storage engine is available as part of MongoDB Enterprise Advanced. Refer to the documentation to learn more, and see a tutorial on how to configure the storage engine.

Flexible In-Memory Computing with MongoDB

The advantages of in-memory computing are well understood. Data can be accessed in RAM nearly 100,000 times faster than retrieving it from disk, delivering orders-of-magnitude higher performance for the most demanding applications. Amazon famously published research showing that every 1/10th additional second in latency resulted in a 1% loss in sales. IBM estimates that 60% of data starts to lose its value within milliseconds of being generated. Examples of applications benefiting from in-memory computing include real-time re-scoring of personalized product recommendations as users are browsing a site, or trading stocks in immediate response to market events. Critical for modern applications, in-memory computing enables data access and analytics at speeds never before possible.

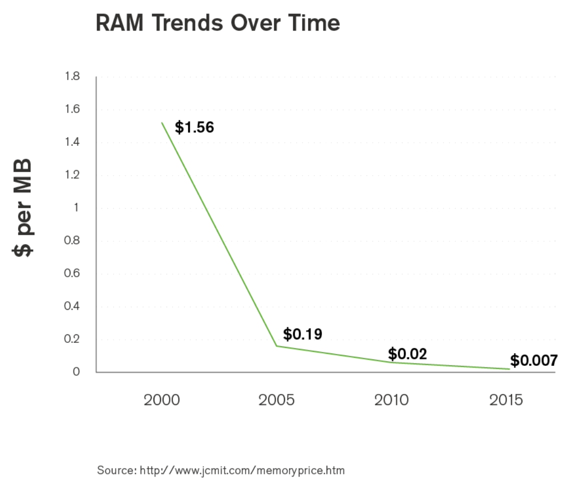


Figure 3: Reductions in RAM prices make In-Memory computing economically viable

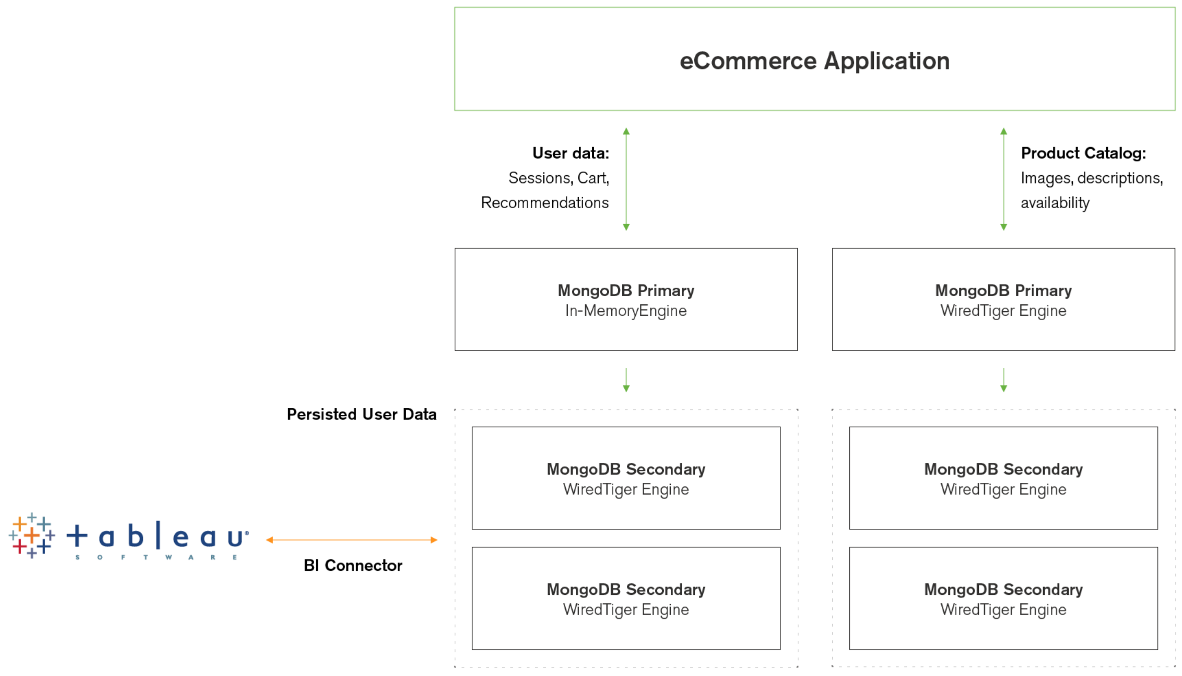


Figure 4: Using MongoDB pluggable storage engines allows a single database to power multiple applications

Despite these benefits, the adoption of in-memory computing has been constrained by high memory costs. However, with RAM prices continuing to tumble, the promised performance gains can now be realized with an acceptable ROI. Even with growing demand from the consumer sector, memory prices have declined over 220x since the start of the century, and nearly 3x in the past five years alone.

