RavenDB

Comparing RavenDB & MongoDB

Our commitment to simplification of experience without any trade-offs matters and makes a difference

Oren Eini

Contents

RavenDB vs MongoDB

One question that we often hear from prospective clients is 'How does RavenDB stack up against MongoDB?' To provide a comprehensive answer, we've organized this white paper and broken it down into topics of interest.

If you are interested in third-party empirical performance data, *Rakuten Kobo CTO Trevor Hunter has shared a video* detailing their extensive tests on RavenDB, MongoDB, and Couchbase as part of their NoSQL data platform selection process.

Their findings reveal (skip to 8:00), that while RavenDB and MongoDB often delivered similar performance levels, RavenDB consistently used fewer machine resources and offered instant server failover (zero downtime), while MongoDB took a few minutes.

Final verdict spoiler? Ultimately, Rakuten Kobo chose RavenDB for its unique capabilities.

This white paper, however, rather than focusing on performance metrics, aims to shed light on the distinct functionalities of the two veteran NoSQL data platforms, RavenDB and MongoDB, and doesn't focus on performance metrics.

Our goal is to provide you with a valid comparison that covers the key features, capabilities and differentiators between the platforms.

Introduction to RavenDB and MongoDB

RavenDB is an open-source, NoSQL, high-performance, *multi-model* distributed data platform specializing in online transaction processing (OLTP). It has been fully transactional (ACID) since its first launch in 2009.

Some key advantages:

- **Peace of mind** - Consistent zero downtime for clients when a server goes down thanks to high availability in a multi-master cluster
- **Safe-by-default**
    - Industry-standard security, including encryption at rest and in transit, is relatively simple to apply (automatic and HIPAA certified on RavenDB Cloud)
    - Default ACID transactions protect data integrity
    - Default settings prevent hardware overload
    - Built-in ongoing backups to various storage platforms
- **Automatic indexes** in RavenDB adjust to your query behavior and ensure consistently high performance, without your DBAs having to spend time and effort on manual optimization
- **Out-of-the-box high performance** (150K writes/1M reads per second on commodity hardware) means you can focus on your app instead of optimizing the database
- **Minimal administration** and need for tech support saves lots of developer time
- **Unstructured data from multiple sources** is efficiently imported and automatically organized as aggregated
- **Rich native feature set** - RavenDB comes with a comprehensive native feature set, minimizing the reliance on third-party plugins and eliminating the need for other databases.
- **You can code intelligent indexes** to run computations, including AI/ML models, to be performed in the background as data changes so that queries have turbo-charged speed
- **Integrates smoothly with SQL and OLAP** as well as other analytics tools such as Kafka, RabbitMQ, PowerBI, Grafana, Elasticsearch, via various ETL & replication options
- **Self-optimization** according to system usage and hardware
    - **Ideal for edge deployments** ~20K requests/second on Raspberry Pi or ARM64

These features add up to a database system trusted and employed by a global client base, including Fortune100 companies spanning several continents.

RavenDB is deployed in various systems, both on-prem and cloud, using the RavenDB Cloud DBaaS ManagedService on AWS, Azure, or Google Cloud hardware. Distributions vary from simple, single-server deployments to global systems of geo-distributed clusters to a major chain with over 1.5 million instances of Point of Sale (PoS)machines deployed worldwide.

MongoDB has been offered since 2009 as an open-source, high-performance document database. MongoDB takes its name from "humongous," referring to its intended usage for storing large data. It is used in applications ranging from simple TO-DO apps to critical business systems and is widely used and known in then on-relational database community. MongoDB spends a lot of money branding itself as the database that solves the object-relational

impedance mismatch problem, though every document database does this as well.

MongoDB began offering multi-document ACID transactions in 2018.

MongoDB integrates with Vercel and Netlify as application platforms. Several 3rd party plugins and MongoDB's "Connector" enable MongoDB to integrate with other systems.

Maintaining Data Integrity

How well does each database preserve data integrity, preventing data corruption or loss?

RavenDB is a fully transactional NoSQL database. It ensures data integrity with default ACID transactions throughout and across your database cluster so your data is safe. You can modify multiple documents in a single transaction and be assured that all changes will be persisted to disk or all of them will be rolled back for another attempt.

