What’s New: MongoDB Server 4.2

February 2020
Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Getting Started</td>
<td>3</td>
</tr>
<tr>
<td>Transactional Data Integrity and Consistency</td>
<td>4</td>
</tr>
<tr>
<td>Supporting Complex Queries &amp; Real-Time Analytics</td>
<td>7</td>
</tr>
<tr>
<td>Resilience and Scalability</td>
<td>14</td>
</tr>
<tr>
<td>Enterprise Security</td>
<td>16</td>
</tr>
<tr>
<td>Freedom to Run MongoDB 4.2 Anywhere</td>
<td>19</td>
</tr>
<tr>
<td>Conclusion</td>
<td>24</td>
</tr>
<tr>
<td>We Can Help</td>
<td>25</td>
</tr>
<tr>
<td>Resources</td>
<td>26</td>
</tr>
</tbody>
</table>
Introduction

Since its initial release a decade ago, MongoDB is now firmly established as the most popular modern database in the industry. It is built on a technology foundation that enables developers to build apps faster and with high quality through:

1. The document data model – presenting you the best way to work with data.
2. A distributed systems architecture – allowing you to intelligently put data where you want it.
3. The freedom to run anywhere – giving you platform agility and eliminating vendor lock-in.

MongoDB Server 4.2 builds on this core design, advancing the state of the art in modern transactional and analytical data platforms. It gives you ACID guarantees and sophisticated data processing pipelines at any scale, secured by some of the industry’s most advanced encryption controls. You can run 4.2 anywhere: in your data center, in the cloud, or in a hybrid model, and with Atlas, you get the only fully-managed, cloud-native MongoDB service available on AWS, Azure, and GCP.

Getting Started

MongoDB Server 4.2 is Generally Available (GA), ready for production deployments. You can:

- Spin it up in the cloud using the on-demand MongoDB Atlas managed global database service.
- Download it to run on your own infrastructure (select 4.2.x under Version).

To get you up to speed, you can also take our new free online course covering the new features and tools in MongoDB 4.2.

Highlights of the Server 4.2 release are discussed through the rest of this What’s New guide.

---

**Figure 1:** Key MongoDB developments since 3.2 (November 2015)
Transactional Data Integrity and Consistency

Multi-document ACID transactions were introduced in MongoDB 4.0, declared GA in June 2018. MongoDB 4.2 sees the next phase of the transactions roadmap with the introduction of Distributed Transactions. 4.2 also includes driver and server-side enhancements to simplify application development, along with the ability to update a document’s shard key value in-place, and support for global point-in-time reads.

Distributed Transactions extend MongoDB’s multi-document ACID guarantees from replica sets to sharded clusters. As a result, it is now even easier for you to address a complete range of use cases by enforcing transactional guarantees across highly scaled apps.

Distributed Transactions adhere to our original design goals; they feel just like the transactions you grew up with in relational databases. They have a similar syntax, are multi-statement, and enforce snapshot isolation, making them familiar to anyone with prior transaction experience. If you’ve already used transactions in MongoDB 4.0, there is no difference using Distributed Transactions in 4.2 – the API and implementation is consistent whether you are executing transactions across documents, collections, and databases in a replica set, or across a sharded cluster. Full atomicity is maintained – if a transaction fails to commit on one shard, it will abort on all participant shards.

The Python code snippet as follows shows the core transactions API:

```python
with client.start_session() as s:
    s.start_transaction()
    collection_one.insert_one(doc_one, session=s)
    collection_two.insert_one(doc_two, session=s)
    s.commit_transaction()
```

The following snippet shows the transactions API for Java:

```java
try (ClientSession clientSession = client.startSession()) {
    clientSession.startTransaction();
    collection.insertOne(clientSession, docOne);
    collection.insertOne(clientSession, docTwo);
    clientSession.commitTransaction();
}
```
Use Cases for Distributed Transactions

MongoDB’s existing single document atomicity guarantees will meet 80-90% of an application’s transactional needs. This is because documents can bring together related data that would otherwise be modeled across separate parent-child tables in a tabular schema. One or more fields may be written in a single operation, including updates to multiple sub-documents and elements of an array. The guarantees provided by MongoDB ensure complete isolation as a document is updated; any errors cause the operation to roll back so that clients receive a consistent view of the document.

There are some workloads, however, where support for ACID transactions across multiple documents and shards is required. Back office “System of Record” or “Line of Business” (LoB) applications are the typical class of workload where Distributed Transactions can be useful. Examples include:

- **Payment and high frequency trading systems** – for example, updating the positions of securities composed of heavily traded stocks.
- **Event processing** – for example, complex orders comprised of products from multiple suppliers that all need to be notified as orders are placed.
- **Billing systems** – for example, adding a new Call Detail Record and then updating the monthly call plan when a cell phone subscriber completes a call.
- **Redistributing data** – modifying a document’s shard key value (discussed later in this Guide).

Multi-document ACID transactions enable you to serve these types of use-cases with MongoDB, without having to write your own application-side transactional logic.

Usage Guidance

Specific considerations for using Distributed Transactions in sharded clusters include:

- Only use drivers that have been updated for MongoDB 4.2.
- The existing default 60-second transaction run time limit still applies to transactions across shards. Note the run time is configurable and can be tuned for specific workload patterns, so if write volumes to the database are low, the limit can be increased.
- Transactions that affect multiple shards incur a greater performance cost as operations are coordinated across multiple participating nodes over the network.
- As with transactions on a replica set, there are no hard limits to the number of documents that can be read within a transaction. As a best practice, no more than 1,000 documents should be modified within a transaction. If an operation must modify more than 1,000 documents, developers should break the transaction into separate parts that process documents in batches.
- Snapshot read concern is the only isolation level that provides a consistent snapshot of your data across multiple shards. If latency is more critical, use the default local read concern which operates on a local version of the snapshot.
- Transactions spanning shards will error and abort if any participating shard contains a replica set arbiter.
- The performance of chunk migrations – used to rebalance data across shards – will be impacted if a transaction runs against a collection that is subject to rebalancing.