With the addition of the new In-Memory engine based on WiredTiger, MongoDB users can now realize the performance advantages of in-memory computing, without trading away the rich query flexibility, real-time analytics, scalable capacity, or durability guarantees offered by conventional disk-based databases. The In-Memory storage engine delivers the extreme throughput and predictable latency required by the most demanding applications in AdTech, finance, telecoms, e-commerce, and more.

The benefits of storage engine flexibility extend beyond the boundaries of a single application. Unlike monolithic code bases of the past, modern applications typically comprise multiple services, each can have its own unique data access patterns and performance profiles. MongoDB's storage architecture allows users to optimize for the requirements of each service. As illustrated by the e-commerce example in Figure 4, user data is managed by the In-Memory engine to provide the throughput and bounded latency essential for great customer experience. However, the product catalog's data storage requirements exceed server memory capacity, so is provisioned to another MongoDB replica set configured with the disk-based WiredTiger storage engine.

In this example, MongoDB's flexible storage architecture means developers are freed from the complexity of having to use different in-memory and disk-based databases to support the e-commerce application. Administrators are freed from the complexity of having to configure and manage separate data layers. Instead, the application uses the same MongoDB database with each service powered by the storage engine best optimized for the use case.

The In-Memory storage engine is part of MongoDB Enterprise Advanced.

Mission-Critical Deployments

MongoDB 3.2 delivers major advances in the critical areas of governance, high availability, and disaster recovery.

Ensuring data quality and maintaining continuous service availability is essential as more mission-critical applications are deployed to MongoDB.

Document Validation: Data Governance for Dynamic Schema

Dynamic schemas bring great agility, but it is also important that controls can be implemented to maintain data quality, especially if the database is powering multiple applications, or is integrated into a larger data management platform that feeds into upstream and downstream systems. Rather than delegating 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.

Validating Documents in MongoDB 3.2

There is significant flexibility to customize which parts of the documents are **and are not** validated for any collection. For any key it might be appropriate to check:

- That it exists
- If it does exist, that the value is of the correct type
- That the value is in a particular format (regular expressions can be used to check if the contents of the string matches a particular pattern – that it's a properly formatted email address, for example)
- That the value falls within a given range

As an example, the following snippet adds a rule to the `contacts` collection that validates:

- The year of birth is no later than 1994
- The document contains a phone number and/or an email address
- When present, the phone number and email addresses are strings

```
db.runCommand({
    collMod: "contacts",
    validator: {
        $and: [
            {year_of_birth: {$lte: 1994}},
            {$or: [
                {phone: { $type: "string" }},
                {email: { $type: "string" }}
    ]}]
}})
```

Adding the validation checks to a collection is very intuitive to any developer or DBA familiar with MongoDB, as Document Validation uses the standard MongoDB Query Language.

In summary, Document Validation delivered in MongoDB 3.2 allows users to enforce governance across MongoDB data, while maintaining the agility benefits of a dynamic schema. For a more in-depth discussion of this feature, including code samples and a tutorial, please see the Document Validation blog post or the documentation.

Enhanced Replication Protocol: Fast Failover and Stricter Durability Guarantees

In a distributed system like MongoDB, it should be assumed that individual nodes can and will fail. MongoDB's priorities in the event of a node failing are to:

- Ensure data consistency
- Resume full service as quickly as possible to maximize availability

MongoDB 3.2 introduces an enhanced replication protocol that delivers faster service recovery in the event of a primary failure, as well as stricter durability guarantees. The enhanced replication protocol extends the Raft consensus algorithm to offer greater deployment flexibility while maintaining compatibility with replication constructs offered in earlier MongoDB releases. Specifically, the protocol maintains support for replica set arbiters, replica set member election priorities and secondary members replicating from other secondaries to enable chained replication.