RavenDB can ensure that your transaction boundaries will be maintained when the data is replicated among the different nodes in the cluster. Transactions can be set to be cluster-wide for stronger consistency, though this mode is more time-consuming due to the need for Raft consensus. RavenDB's default transaction mode is on a single preferred node, which is then replicated asynchronously and atomically to other nodes in the database group of nodes.

RavenDB uses a multi-master model for high availability and instant, seamless failover if your preferred node goes down. In this case, all tasks handled by that server are instantly transferred to other nodes. This means your application(s) will have zero downtime with RavenDB, even if a server crashes or the network is disrupted.

RavenDB can integrate with a wide range of technologies, including most relational databases, OLAP solutions,Kafka, RabbitMQ, PowerBI, Grafana, and Elasticsearch. This integration simplifies both migrating to RavenDB as well as using hybrid solutions and greatly reduces the cost of data flow, and enhances cohesiveness in your systems and applications.

With RavenDB, you can maintain ACIDity throughout your current data architecture while enjoying the ability to scale up or out quickly. You can enjoy the speed, agility, and performance of a document database solution that queries rapidly via indexes while keeping the data integrity guarantees that many other NoSQL databases don't offer. RavenDB has been transactional from the very start, continually enhancing performance without compromising on ACID assurance, so you don't have to trade performance for data integrity.

MongoDB became a transactional database in 2018, announcing support for transactions covering multiple documents after a decade of supporting atomic operations on a single document only. Their challenge will be to improve upon their new ACID guarantees without sacrificing performance.

Multi-document ACID transactions with MongoDB come with performance costs. MongoDB transactions have to be set explicitly to ensure data integrity. Even to this day, MongoDB documentation calls out that transactions are more expensive than single document modifications and recommends changing your data model to avoid them if possible.

MongoDB also maintains high availability with trivial client downtime when the primary node goes down.

Querying & Aggregating Data

How fast can you query data?
How do you get aggregation results?

RavenDB is a leader in running complex queries, showcasing unparalleled efficiency. Typically, complex SQL queries that require multiple joins and take about 3000ms consistently require less than 30 ms for RavenDB to execute.

For one complex query, RavenDB is 100x more efficient than SQL. Duplicating that for a page that lists 30product options, *RavenDB is 3000x more efficient than SQL in complex queries*. How does RavenDB achieve this level of efficiency?

All queries made with RavenDB use an intelligent index. You never have to fear a full table scan or an unoptimized query, grinding your business to a halt. When you make a query, the query optimizer will detect whether a query can be answered with an existing index and will modify and optimize the index definition on the fly as needed.

If there is no suitable index, RavenDB will automatically create one and remove it if it isn't used for a specified time (30 minutes till idle and 72 hours till removal by default).

As you make more queries, the query optimizer learns and adapts your indexes to match your needs. RavenDB's auto-indexes free you from the need to predict and write indexes for every possible query scenario.

Latency is obliterated as query results come faster because RavenDB is not required to comb and process your data during every query to return results.

Once an index is set up, query times drop by over 99.9%, using precomputed results that are kept current for you behind the scenes as the data changes. This not only saves your users time, but in the cloud, it saves you money.

Your database administrator doesn't need to constantly monitor and adjust the database indexes to achieve outstanding performance. RavenDB already does that for you with its automatic indexes. For production usage, you can let RavenDB generate the optimal set of indexes you need to run your application completely automatically.

You can also apply the same learnings from a test or QA environment to production, allowing the RavenDB cluster to apply learned behavior about the next version of your system as part of your deployment.

There's no need to worry if you didn't explicitly set up indexes for aggregation and reports. When an aggregation query is sent to RavenDB, the query optimizer will create an (aggregation) auto index to answer it if one doesn't exist already.

Aggregation queries in RavenDB are inexpensive and require almost no work from developers or admins to get things working. As a native part of RavenDB, you do not need to maintain third-party components to perform aggregates. They happen automatically, behind the scenes, to keep the data in your queries fresh.

The nature of indexing in RavenDB means that RavenDB doesn't need to create more and more indexes as you query various aspects of your documents. Instead, the database is able to merge relevant indexes and answer multiple types of queries using a single index, greatly reducing the amount of work the database needs to do behind the scenes to answer your queries quickly.