You can learn more about each of these in the production considerations documentation.
Path to Distributed Transactions

Adding multi-document transactions to MongoDB represents a multi-year engineering effort, beginning back in early 2015 with the integration of the WiredTiger storage engine to the MongoDB server. We’ve laid the groundwork in practically every part of the platform – from the storage layer itself to the replication consensus protocol, to the sharding architecture. We’ve built out fine-grained consistency and durability guarantees, introduced a global logical clock, refactored cluster metadata management, and more. And we’ve exposed all of these enhancements through APIs that are fully consumable by our drivers. With MongoDB 4.0 we delivered transactions on replica sets, and with 4.2 we are now feature complete in bringing multi-document transactions to sharded clusters.

Figure 2 presents a timeline of the key engineering projects that have enabled multi-document transactions in MongoDB. Two critical projects in the delivery of Distributed Transactions in 4.2 are the Oplog Applier Prepare Support and Distributed Commit Protocol:

- Prepare Support introduces a new state for a document that prevents threads from reading it while it is being committed atomically by a cross-shard transaction. This ensures MongoDB’s consistency and isolation guarantees are enforced.
- The Distributed Commit Protocol selects a coordinator node which is responsible for executing the transaction across shards. The new protocol determines whether the transaction commits or aborts, enforcing MongoDB’s all-or-nothing ACID guarantees.

The key design goal underlying all of the projects shown in Figure 2 is that their implementation does not sacrifice the key benefits of MongoDB. Developers still get the full power of the document data model and the advantages of distributed systems, without incurring any performance impact to workloads that don’t require multi-document transactions.

Take a look at our multi-document ACID transactions page where you can hear directly from the MongoDB engineers who have built transactions, review code snippets, and access key resources and documentation to get started.

<table>
<thead>
<tr>
<th><strong>MongoDB 3.0</strong></th>
<th><strong>MongoDB 3.2</strong></th>
<th><strong>MongoDB 3.4</strong></th>
<th><strong>MongoDB 3.6</strong></th>
<th><strong>MongoDB 4.0</strong></th>
<th><strong>MongoDB 4.2</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>New Storage engine (WiredTiger)</td>
<td>Enhanced replication protocol: stricter consistency &amp; durability</td>
<td>Shard membership awareness</td>
<td>Consistent secondary reads in sharded clusters</td>
<td>Replica Set Transactions</td>
<td>Distributed Transactions</td>
</tr>
<tr>
<td>WiredTiger default storage engine</td>
<td>Logical sessions</td>
<td>Make catalog timestamp-aware</td>
<td>Oplog applier support</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Config server manageability improvements</td>
<td>Retryable writes</td>
<td>Snapshot reads</td>
<td>Distributed commit protocol</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read concern “majority”</td>
<td>Causal Consistency</td>
<td>Recoverable rollback via WT checkpoints</td>
<td>Global point-in-time reads</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Cluster-wide logical clock</td>
<td>Recover to a timestamp</td>
<td>More extensive WiredTiger repair</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Storage API to changes to use timestamps</td>
<td>Shard catalog improvements</td>
<td>Transaction manager</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Read concern majority feature always available</td>
<td>Collection catalog versioning</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>UUIDs in sharding</td>
<td>Fast in-place updates to large documents in WT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Path to Distributed Transactions. Multi-year engineering investment
Ease of Developing with Transactions

Combined with Distributed Transactions, MongoDB 4.2 introduces additional enhancements that simplify application development:

- **Large Transactions**: By representing transactions across multiple oplog entries, you can now write more than 16MB of data in a single ACID transaction (subject to the existing 60-second default runtime limit). This means you no longer need to experiment to see whether you would hit the 16MB limit, especially for those use cases where you are atomically inserting multiple large documents into the database.

- **Deep Transactions Diagnostics**: Transactions metrics are now written to the mongod log to expose telemetry on execution time, locks, numbers of documents touched, time spent in the storage engine, and more, helping you tune application performance. In addition, the currentOp and serverStatus commands have been expanded with transaction-specific sections, enabling developers to get instant oversight of all running transactions in the database.

- **Transaction Error Handling**: New driver-side helpers with a callback API make it easier for you to develop retry logic in your app that handles any transactions aborts.

Mutable Shard Key Values

With MongoDB 4.2 you can now update shard key values to modify the placement of a document in a cluster, and MongoDB will automatically use a Distributed Transaction to move the document to the new shard for you (subject to the transaction run time limit).

There are times when you need to move a document to a new shard. For example:

- **Geo-Redistribution**: Re-homing documents to a newly provisioned geographic region to meet new regulatory requirements for data sovereignty, or to place data physically closer to local users, thus ensuring consistent, low-latency access.

- **Tiered Storage**: Migrating older documents to a low-cost storage tier based on your data aging policies.

Moving a document to a new shard would previously have required you to delete and then reinsert the document with the new shard key, managing the atomicity of the two operations yourself. Now MongoDB will automatically detect the updated shard key value and move the document to the correct shard for you in an all-or-nothing, ACID-compliant Distributed Transaction.

Global Point-In-Time Reads

Taking advantage of the transactions infrastructure in MongoDB 4.2, snapshot read isolation ensures queries and aggregations executed within a read-only transaction will operate against a globally consistent snapshot of the database across each primary replica of a sharded cluster.

As a result, a consistent view of the data is returned to the client, even when the data is distributed across shards, and as it is being simultaneously modified by concurrent write operations.

Supporting Complex Queries & Real-Time Analytics

With MongoDB’s rich query language, aggregation framework, and secondary indexes, you can query documents in many different ways – from simple lookups and range queries to creating sophisticated processing pipelines for data analytics and transformations, through to faceted search, JOINs, geospatial processing, and graph traversals. This is in contrast to most non-tabular databases which
offer little more than simple key-value access to your data.

MongoDB 4.2 expands support for complex queries and analytics with the addition of On-Demand Materialized Views, Wildcard Indexes, Conditional Update Expressions, and new mathematical operators in the Aggregation Pipeline. Making it easier to stream data through your data pipelines, MongoDB 4.2 also introduces a new official connector for Apache Kafka, along with updates to the MongoDB Compass GUI, and the General Availability of MongoDB Charts.

On-Demand Materialized Views

Using the new $merge stage, outputs from aggregation pipeline queries can now be merged with existing stored result sets whenever you run the pipeline, enabling you to create materialized views that are refreshed on-demand. Rather than a full stop replacement of the existing collection’s content, you can increment and enrich views of your result sets as new data is processed by the aggregation pipeline.