The enhanced replication protocol reduces the failover interval by optimizing the algorithms used to detect replica set primary failures and elect a new primary. Failover time is dependent on several factors, including network latency. It is important for the system to avoid unnecessary failovers, and to provide flexibility for the needs of different deployments. MongoDB 3.2 introduces a new parameter called `electionTimeoutMillis` to allow users to configure their systems for optimal failover behavior:

- Higher values result in slower failovers but decreased sensitivity to network latency and load on the primary node
- Lower values result in faster failover, but increased sensitivity to network latency and load on the primary.

`electionTimeoutMillis` defaults to 10,000 milliseconds (10 seconds).

Review the documentation for more information.

Durability Guarantees

In addition to faster failover, the enhanced protocol offers stricter consistency and durability controls for write operations across replica sets. With a write concern configured to apply writes to one or more secondaries before acknowledging the operation, the enhanced protocol will now commit operations to both the memory and the journal on those secondary members before reporting successful completion to the application. Additionally, when using the `majority` write concern, the change will also be committed to the memory and journal on the primary before the acknowledgment is made.

By default, any replica set upgraded from a pre-MongoDB 3.2 release will use the previous replication protocol, while new replica sets will use the enhanced protocol. It is possible to manually switch to the old or new protocol by setting `protocolVersion` to 0 or 1 respectively.

Simplified Sharded Cluster Management

MongoDB scales out databases on commodity hardware using a technique called sharding. The config servers are a critical component in a sharded cluster, storing the metadata used by the mongos query router to direct read and write operations to the appropriate shards, thereby abstracting complexity from applications accessing the database.

Prior to MongoDB 3.2, the config servers were implemented as three special-purpose mongod instances using their own write protocols, coordinators, and consistency checking. This architecture complicated the management of sharded clusters, and increased latency when deploying MongoDB across more than three data centers.

Starting with MongoDB 3.2, the config servers will, by default, be deployed as a MongoDB replica set. This change improves metadata consistency and manageability as the config servers can now take advantage of the standard replica set read and write protocols. Furthermore, config server replica sets can now span more than three data centers with up to 50 replica set members supported, providing higher levels of availability and lower cross-region latency. Review the documentation to learn more.

New Users

With MongoDB deployed across a wider range of an organization's application portfolio, data analysts, DBAs, and operations teams will need to integrate MongoDB within their existing processes and tool sets. MongoDB 3.2 allows analysts to support the business with new insights from untapped data sources, while DBAs and Ops teams are able to operationalize MongoDB alongside existing relational databases, protecting existing investments in management platforms and skillsets.

Data Analysts and Scientists

MongoDB Connector for BI

Driven by growing requirements for self-service analytics, faster discovery and prediction based on real-time operational data, and the need to integrate multi-structured and streaming data sets, BI and analytics platforms are one of the fastest growing software markets.

To address these requirements, modern application data stored in MongoDB can for the first time be easily explored with industry-standard SQL-based BI and analytics platforms. Using the MongoDB Connector for BI, analysts, data scientists and business users can now seamlessly

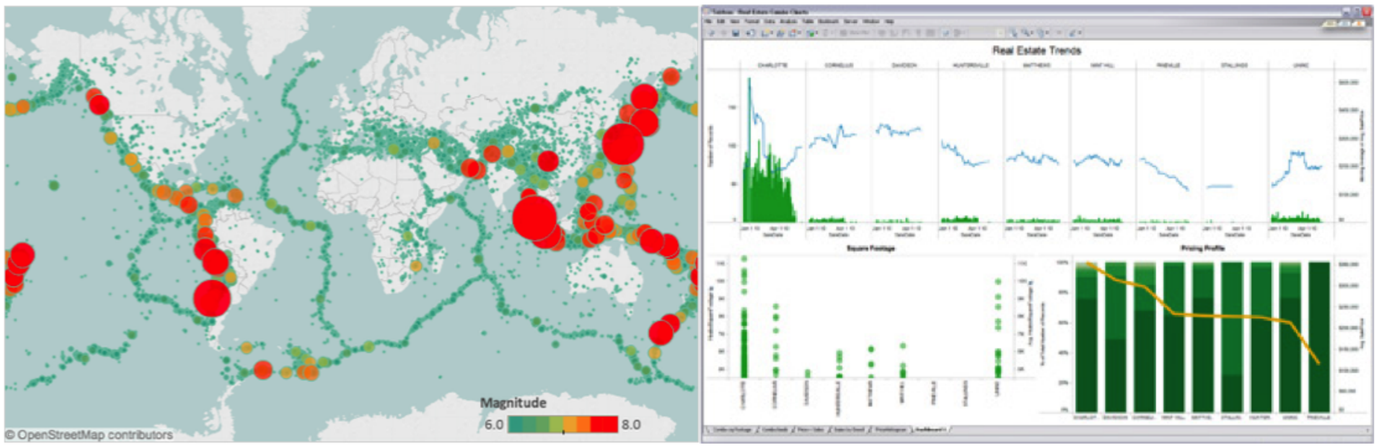


Figure 5: Uncover new insights with powerful visualizations generated from MongoDB

visualize semi-structured and unstructured data managed in MongoDB, alongside traditional data in their SQL databases, using the same BI tools deployed within millions of enterprises.