On the other hand, developers can explicitly define static indexes that can make complex computations on your data whenever it is modified. The indexes then provide the pre-computed data to queries so that frequent queries don't need to do the work and are thus much faster. This is most noticeable when considering aggregation queries.

Unlike auto-indexes, user-defined indexes are not removed automatically if they're not used. The philosophy is that such indexes were created explicitly by the developers, so they should also be removed or edited explicitly. RavenDB has an Index Cleanup feature that analyzes index usage and suggests indexes that can be merged or removed to streamline CPU work and storage used by indexes.

When your documents are updated, RavenDB will update all the relevant indexes. Unlike most databases,RavenDB does that outside of the update transaction. This means your actual operations are much faster since they don't need to wait for the indexes to complete. RavenDB also takes advantage of this behavior to optimize index updates and merge multiple changes into a single operation.

Concurrent queries on those indexes get a choice, either read the current state of the index (potentially stale results) for faster dashboard rendering, showing whatever information is ready to serve, or wait until the indexes are up to date for rendering pages like the order history.

MongoDB supports dynamic queries, but not automatic indexes. MongoDB Atlas, the hosted version on the cloud, has a feature called Index Autopilot that can automatically build indexes out of their query performance suggestions. MongoDB still has no way to remove unnecessary indexes automatically.

This means that if they create indexes, but these aren't really used, they will continue to use system resource sunless they're explicitly removed. This can quickly turn into system overload and thus slow the system.

This typically results in acceptable performance initially, with small datasets, but quickly causes performance deterioration and system overload as the data size grows.

This can be exceptionally costly on the cloud and causes users to wonder why the app is so slow. Creating indexes ahead of time resolves this issue, but it is difficult to anticipate every future query scenario and write an index for each one. MongoDB's Performance Advisor is better than nothing, but it is like taking your delivery van in for repairs after it broke down. Wouldn't you rather have a system that optimizes itself in real-time according to actual usage?

Today, unless the administrator has taken steps to define indexes ahead of time or toggle and maintain the autopilot, queries will scan the entire database and filter results on documents each time. If an index isn't being used, it will continue to take system resources unless someone cleans up.

Even with Index AutoPilot enabled on MongoDB Atlas, indexes in MongoDB are not flexible enough to allow a single index to cover multiple different queries, easily leading to a great number of indexes being created and maintained, at a performance cost.

Indexing in MongoDB is single-purpose. It doesn't mean an index can only satisfy a single query. Rather, that MongoDB indexes are optimized for specific query patterns. Each index is built considering the fields that are queried upon, their order in the query, and the sort order.

Here are some examples to further explain:

Example 1: Filtering by Author and Date, then Sorting by Likes

```
db.Posts.find({ Author: $uid, Date: {
$gte: $start, $lte: $end } }).sort({
Likes: 1 })
```

Here, we are fetching all the posts of a particular author written in a specified time frame and then sorting the results based on the number of likes.

The optimal index for this query would be:

```
{ Author: 1, Date: 1 }
```

The index follows the exact order in which the fields are queried. Author is the primary field, followed by Date,and then finally, Likes is used for sorting (but cannot use an index).

Example 2: Filtering by Author and Likes, then Sorting by Date

```
db.Posts.find({ Author: $uid, Likes: {
$gte: $minLikes } }).sort({ Date: 1 })
```

In this scenario, posts by a specific author that have garnered a certain number of minimum likes are retrieved. These results are then sorted based on their publication date.

```
{ Author: 1, Likes: 1 }
```

Notice the subtle change? Even though Date and Likes are present in both queries, we cannot have both of them in a single index and take advantage of that.

A reasonable question one might ask is why not have a more generic index to satisfy both queries? The specificity of MongoDB indexes ensures that data retrieval is as fast as possible.

A generic index might not be as optimal because MongoDB reads indexes from left to right. If the index doesn't match the query pattern, some parts of the index might be skipped, making the query less efficient.

Understanding the nuances of MongoDB indexing is crucial for efficient data retrieval. While MongoDB provides flexibility, the onus is on the developers and database administrators to design the right indexes for their use cases. Properly indexed collections can greatly enhance the performance of the application, leading to a smoother user experience.