Materialized views are a common feature in relational databases, giving you the ability to pre-compute and store the results of common analytics queries. Typical use cases for them in MongoDB include:

- Rolling up a summary of sales data every 24 hours.
- Aggregating averages of sensor events every hour in an IoT app.
- Merging new batches of cleansed market trading data with a centralized MongoDB-based data warehouse so traders get refreshed market views across their portfolios.

By caching the results of common queries, you eliminate the performance overhead and latency of querying base collections to regenerate the entire result set when batches of new data are received. By defining unique keys on the output aggregations, you can choose what happens to the existing collection – for example, inserting documents to the materialized view if there is no match with the unique key, or if there is a match, then either replacing the document, or merging it with newly computed values. In the latter case, you avoid replacing the entire document, but rather take the newly computed values from the aggregation output and overwrite just the relevant existing fields of the document. So you could for example incrementally add counts to a daily rollup summary table without changing any of the other statistics computed for that day.

With 4.2’s Materialized Views you have the flexibility to output results to sharded collections – enabling you to scale-out your views as data volumes grow. You can also write the output to collections in different databases, further isolating operational and analytical workloads from one another. As the materialized views are stored in a regular MongoDB collection, you can apply indexes to each view, enabling you to optimize query access patterns, and run deeper analysis against them using MongoDB Charts, or the BI and Apache Spark connectors.

On-Demand Materialized Views represent a powerful addition to the analytics capabilities offered by the MongoDB server – enabling your users to get faster insights from live, operational data, without the expense and complexity of moving data through fragile ETL processes into dedicated data warehouses.

Wildcard Indexes

With Wildcard Indexes, you can now model your data even more naturally and still have great index support to efficiently serve a diverse range of query patterns.

Rather than try and second-guess every access pattern in advance, you can now define a filter that automatically indexes all matching fields, sub-documents, and arrays in a collection. For memory and storage efficiency, Wildcard Indexes are
implemented as a **sparse index**, where the index only contains entries for fields with non-empty values. You can also omit specific fields from being included in the Wildcard Index, even if those fields are contained within the document’s Wildcard Index coverage as part of the index definition.

While Wildcard Indexes are not a replacement for workload-based index planning, they do enable much simpler schema design for polymorphic document structures. These are typical in product catalog and content management applications that can have a wide variety of different fields in each document. For example:

- An e-commerce retailer will have very different document schemas for home furnishing items as compared to clothing lines.
- Content management apps typically require many dynamic fields to reflect differences between multiple content types such as external marketing assets and internal training materials.

In addition to the examples above, analytics is another use case that benefits from Wildcard Indexes. This is because they eliminate the overhead of full collection scans for ad-hoc queries and data discovery against MongoDB-based data warehouses or data lakes.

Wildcard Indexes support covered queries – so if the query can be resolved by the field in the index, there is no need to retrieve matching data from the base collection. They also support **collations**, allowing you to enforce language-specific rules for string comparisons.

Before the availability of Wildcard Indexes, users would often add an external search engine to their MongoDB deployments to provide this type of indexing flexibility. Now you can simplify your platform architecture by consolidating all query types against MongoDB 4.2. If you do need the full capability of a dedicated search engine, then Atlas Search, discussed later, is a powerful addition.

Like all MongoDB server indexes, Wildcard Indexes are updated synchronously and atomically with the base collection data, thus ensuring queries on indexes never return stale or deleted data. This is a major benefit over many non-tabular databases that only update server-side indexes “eventually,” causing data consistency and quality challenges you must resolve back in the application.

**More Expressive Update Language**

You can now use the aggregation pipeline to specify an update operation based on values in other fields, and then apply the update atomically.

For update and findAndModify commands, you can set a field to a value where that value depends on another existing value in the same document, or that must be computed differently depending on the current value of a field.

As an example of where you can use this:

- You have a customer document that includes credit_limit and payment_history fields. You can use an Expressive Update to only increase a customer’s credit limit if they have not been late in settling a payment in the past three years.
- When increasing the customer credit limit, you can set a field called previous_limit with the value that you are about to overwrite with the customers previous credit limit.
- Now that we have the previous credit limit value, a change stream can listen for events where the previous and new values cross a specific credit limit threshold, for example, firing when the credit limit has been increased by more than 30%. The change stream can then reactively notify that event to your credit risk service.

Expressive Updates mean there is less application code for you to write – now you can push such
operations down to the database for simpler, more expressive, and faster document updates.

New Aggregation Operators and Expressions

MongoDB 4.2 adds support for new regex aggregation operators, a dozen new mathematical expressions for rounding and trigonometry, as well as current time expressions like `$NOW`, all of which reduce the amount of code you need to implement back in your app tier.

With close to 100 existing expressions, the aggregation pipeline makes it quick and easy to compute and transform your data. The new trigonometry expressions further simplify apps that need to work with directions and distances between objects. Use cases include navigation systems, medical imaging, audio and graphics processing, electrical and mechanical engineering, and games development. The new `$round` expression pushes all operations that round a number to a whole integer or to a specified decimal place down into the database.

Using `$NOW`, developers can reference the current wall clock time to allow filtering documents by date in aggregation queries, while regular expression support has been added to allow string matching and manipulation in the aggregation pipeline.

MongoDB Connector for Apache Kafka

Many customers use MongoDB and Kafka as part of microservices architectures and event-driven data pipelines, with integration provided by community-developed connectors. Making it easier to build robust, reactive data pipelines that move events between systems, MongoDB is introducing an official Kafka connector, developed in collaboration with the community, verified by Confluent, and supported by MongoDB engineers.

Unlike many Kafka connectors, the official MongoDB connector, enables MongoDB to be configured as both a sink and as a source for Kafka:

- Configured as a Kafka sink, you can ingest events from your Kafka topics directly into MongoDB collections, exposing the data to your services for efficient querying, enrichment, and analytics.
- When the connector is configured as a source for a Kafka topic, data is captured via Change Streams within the MongoDB cluster and published straight into Kafka topics. This enables consuming apps to react to data changes in real time using an event-driven programming style.