MongoDB Connector for BI Implementation

SQL-based BI tools expect to connect to a data source with a fixed schema presenting tabular data. This presents a challenge when working with MongoDB's dynamic schema and rich, multi-dimensional documents. In order for BI tools to query MongoDB as a data source, the BI Connector does the following:

- Provides the BI tool with the schema of the MongoDB collection to be visualized. Users can review the schema output to ensure data types, sub-documents and arrays are correctly represented
- Translates SQL statements issued by the BI tool into equivalent MongoDB queries that are then sent to MongoDB for processing
- Converts the returned results into the tabular format expected by the BI tool, which can then visualize the data based on user requirements

The BI Connector is available with MongoDB Enterprise Advanced. Watch the demo to see the BI Connector in action, and review the documentation to learn more. You can also download the BI Connector for evaluation

Dynamic Lookup: Bringing Left Outer JOINs to MongoDB

Applications get great efficiency from MongoDB by combining data that is accessed together into a single document. In contrast, a typical relational database schema scatters related data across scores of tables – e.g., a blog site that stores every tag, category, comment, author and callback as rows in separate tables from the blog post they're associated with.

Typically it is most advantageous to take a denormalized data modeling approach for operational databases – the efficiency of reading or writing an entire record in a single operation outweighing any modest increase in storage requirements. However, there are examples where normalizing data can be beneficial, especially when data from multiple sources needs to be blended for analysis. Consider a shopping cart, which presents two options for handling the order and product information:

- **Include all data for an order in the same document**
    - Fast reads – one `find` delivers all the required data
    - The order document contains the product details that were correct at the time the order was placed; the price of that product may change in the future but the order document remains an accurate representation of the order
    - Consumes additional storage – the details of each product are stored in many order documents; this has become less of an issue as memory and storage prices have fallen

- **Order document references product documents**
  - Space efficient – product details are stored just once
  - Slower reads – multiple trips to the database
  - If an attribute of the product (such as the unit price) changes in the future, any older order documents are then incorrect as they reference this newer version
  - Extra application logic – an application must iterate over product IDs in the order document and then fetch the product documents

MongoDB 3.2 introduces the ability to combine data from multiple collections by implementing left outer joins through the `$lookup` operator, which can now be included as a stage in a MongoDB Aggregation Framework pipeline. The new `$lookup` stage provides more flexibility in data modeling, and allows richer analytics to be run with higher performance and less application-side code.

You can learn more from the documentation and see worked examples in this series of blog posts.

Real-Time Analytics and Search

With the emergence of new data sources such as social media, mobile applications, and sensor-equipped Internet of Things networks, organization can extend analytics to deliver real-time insight and discovery into such areas as operational performance, customer satisfaction, and competitor behavior. However, time to value is everything. For example, having access to real-time customer sentiment or fleet tracking is of little benefit unless the data can be analyzed and reported in real time.

MongoDB 3.2 extends the options for performing analytics on the live, operational database – ensuring that answers are delivered quickly and simply, and are based on current data. Work that would previously have needed to be implemented in client code can now be performed by the database – freeing the developer to focus on building new features.

Improved Aggregation

As the size of operational databases grow, efforts to analyze and derive insights from the data becomes increasingly important, as well as complex.

The aggregation pipeline is a powerful way to perform complex analytical queries on MongoDB data. Aggregation pipeline stages allow manipulation of a "stream" of documents from a collection that can either be returned via a cursor to the client (similar to `find`), or be stored in a new collection via a final `$out` stage.

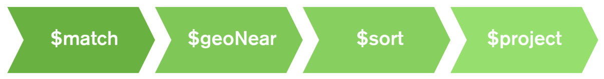


Figure 6: Example stages in an aggregation pipeline

When analyzing very large data sets, it is frequently sufficient to look at a random sample of documents rather than all of the data. For example, if you wanted to compare the number of check-ins to coffee shops to those at bars, you can get a good approximation without searching through every single check-in. Previously, this sampling would have to have been implemented in the application, but MongoDB introduces the `$sample` operator, which can be included at any point in the aggregation pipeline.