To illustrate the power of the MongoDB Kafka connector, consider an e-commerce scenario where you have many autonomous microservices providing functionality such as inventory management, pricing, and logistics. Changes to an item’s properties such as an increase in fuel cost may affect pricing and inventory replenishment:

- When using MongoDB as a Kafka sink, this price change data from a Kafka topic will be inserted into a MongoDB collection. From there the microservice that manages the pricing can leverage all the advanced analytical querying capabilities such as the MongoDB aggregation pipeline to calculate the new price. This new pricing data can be further analyzed and visualized using tools like MongoDB Charts and the BI Connector.
- With MongoDB change streams, data can also be pushed back to a Kafka topic using the source capability of the connector. This enables other microservices that depend on the pricing data to be notified of the latest changes, using a fully reactive, event driven programming pattern.
For more information and updates, check out the web page and try the latest release on the Confluent Hub.

**MongoDB Compass**

MongoDB Compass, the GUI for MongoDB, makes it easy to interact with your data. You can see and manage database structure, analyze the schema of a collection, run ad hoc queries and evaluate their performance, view and create indexes, graphically build aggregation pipelines, and more. Now, Compass gives you more capabilities than ever for interacting with and managing your MongoDB data.

**Schema Validation**

MongoDB Compass now supports JSON Schema format for Schema Validation. The full validation capabilities of the $jsonSchema operator are available in Compass’s Validation tab: flag certain fields as required, define properties for specific fields, and allow or disallow additional fields not defined in the schema. As you build a rule, Compass will auto-suggest field names, BSON data types, and validation keywords to simplify the process. As shown in Figure 3, Compass shows a live preview of documents that pass and fail the validation rule, making it easy to see whether the rule has the desired behavior. You can also view and manage your existing validation rules, altering or deleting them as needed.

---

**Sample Document That Passed Validation**

```json
_id: ObjectId("5c45bf2447de2b5c9979bb98")
month: 2018-08-01T00:00:00.000+00.00
comment: "aug 2018"
value: 111675
```

**Sample Document That Failed Validation**

No Preview Documents

---

**Figure 3:** Configuring schema validation rules in MongoDB Compass
Read-Only View Management

You can now interact with read-only views in MongoDB Compass, making it easier to create and manage views on a collection. Compass’s Aggregation Pipeline Builder already provides an intuitive, GUI for building aggregation pipelines, with intelligent auto-complete and drag and drop of stages. With View Management, you can now create a read-only view directly from the Aggregation Pipeline Builder. Simply build the aggregation pipeline that will define the view, choose to create a view from that pipeline, and give the view a name to save it.

Views that exist – whether created via Compass or the shell – will appear in Compass’s collection view and sidebar. You can use Compass to explore data in a view reading documents as you would in a collection. You can also choose to drop a view or revise the aggregation pipeline that defines it.

Support for New Features

MongoDB Compass also supports features now available in MongoDB 4.2 and MongoDB Atlas. If you use MongoDB Atlas Data Lake (discussed later) to analyze your S3 data, you can build and issue queries, including complex aggregation pipelines, in MongoDB Compass. Atlas Search will also be accessible via Compass, and other new and updated aggregation pipeline stages will be available in Compass’s Aggregation Pipeline Builder. Compass will be able to connect to instances using Field-Level Encryption, returning only ciphertext for the encrypted fields. Wildcard indexes will also be supported in Compass’s Indexes tab.

MongoDB Charts General Availability

MongoDB Charts is the fastest and easiest way to create visualizations of MongoDB data. Available as a managed service in MongoDB Atlas, or downloadable for you to run on your infrastructure, you can create charts and graphs, build dashboards, share them with other users for collaboration, and embed them directly into your web apps to create engaging user experiences.

MongoDB Charts natively supports the richness of the document model so there is no need to flatten subdocuments and arrays into the tabular structures for use with traditional BI tools. As shown in Figure 4, with its intuitive UI, MongoDB Charts enables even non-technical uses, such as business analysts, to visualize rich, complex data so everyone on your team can get value from data faster regardless of their skills.
New in the GA release, MongoDB Charts helps you make sense of geospatial data, such as latitude and longitude coordinates, or polygons on the earth’s surface. Geolocation capabilities are already a part of the MongoDB platform including the ability to natively store and analyze geolocation coordinates and optimize queries with geolocation indexes. Charts extends these capabilities by allowing you to perform geolocation analysis without having to write any code at all. MongoDB Charts supports:

- Geo-scatter plots to display data points on a map based on their latitude and longitude.
- Choropleth charts that shade geographical areas or regions to show variation or patterns across the displayed location.
- Geo-heatmap charts to visualize the density of a value as a smoothed continuous heat map overlaid on top of a map.

Dashboard viewers can pan and zoom maps containing geospatial visualizations adding to the overall explorability of location data.

The Charts GA release also supports workload isolation, working against secondary replicas or dedicated analytics nodes in MongoDB Atlas clusters to avoid contending for resource with your operational apps. You also have the flexibility to create either standalone dashboards or embed charts directly into your apps. Embedding enables you to build richer user experiences while reducing development time and effort. Web application designers will be able to rely on the consistent visual presentation of those charts, getting rid of hard to maintain, one-off graphical rendering code. All MongoDB Charts are live updated and deeply customizable too, making them the perfect fit for your modern, styled web applications and sites.

Figure 4: Creating rich visualizations of your data with MongoDB Charts
Resilience and Scalability

With its distributed systems architecture, MongoDB enables you to put your data where your apps and users need it. You can deploy MongoDB both within and across geographically dispersed data centers and cloud regions, providing levels of availability, workload isolation, scalability, and data sovereignty unmatched by traditional relational databases or NoSQL stores.

MongoDB 4.2 continues to build on this distributed systems architecture with Retryable Reads and Writes, New Online Index Builds, 10x faster Primary stepDowns, Keepalive Connections, the Storage Node Watchdog, and a faster initial sync – all providing higher system resilience. The addition of Zstandard compression enables you to create even more scalable and efficient storage infrastructure for your MongoDB deployment.

Retryable Reads

Failed read operations are now automatically retried by the MongoDB drivers. A Retryable Read is triggered after 30 seconds if the driver has not received any results from a query; it is retried once, and honors the query’s configured read preference.

The addition of Retryable Reads moves the complexity of handling temporary system failures from the application to the drivers. Now, rather than the developer having to implement custom, client-side code, the MongoDB driver will automatically retry reads that have failed due to transient network errors or a replica set node being temporarily unavailable due to an election or maintenance.

When coupled with self-healing node recovery – typically within 5-seconds or less – MongoDB’s Retryable Reads and Writes (discussed below) enable you to develop more resilient apps with dramatically less boilerplate client-side code.