MongoDB documents can store arrays as well as simple values. While this feature is very expressive and powerful, without the corresponding ability to manipulate and filter arrays in documents during aggregations, their usefulness has been limited in an analytical context. New operators have been added to allow more flexibility when dealing with arrays: `$slice`, `$arrayElemAt`, `$concatArrays`, `$isArray`, `$filter`, and `$min`.

New mathematical operators have been added for operations such as truncate, ceiling, floor, absolute, rounding, square root, logarithms and standard deviations. These operators can be used to move code from the client tier directly into the database, allowing higher performance with lower developer complexity.

By combining the new and existing operators aggregation pipelines can be built to generate sophisticated results with a single query. Review the documentation to learn more about all of the aggregation pipeline enhancements.

Improved Text Search

Text searches on the data in MongoDB can either be performed in the database or by an external search engine.

Performing the search within the database is more efficient and simpler to administer, so that is the preferred option whenever possible.

MongoDB 3.2 increases the set of use cases that can be met with in-database text searches by adding support for case-sensitive searches, as well as additional languages including Arabic, Farsi, Urdu, Simplified Chinese, and Traditional Chinese. To provide support for these languages, MongoDB Enterprise Advanced provides integration with Basis Technology Rosette Linguistics Platform (RLP) to perform normalization, word breaking, sentence breaking, and stemming or tokenization depending on the language.

More information on text search enhancements can be found in the documentation

DBAs: MongoDB Compass

MongoDB's dynamic schema and rich document model make developers more productive, but they also make it difficult to explore and understand the underlying data and its structure – in particular for non-developers who aren't familiar with the MongoDB query language.

The MongoDB Compass GUI allows users to understand the structure of data in the database and perform ad hoc queries against it – all with zero knowledge of MongoDB's query language. Typical users could include architects building a new MongoDB project or a DBA who has inherited a database from an engineering team, and who must now maintain it in production. They need to understand what kind of data is present, define what indexes might be appropriate, and identify if Document Validation rules should be added to enforce a consistent document structure.

Until now, users wishing to understand the shape of their data would have to connect to the MongoDB shell and write queries to reverse engineer the document structure, field names and data types. Similarly, anyone wanting to run custom queries on the data would need to understand MongoDB's query language.

MongoDB Compass provides users with a graphical view of their MongoDB schema by sampling a subset of

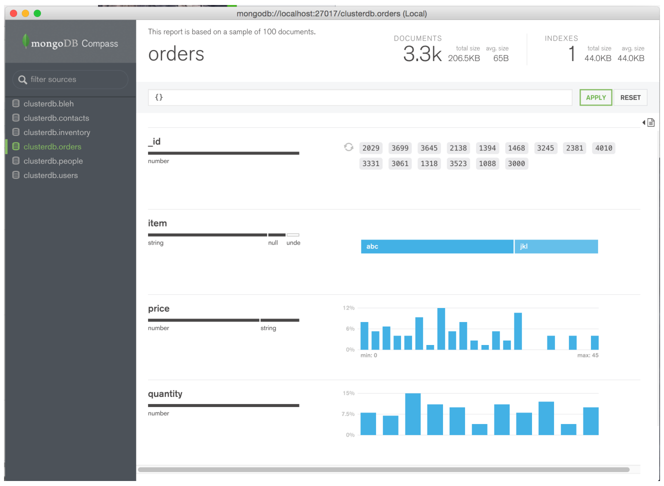


Figure 7: Document structure and contents exposed by MongoDB Compass

documents from a collection. By using sampling, MongoDB Compass minimizes database overhead and can present results to the user almost instantly.