Retryable Writes

All official MongoDB drivers now default to retrying writes that failed due to a transient error. This gives you consistent exception handling behavior whatever programming language you use, and does not incur any performance overhead or data inconsistency.

MongoDB 3.6 introduced Retryable Writes back in 2017. The MongoDB driver automatically retries writes in the event of a temporary network failure or a primary replica election, while the MongoDB server enforces exactly-once processing semantics. By assigning a unique transaction identifier to each write operation, the driver re-sends that ID to enable the server to evaluate success of the previous write attempt, or retry the write operation if needed. This means retryable writes can be applied to both idempotent and non-idempotent operations.

Applications that cannot afford any loss of write availability, such as e-commerce applications, trading exchanges, and time-series data ingestion immediately benefit from Retryable Writes. You can deliver always-on, global availability of write operations, while minimizing the amount of error handling code you have to implement, and without the risks of data loss and stale reads imposed by eventually consistent, multi-master or masterless systems.

New Online Index Builds

MongoDB 4.2 introduces a new approach to index builds that combines the best qualities of the existing foreground and background index builds, and replaces those former methods:

- Foreground index builds were the fastest approach to building indexes and produced the most optimized index structures. However, they took an exclusive, non-yielding lock on the database that blocked other database operations.
- Background index builds allowed operations to continue running, but took longer to
complete and were less space efficient than foreground builds.

Index builds in 4.2 are as fast as foreground builds and produce the same efficient B-Tree index structures, but avoid blocking service to the app so you maintain availability for your users. Creating an index follows precisely the same syntax as pre-4.2 index commands, and if the background argument is used, it is simply ignored and the new index build process is used.

10x Faster stepDown and Keepalive Connections

Whatever database you use, planned maintenance events such as system upgrades and patching can account for up to a third of all downtime. Reducing the length of your maintenance windows can drive an immediate uplift in system availability and customer satisfaction, and this has therefore been a focus for MongoDB 4.2 development:

- The stepDown command instructs the primary member of the replica set to become a secondary, and is typically used during rolling replica restarts, used to perform system maintenance without taking the entire cluster down. As soon as you issue a stepDown command, the MongoDB primary now immediately initiates an election rather than waiting on the secondaries to call it. Before 4.2, this process by default took 10 seconds, but many users, including the MongoDB Atlas service, turn this interval down to 5 seconds or less. With the new stepDown process, a replacement primary is promoted in less than a second, significantly reducing any user interruption.
- After the stepDown, existing read connections on the former primary are kept open, and are only re-established on the newly elected primary once reads have completed and their associated cursors closed. This eliminates both failed reads and your apps waiting for the drivers to restart queries on the new primary.

One global e-commerce customer using MongoDB as part of its platform, serving more than 100 million customers around the world, reports that maintenance events can cost them $10,000 for each primary node they upgrade. With MongoDB 4.2, planned maintenance events are now simpler, faster, and have little to no user or business impact.

Storage Node Watchdog

Adding to existing network heartbeats that monitor the accessibility of cluster members, the Storage Node Watchdog can be used to monitor MongoDB’s underlying filesystems and initiate an automated failover if any of them becomes unresponsive. This allows the cluster to automatically self-heal in the face of a wider range of failure conditions, increasing overall database resilience and uptime.

The Storage Node Watchdog monitors the filesystem directories containing the database files, the journal, and the diagnostics and audit logs on both the primary and secondary replica set members. If any of the filesystems become unresponsive, the watchdog terminates the mongod process, triggering an election if it is a primary replica that is affected. This allows faster cluster recovery by proactively catching those nodes that are still able to generate heartbeats, but are encountering storage subsystem failures that prevent primaries from applying new write operations from the app or secondary replicas from applying oplog entries.

The Storage Node Watchdog has been available to customers of the MongoDB Enterprise Server packaged with MongoDB Enterprise Advanced and to customers of MongoDB Atlas. Now with 4.2, the watchdog has been integrated into the community builds as well, providing full access for all MongoDB users.
15% Faster Initial Sync

The initial sync process has been refactored in MongoDB 4.2, making it 15% faster, while also reducing load on the source replica. Initial sync is used in multiple scenarios:

- Adding new replicas to your cluster to scale out read traffic, creating analytics-only nodes for workload isolation, or distributing data closer to users in newly provisioned geographic regions.
- Replacing a failed secondary node to restore a cluster to full resiliency levels.
- Resynchronizing an existing replica set member that has become stale with the rest of the cluster and can never catch up using regular replication. This typically occurs after the node has been recovered after a lengthy outage and the oplog’s capped collection has rolled over.

Whatever your use case for initial sync, MongoDB 4.2 gets it done faster, with the greatest gains coming when you are synchronizing several large collections, or many small collections across replicas, or when replicas are connected via high latency network connections.

Zstandard Compression

Adding to the existing snappy and zlib libraries, MongoDB 4.2 now supports Zstandard (Zstd) compression, enabling storage savings equal to or better than zlib with lower CPU overhead.

Zstandard was introduced by Facebook in 2016, offering a tunable, real-time compression algorithm that allows users to balance compression ratios against CPU utilization. Zstd is now configurable for both the default MongoDB WiredTiger storage engine, and the Encrypted storage engine, and applies compression to your collections and journal on disk, and network communications across the cluster. To use Zstd rather than the default snappy compressor, you simply modify the block compressor option in the WiredTiger configuration file. You can learn more about configuration options in the MongoDB 4.2 Zstd release notes.

While storage savings are dependent on the actual data to be compressed, internal testing has shown Zstd achieves file size reductions of between 35% and 55% over snappy, while incurring lower CPU overhead than zlib. If you are using zlib today to achieve the highest compression ratios, you should evaluate Zstd as an alternative.

As a result of Zstd’s efficiency improvements, you can create more scalable and efficient storage infrastructure for your MongoDB deployment, and take advantage of the latest in high performance storage technologies, while incurring lower compute overhead and cost.

Enterprise Security

Whether running MongoDB yourself, or via the MongoDB Atlas global cloud database service, you have access to some of the most advanced security controls available in a modern database — fine grained Role-Based Access Control (RBAC), end-to-end (E2E) encryption of data in-flight and at-rest, and extensive auditing.

With new features including Client-Side Field Level Encryption (FLE), multi-CA support, and encryption key protocol forward secrecy, MongoDB 4.2 continues to enhance the security controls available to you. This brings even more protection to your most valuable information assets and exceeds emerging demands of privacy regulations.