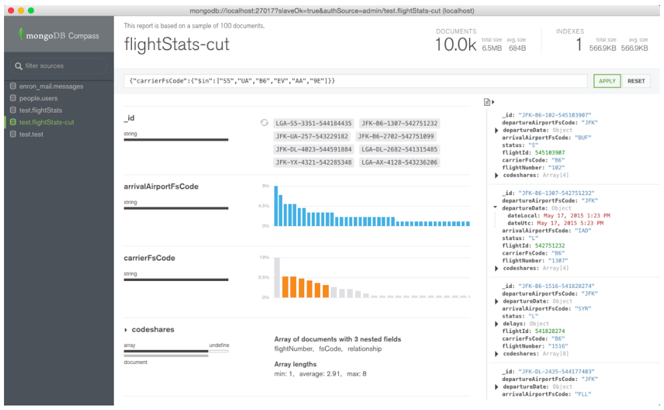


Figure 8: Interactively build and execute database queries with MongoDB Compass

Querying Data

As illustrated in Figure 8, a query can be built and executed by simply selecting document elements from the MongoDB Compass user interface. By selecting multiple values, sophisticated queries can be built. The query can then be executed at the push of a button and the results viewed both graphically and as a set of JSON documents.

MongoDB Compass samples the database to provide a fast, interactive experience no matter how large the database. If the full results are needed then the query can be simply copied and pasted into a MongoDB shell window.

MongoDB Compass is included with MongoDB Professional and MongoDB Enterprise Advanced. You can learn more about Compass in the documentation, and see it in action in our short 3-part demo series:

- Part 1: Introduction to Compass
- Part 2: Schema visualization
- Part 3: Visual query building

To evaluate MongoDB Compass, head to the MongoDB Download Center.

Operations Teams

For those needing full control, MongoDB Ops Manager and Cloud Manager are the most powerful way to run MongoDB, reducing tasks such as deployment, scaling, upgrades and backups to just a few clicks or an API call. Operations teams can be 10-20x more productive using the Ops or Cloud Manager platforms. For those seeking even greater simplification and automation, consider MongoDB Atlas – the database as a service for MongoDB - described later in this paper.

With the enhancements to both Ops and Cloud Manager in MongoDB 3.2, administrators can:

- Integrate MongoDB alongside existing Application Performance Monitoring platforms for global health visibility over the entire IT estate, all from a single pane of glass
- Drill down into any MongoDB-specific issues using Ops Manager's granular monitoring of database telemetry, including new query profiler visualizations
- Use Ops Manager automation to initiate zero-downtime maintenance and upgrade activities, such as rolling out new indexes across a sharded cluster
- Create point-in-time, consistent snapshots of the database on standard network-mountable filesystems, and restore complete running MongoDB clusters from backup files.

APM Integration: New Relic & AppDynamics

Many operations teams use Application Performance Monitoring (APM) platforms such as New Relic and AppDynamics to gain global oversight of their complete IT infrastructure from a single management UI. Issues that risk affecting customer experience can be quickly identified and isolated to specific components – whether attributable to devices, hardware infrastructure, networks, APIs, application code, databases and more.

The MongoDB drivers have been enhanced with a new API that exposes query performance metrics to APM tools. Administrators can monitor time spent on each operation, and identify slow running queries that require further analysis and optimization.