Client-Side Field Level Encryption

With MongoDB 4.2, you can now selectively encrypt individual document fields, each optionally secured with its own key and decrypted seamlessly on the client.

Our implementation of Field Level Encryption is totally separated from the database, making it transparent to the server, and instead handled
exclusively within the MongoDB drivers on the client. All encrypted fields on the server – stored in-memory, in system logs, at-rest, and in backups – are rendered as ciphertext, making them unreadable to any party who does not have client access along with the keys necessary to decrypt the data. This is a different and more comprehensive approach than the column encryption used in many relational databases. As most handle encryption server-side, data is still accessible to administrators who have access to the database instance itself, even if they have no client access privileges.

In developing Field Level Encryption, we have worked with two of the world’s leading authorities on database cryptography, including a co-author of the IETF Network Working Group Draft on Authenticated AES encryption. Drawn from academia and industry, these teams have provided expert guidance on our FLE design and reviewed the FLE software implementation.

Field Level Encryption serves as an important addition to your defense-in-depth security strategy. Consider a typical MongoDB deployment from a risk management perspective:

- With filesystem encryption alone, system administrators or attackers who elevate system-level user privileges still have access to plaintext database files both on server storage and in memory.
- The MongoDB Encrypted Storage Engine provides a way to mitigate filesystem and backup file access risks by encrypting all MongoDB data before it is written to disk, and ensuring keys are non-persistent. Any attacker obtaining database files from the filesystem would be unable to read them. However, administrators and compromised authenticated database users still have access to the underlying data on a running instance.
- Using MongoDB FLE you can now protect individual fields with all key management, encryption, and decryption operations occurring exclusively outside the database server. With FLE enabled, a compromised administrator or user obtaining access to the database, the underlying filesystem, or the contents of server memory (for example, via scraping or process inspection) will only see unreadable encrypted data. While storage engine encryption and FLE can be used independently, they bring the greatest levels of protection when used together.

MongoDB Field Level Encryption provides:

- **Automatic, transparent encryption:** Application code can run unmodified for most database read and write operations when FLE is enabled. Other client-side approaches require developers to modify their query code to use the explicit encryption functions and methods in a language SDK.
- **Separation of duties:** System administrators who traditionally have access to operating systems, the database server, logs, and backups cannot read encrypted data unless explicitly given client access along with the keys necessary to decrypt the data.
- **Regulatory Compliance:** Comply with “right to be forgotten” conditions in new privacy regulations such as the GDPR and the CCPA – simply destroy the customer key and the associated personal data is rendered useless.
- **Minimal performance penalty:** As encryption is handled on the client, impacts to server performance are minimal when working with encrypted fields.

For isolation, each field can be encrypted with its own unique key natively integrated with external key management services backed by FIPS 140-2 validated Hardware Security Modules (HSMs) such as Amazon’s KMS.

To understand more about implementing FLE, let’s take a look at the flow of a query submitted by an authenticated client, as represented in Figure 5.
1. Upon receiving the query, the MongoDB driver analyzes the query to determine if any encrypted fields are involved in the filter.
2. The driver requests the fields’ encryption keys from the external key manager.
3. The key manager returns the keys to the driver, which then encrypts the sensitive fields.
4. The driver submits the query to the MongoDB server with the encrypted fields rendered as ciphertext.
5. MongoDB returns the encrypted results of the query to the driver.
6. The query results are decrypted by the keys held by the driver, and returned to the client.

Since the database server has no access to the encryption keys, certain query operations such as sorts and range-based queries on encrypted fields are not possible unless implementing a client-side solution, which may have additional performance impacts. As a result, FLE is best applied to selectively protect just those fields containing highly sensitive, personally identifiable data such as credit card information and social security numbers.

Field Level Encryption is especially powerful when using managed database services like MongoDB Atlas. In Atlas, all cluster storage and backups are encrypted at rest by default. You can provide an additional layer of encryption by protecting the database keys using the cloud providers’ key management service. With the addition of FLE, keys are protected in an isolated, customer-managed AWS KMS account (additional key management solutions are coming online in the near future). Atlas Site Reliability Engineers and Product Engineers have no mechanism to access FLE KMS keys, rendering data unreadable to any MongoDB personnel managing the underlying database or...
infrastructure for you. By combining these security capabilities, you eliminate common security concerns when moving database workloads to managed services in the cloud. This is because you both control and manage the encryption keys, rather than having the database operator manage the keys for you.

For more information please see the FLE documentation.

Other Security Control Enhancements

3x Lower Overhead to Auditing

In MongoDB 4.2, internal locking within the audit plug-in has been refactored, resulting in a 3x reduction in auditing overhead. This makes it more efficient to run auditing and reduces costs, given regulatory compliance standards that demand you maintain a comprehensive audit trail of activity against the database.

Multi-Certificate Authority and Forward Secrecy in TLS Encryption

With MongoDB 4.2, you can now configure separate Certificate Authorities (CAs) for context-sensitive certificate validation on a server instance based on the communication type – for example between inbound or outbound client and server communications, and for internal cluster communications between cluster members.

Additionally, in-flight encryption also now supports the latest TLS cipher suites offering Forward Secrecy, providing the assurance that your users’ session keys will remain secure even if the server’s private key is compromised.

These enhancements improve network security by reducing potential attack surface area.

Zero Downtime Certificate and Keyfile Rotation

You can now change your keyfiles and X.509 certificates – even when modifying the Distinguished Name – without having to take down your cluster.

With MongoDB 4.2, the server can honor more than one CA and valid keyfile at once, so that a cluster can trust new credentials incrementally, staging the rollout of a new key and deferring invalidation of the old key until every node in cluster has been updated. This allows you to modify the cluster’s credentials via a rolling restart – a process automated by MongoDB’s operational tools and the Atlas service – thereby avoiding application downtime.

Integration with Secrets Managers

Configuration files now support __exec and __rest directives that allow you to securely provide passwords and secrets to MongoDB at startup. This provides easy integration with external secrets managers such as HashiCorp Vault, OS X Keychain, Windows Credential Manager, and AWS Secrets Manager.

You can now manage MongoDB security credentials alongside your existing enterprise application estate. You can centrally enforce password management policies, and eliminate storing user credentials in MongoDB configuration files.

Next Steps

You can learn more about how to secure your self-managed MongoDB deployments from our Security Checklist, and how we do it for you at the MongoDB Cloud Services Trust Center.

Freedom to Run MongoDB 4.2 Anywhere

Whether you plan to run your apps in your own facilities, as a serverless, cloud-native platform, or a hybrid deployment model in between, MongoDB provides complete infrastructure agility. With the freedom to run anywhere you can future-proof your apps and eliminate vendor lock-in.
Run by Us, For You: MongoDB Atlas

Launched back in 2016, the fully-managed and fully-automated MongoDB Atlas global cloud database service is now available in close to 70 regions on AWS, Azure, and GCP, serving a broad range of workloads for startups, Fortune 500 companies, and government agencies.

MongoDB 4.2 is available on Atlas, enabling you to evaluate and deploy the features discussed earlier using a fully-managed service, so you have nothing to install or run yourself. We are also releasing additional Atlas enhancements that give you auto-scaling and full text search. Beyond the database, we are previewing the new MongoDB Atlas Data Lake Service. An overview of all of this follows.

Auto-Scaling

Auto-Scaling for MongoDB Atlas brings fully automated capacity management to your managed MongoDB databases.

Even before Auto-Scaling, Atlas made it easy to adjust the dimensions of your managed databases with a simple API call or a button click in the UI. The database service automates scaling up or down with no impact to your application. In addition, Atlas automatically increases storage capacity when disk utilization crosses a preset threshold. These existing features make it much simpler to carry out scaling events in response to workload changes, but they still require you to review performance metrics or respond to alerts to decide when to scale.

With Auto-Scaling enabled, Atlas will track key resource utilization metrics in real time and adjust database sizes up or down as needed by using predictive modeling and proven practices developed from managing tens of thousands of MongoDB deployments. As with manual scaling, events are executed in a rolling fashion across your Atlas deployment, ensuring no impact to your apps. The feature can be easily toggled on or off using the API or UI to preemptively prepare for periods of increased load or application usage, for example.

In the coming months, Auto-Scaling will become more sophisticated by leveraging multidimensional decision-making that incorporates additional real-time performance metrics.

Atlas Search (Beta)

Atlas Search provides rich text search capabilities against your fully managed databases with no additional infrastructure or systems to provision, manage, or scale. When you enable Atlas Search for a cluster, index creation and analyzer configuration can both be executed through the Atlas Data Explorer or via the Atlas API. Once indexes have been created, you can run search queries using MongoDB’s aggregation pipeline using the new $searchBeta stage.

Many applications leverage full-text search to deliver a better user experience. Whether it’s an e-commerce site, a travel booking platform, a job board, or a content repository, language-aware search grants developers and end-users the flexibility to filter, rank, and sort through their data to quickly surface the most relevant results. To gain access to these capabilities, a common architecture pattern is to index and query the data from your database using a search engine such as Elasticsearch or Solr. This can complicate development and operations as it often requires additional infrastructure as well as learning, maintaining, and scaling an entirely separate system.

Atlas Search allows you to:

- **Get access to rich text search capabilities** without the need to export data and set up, maintain, and scale a separate search platform. Simply enable Search within Atlas, create your indexes, and go. Atlas Search is available with replica sets and sharded clusters.
- **Move fast.** For developer simplicity, Atlas Search leverages the MongoDB query language. This allows you to easily combine
your search queries with other MongoDB operators to build powerful data processing pipelines.

- **Search just what you need.** Dynamic indexing, which makes your entire collection searchable, is a good way to get started quickly. Or leverage your knowledge of your data and make use of static indexes to selectively choose which fields in your collection to index for better performance.

- **Take advantage of the power of Lucene.** A wide range of query operators for text data are supported — everything from wildcard and fuzzy search to boolean and compound queries. Improve search quality with language analyzers that process text using linguistic rules. Inject business data into search results with sophisticated scoring and improve the user experience with highlighted snippets.

Atlas Search is available in beta for M30 clusters and above running MongoDB 4.2.

**MongoDB Atlas Data Lake (Beta)**

MongoDB Atlas Data Lake, illustrated in Figure 6, allows you to quickly and easily query data in any format on Amazon S3 using the MongoDB Query Language (MQL) and tools.

With Atlas Data Lake you can realize the value of your data lake faster: you don’t have to move data anywhere, you can work with complex data immediately in its native form, and with its fully-managed, serverless architecture, you control costs and remove the operational burden.
Unlocking the Data Lake

Traditional data warehouses have been unable to keep up with the data avalanche generated by the digital economy, while Hadoop incurs massive cost and complexity. Here is how the MongoDB Atlas Data Lake helps you get value from your data faster:

- **No schema definition required.** Any richly structured data stored in JSON, BSON, CSV, TSV, Avro, and Parquet formats can be analyzed in place without incurring the complexity, cost, and time-sink of ingestion, transformation, and metadata management.
- **Work with data at any scale, anywhere.** Decoupled compute and storage allows you to seamlessly expand and contract each tier of your data lake independently. Parallelize complex queries to take advantage of existing data partitioning.
across multiple S3 buckets, delivering performance – even for global analytics that access data in multiple AWS regions.

- **Accelerate productivity of your developer and data teams.** Spin up your data lake right alongside your Atlas OLTP clusters from a common UI. Then query and visualize your data using the same MongoDB Query Language and tools. You can query data any way you want – from simple lookups and ad-hoc queries through to sophisticated aggregation and data processing pipelines.

- **On-demand and serverless.** Simply provide access to your existing S3 buckets with a few clicks from the Atlas UI and start running queries immediately. Atlas Data Lake is completely serverless so there is no infrastructure to set up and manage, and you pay only for the queries run and only when actively working with the data.

### Use Cases

All users across your organization can now benefit from the data in your S3 data lake, working together to answer questions and build data-rich applications. Common use cases for MongoDB Atlas Data Lake include:

- **Data Lake Analytics.** Explore all of your rich data naturally, democratizing access across business units and users.

- **Data Products and Services.** Create data snapshots, market research, and insight-as-a-service products.

- **Machine Learning and AI.** Accelerate data exploration and feature engineering, exposing models to build more intelligence into your apps and services.

- **Active Archives.** Run historical analysis against data assets retained in long-term cold storage.

You can learn more from the MongoDB Atlas Data Lake product page.