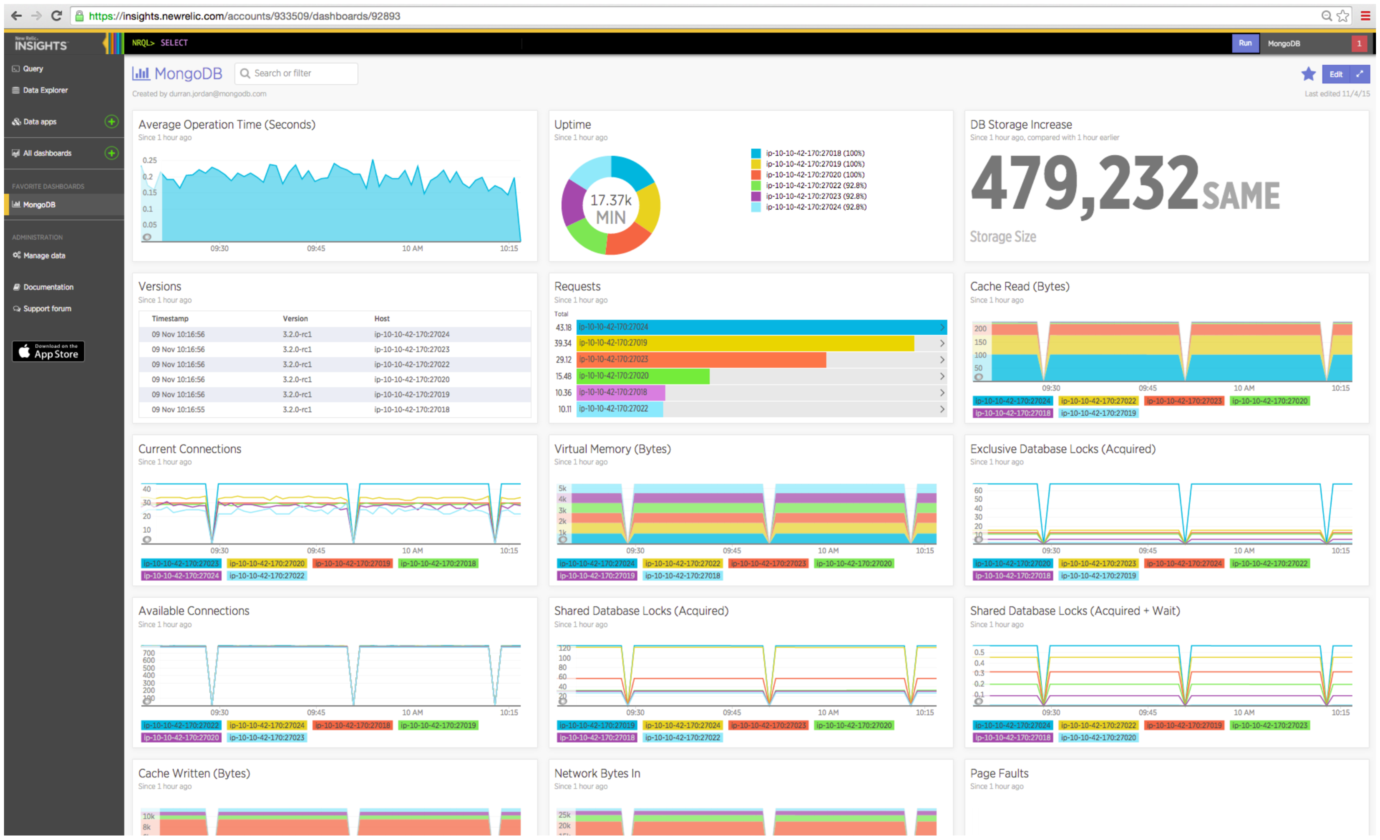
In addition, MongoDB Atlas and Cloud Manager provide packaged integration with the New Relic platform. Key metrics from MongoDB Atlas or Cloud Manager are accessible to the APM for visualization, enabling MongoDB health to be monitored and correlated with the rest of the application estate.

As shown in Figure 9, summary metrics familiar to Cloud Manager users are presented within the APM's UI. Administrators can also run New Relic Insights for analytics against monitoring data to generate dashboards that provide real-time tracking of Key Performance Indicators (KPIs).

If the operations team needs finer grained telemetry, they can drill down into the 100+ system metrics maintained by Cloud Manager. For example, the new visual query profiler helps diagnose slow running queries, which can then be resolved by adding a new index and automatically deploying that across every node in the cluster.

Query Performance Visualization: Enabling Fast and Simple Query Optimization

The MongoDB database profiler collects fine-grained information that can be used to analyze query performance. However, the output could be difficult to parse, making slow running queries difficult to correct. In addition, the profiler had to be individually activated for each MongoDB instance, and the output manually aggregated from every

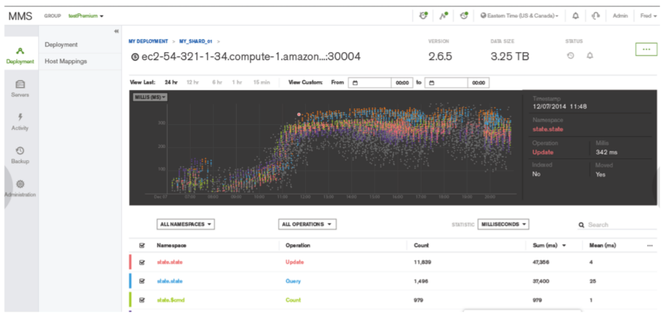


Versions
Since 1 hour ago

Timestamp	Version	Host
09 Nov 10:16:56	3.2.0-rc1	ip-10-10-42-170:27024
09 Nov 10:16:56	3.2.0-rc1	ip-10-10-42-170:27023
09 Nov 10:16:56	3.2.0-rc1	ip-10-10-42-170:27022
09 Nov 10:16:56	3.2.0-rc1	ip-10-10-42-170:27020
09 Nov 10:16:56	3.2.0-rc1	ip-10-10-42-170:27019
09 Nov 10:16:55	3.2.0-rc1	ip-10-10-42-170:27018

Figure 9: MongoDB integrated into a single view of application performance

node to provide a holistic view across the entire deployment.