### Run by You, with Tools from Us: Ops Manager

**Ops Manager** is the best way to self-manage MongoDB, making it fast and easy for you to automate, monitor, and back up MongoDB. A dramatically simplified deployment and backup architecture, coupled with Kubernetes integration are key new features in Ops Manager 4.2.

### Headless Backups

Eliminating the head database from Ops Manager backup reduces required hardware and storage infrastructure by 3x, while accelerating backup and restore operations by an order of magnitude.

The head database was used to store a copy of the database being backed up. Now in Ops Manager 4.2, the process has been rearchitected to leverage WiredTiger backup cursors, a new storage engine feature that eliminates the need for this extra storage requirement.

With the evolution to headless backups, you can tighten RTO and RPO service levels, while reducing your cost of data protection. Note that in this initial release, Headless Backups support replica set deployments in MongoDB Enterprise Server 4.2 and above, with sharded cluster support coming later in 2019.

### One Agent to Rule Them All

Ops Manager 4.2 now combines the previously separate automation, monitoring, and backup agents into a single, consolidated, multi-function agent. This simplifies both initial deployment, and ongoing management and upgrades. Ops Manager automation can install the agent for you, or you can do it yourself manually or via your own DevOps scripts.
**MongoDB and Kubernetes**

Kubernetes is the industry leading container orchestration platform. It provides you with a consistent automation and management experience anywhere from on-premises infrastructure to the public cloud. You can use Kubernetes to manage the full lifecycle of your MongoDB clusters, wherever you choose to run them.

**MongoDB Enterprise Kubernetes Operator GA**

Released as a beta with MongoDB 4.0, in 4.2 the Enterprise Operator for Kubernetes is now GA and certified for production deployments.

Kubernetes Operators are application-specific controllers that extend the Kubernetes API to create, configure, and manage instances of stateful applications such as databases. On self-managed infrastructure – whether on-premises or in the cloud – Kubernetes users can use the MongoDB Enterprise Operator for Kubernetes and MongoDB Ops Manager to automate and manage MongoDB clusters. You have full control over your MongoDB deployment from a single Kubernetes control plane. You can use with the operator with upstream Kubernetes, or with any popular distribution such as Red Hat OpenShift and Pivotal Container Service (PKS).

For more information check out the MongoDB Kubernetes documentation.

**MongoDB Atlas Service Broker**

With the MongoDB Atlas Service Broker (coming soon), you will be able to spin up and manage Atlas clusters directly from Kubernetes, controlling infrastructure provisioning, database setup, global distribution, and more. Compatible with the Open Service Broker API, the Atlas Service Broker makes MongoDB Atlas part of your Kubernetes service catalog, allowing cluster creation and management directly from the Kubernetes control plane.

**Conclusion**

The 4.2 release builds upon MongoDB's core foundations:

- **Best way to work with data**: adding Distributed Transactions and Global Point in Time Reads, On-Demand Materialized Views, Wildcard Indexes. Expressive Updates, and operators for the aggregation pipeline. Updates to the Compass GUI, the GA of MongoDB Charts, and the new official Apache Kafka connector all enable you to build apps faster, while Client-Side Field Level Encryption enables you to build them more securely.

- **Intelligently place data where you need it to be**: with Retryable Reads and Writes, New Online Index Builds, 10x faster Primary stepDowns, Keepalive Connections, the Storage Node Watchdog, and a faster initial sync – all providing higher system resilience. The addition of Zstandard compression enables you to create even more scalable and efficient storage infrastructure for your MongoDB deployment.

- **Freedom to run anywhere**: MongoDB Atlas is expanded with Auto-Scaling, Atlas Search, and the new Data Lake Service. Ops Manager now features Headless Backups and a single agent architecture. Both MongoDB Enterprise and Atlas will be integrated with Kubernetes, enabling you to run your cloud native apps consistently across deployment environments.

MongoDB Server 4.2 is GA and ready for production deployment today:
• Spin it up in the cloud using the on-demand MongoDB Atlas managed global database service.
• Download it to run on your own infrastructure (select 4.2.x under Version).

To get you up to speed, you can also take our new free online course covering the new features and tools in MongoDB 4.2.

Safe Harbor Statement

This paper contains “forward-looking statements” within the meaning of Section 27A of the Securities Act of 1933, as amended, and Section 21E of the Securities Exchange Act of 1934, as amended. Such forward-looking statements are subject to a number of risks, uncertainties, assumptions and other factors that could cause actual results and the timing of certain events to differ materially from future results expressed or implied by the forward-looking statements. Factors that could cause or contribute to such differences include, but are not limited to, those identified in our filings with the Securities and Exchange Commission. You should not rely upon forward-looking statements as predictions of future events. Furthermore, such forward-looking statements speak only as of the date of this presentation.

In particular, the development, release, and timing of any features or functionality described for MongoDB products remains at MongoDB’s sole discretion. This information is merely intended to outline our general product direction and it should not be relied on in making a purchasing decision nor is this a commitment, promise or legal obligation to deliver any material, code, or functionality. Except as required by law, we undertake no obligation to update any forward-looking statements to reflect events or circumstances after the date of such statements.

We Can Help

We are the company that builds and runs MongoDB. More than 15,500 organizations rely on our commercial products. We offer software and services to make your life easier:

MongoDB Atlas is the database as a service for MongoDB, available on AWS, Azure, and GCP. It lets you focus on apps instead of ops. With MongoDB Atlas, you only pay for what you use with a convenient hourly billing model. Atlas autoscales in response to application demand with no downtime, offering full security, resilience, and high performance.

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

MongoDB Atlas Data Lake allows you to quickly and easily query data in any format on Amazon S3 using the MongoDB Query Language and tools. You don’t have to move data anywhere, you can work with complex data immediately in its native form, and with its fully-managed, serverless architecture, you control costs and remove the operational burden.

MongoDB Charts is the best way to create visualizations of MongoDB data anywhere. Build visualizations quickly and easily to analyze complex, nested data. Embed individual charts into any web application or assemble them into live dashboards for sharing.

MongoDB Stitch is a serverless platform which accelerates application development with simple, secure access to data and services from the client – getting your apps to market faster while reducing operational costs and effort.
MongoDB Mobile lets you store data where you need it, from IoT, iOS, and Android mobile devices to your backend – using a single database and query language.

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 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

Presentations

White Papers

Free Online Training

Webinars and Events

Documentation