Namespace	Operation	Count	Sum (ms)	Mean (ms)
state.state	Update	11,839	47,356	4
state.state	Query	1,496	37,400	25
state.$cmd	Count	979	979	1

Figure 10: Visual Query Profiling in MongoDB Ops Manager

Delivered as part of Ops Manager and Cloud Manager, the new Visual Query Profiler provides a quick and convenient way for operations teams and DBAs to analyze specific queries or query families. The Visual Query Profiler (as shown in Figure 10) displays how query and write latency varies over time – making it simple to identify slower queries with common access patterns and characteristics, as well as identify any latency spikes. A single click in the Ops Manager UI activates the profiler, which then consolidates and displays metrics from every node in a single screen.

Index Suggestions & Automated Index Builds

Further simplifying operations, the visual query profiler will analyze data it collects to provide recommendations for new indexes that can be created to improve query performance.

Once identified, these new indexes need to be rolled out in the production system. In order to minimize the impact to the live system, the best practice is to perform a rolling index build – starting with each of the secondaries and finally applying changes to the original primary, after swapping its role with one of the secondaries. While this rolling process can be performed manually, Ops Manager and Cloud Manager can now automate the process across MongoDB replica sets, reducing operational overhead and the risk of failovers caused by incorrectly sequencing management processes.

New Indexing Option: Partial Indexes

Secondary indexes are one of the ways that MongoDB distinguishes itself from NoSQL databases – allowing applications to efficiently access their data in multiple ways. However, secondary indexes do come with a cost:

- Database writes will be slower when they need to update the secondary index
- Memory and storage is needed to store the secondary index

Partial indexes balance delivering good query performance while consuming fewer system resources. For example, consider an order processing application. The `order` collection is frequently queried by the application to display all incomplete orders for a particular user. Building an index on the userID field of the collection is necessary for good performance. However, only a small percentage of orders are in progress at a given time. Limiting the index on `userID` to contain only orders that are in the "active" state could reduce the number of index entries from millions to thousands, saving working set memory and disk space, while speeding up queries even further as smaller indexes result in faster searches.

By specifying a filtering expression during index creation, a user can instruct MongoDB to include only documents that meet the desired conditions.

When performing the database find operation, the application should include the value being used for the filtering as well as the indexed value in order for the partial index to be used by the optimizer. Review the documentation to learn more.

Additional Ops Manager Enhancements

Beyond the functionality discussed above, a number of enhancements improve productivity of Ops teams and simply installation and management.

- Ops teams can now automate their database restores reliably and safely using Ops Manager and Cloud Manager. Complete development, test, and recovery clusters can now be built in a few simple clicks.
- Integrating with existing storage infrastructure, MongoDB backup files can now be stored on a standard network-mountable file system.
- Operations teams can configure backups against specific collections only, rather than the entire database, speeding up backups and reducing the requisite storage space. This enhancement is also available in the Cloud Manager platform.
- Installation and configuration of all application and backup components can now be made through the centralized Ops Manager UI, which also provides a single, at-a-glance dashboard for health monitoring.
- Enhancements to the backup architecture provide faster time to first database snapshot.
- Eliminating false alarms, maintenance windows can now be defined during which Ops Manager and Cloud Manager alerts will not be triggered.

You can learn more about the enhancements discussed above in the Ops Manager documentation and the Cloud Manager documentation.

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 runs on MongoDB 3.2 and the WiredTiger storage engine – providing the easiest way to get access to the great features and enhancements described in this paper

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

Summary

MongoDB 3.2 is a significant release of the world's fastest growing database. New storage engine options, coupled with document validation, the enhanced replication protocol and sharding improvements extend the range of mission-critical applications MongoDB can serve. New tools such as the BI Connector, Compass, and Cloud Manager integration to APM platforms allow organizations to take advantage of MongoDB while protecting investments in existing frameworks and workflows.

- Download MongoDB 3.2 today.
- Evaluate MongoDB Enterprise, along with the Connector for BI, MongoDB Compass and Ops Manager by heading to the MongoDB download center.

To start using MongoDB 3.2 as quickly and efficiently as possible, bring in the experts. MongoDB's consulting engineers can deliver a private training on 3.2 features tailored to your needs, then work with you to develop a customized upgrade plan for your deployment. Learn more on the MongoDB 3.2 Upgrade Services.

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 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)