Unlike MongoDB, which often requires meticulous crafting of indexes to match query patterns, RavenDB can adapt its indexes to cater to a broader range of queries. The mechanism is designed to understand the nature of the data and optimize itself for multiple query scenarios.

Let's revisit our previous scenarios to see how RavenDB handles them:

Example 1: Filtering by Author and Date, then Sorting by Likes

```
from 'Posts'
where Author = $uid and Date between $start and $end
order by Likes
```

Example 2: Filtering by Author and Likes, then Sorting by Date

```
from 'Posts'
where Author = $uid and Likes => $minLikes
order by Date
```

RavenDB's dynamic indexing is an automatic process where the database creates indexes on-the-fly based on the queries it receives. As more queries are made, RavenDB tweaks and merges these dynamic indexes, optimizing them over time. This ensures efficient data retrieval without the need for manual intervention. In this specific scenario, RavenDB will first react to these two queries by creating an automatic index, and after that, the very same index will be used to serve both queries.

The internal structure of indexes in RavenDB is far more flexible than the one used by MongoDB, allowing RavenDB to utilize a single index to efficiently answer different queries on top of the same indexing structure.

For aggregation support, MongoDB provides both MapReduce queries and aggregation pipeline queries. These are more complex than simply using a GROUP BY statement, with MapReduce being more flexible and the aggregation pipeline being faster.

In both cases, MongoDB must evaluate all the matching documents and compute the final total. This happens on every query, resulting in workarounds such as spilling the results of a query to a temporary collection and refreshing that on a routine schedule.

In such cases, you must schedule refreshing the results during off hours and manage it manually. Maintaining the freshness of the results and the cost of refreshing the query require a significant investment of time and effort.

With RavenDB, aggregation queries are the responsibility of the database engine, not your operations teams. RavenDB will do the aggregation ahead of time and keep it continuously up to date, so aggregation queries are served in milliseconds instead of minutes, with no need for operational overhead.

How much "new stuff" must you learn to query each database?

For queries, RavenDB uses RQL (Raven Query Language), which is the equivalent of SQL in the context of the RavenDB Document Database. Like SQL, it is designed to be user-friendly for developers and non-developers. RQL gives you a human-readable, intuitive way to query the database, project results of complex or simple queries, and work with documents in RavenDB.

If you have any experience with SQL, you can understand the RQL syntax easily.

Learning to write queries in RQL with knowledge of SQL is like being a tennis pro and having to learn racquetball.

MongoDB supports queries using JavaScript and JSON - based query objects. This is powerful but can be unfriendly if you aren't a developer familiar with both MongoDB and JavaScript. Let's use a simple query to aggregate results from a Zip Code statistical data collection and get the states with more than 10 million residents and their populations.

Here is how you would need to write your query using SQL, RavenDB, and MongoDB:

SQL:

```
SELECT State, SUM(Population) AS TotalPop
FROM ZipCodes
GROUP BY State
HAVING TotalPop >= (10*1000*1000)
```

RQL (RavenDB Query Language):

```
FROM ZipCodes
GROUP BY State
WHERE SUM(Population) >= 10_000_000
SELECT SUM(Population) AS TotalPopulation, State
```

MongoDB Syntax:

```
db.zipcodes.aggregate([
  { $group: { _id: "$state", totalPop: { $sum: "$pop" } } },
  { $match: { totalPop: { $gte: 10 * 1000 * 1000 } } }
])
```

Source: *https://www.mongodb.com/docs/v6.0/tutorial/aggregation-zip-code-data-set/*

With MongoDB, you need to have a DBA review all queries and ensure that they don't put too much load on the server. You also need to define indexes ahead of time and re-validate your configuration on each deployment of your software.

With RavenDB, all of this is handled automatically behind the scenes as part of the notion that you should have as close to a zero - admin unattended database experience as possible in your application stack. RavenDB makes it possible for your devs to forget about your database and focus on your application.

Leveraging familiarity with SQL, RavenDB queries are simpler to read and understand, generate the appropriate set of indexes automatically, and allow your developers to reach production faster and with more time to actually invest in your core business features.

Performance

How fast can each database process your data?
How well does each handle Enterprise Level load?

With sustained low latency throughput, RavenDB can handle all your writes in a transactional manner with a speed of over 150,000 writes/second per node on commodity hardware (machines selling for less than $1,000) and exceed 1 million reads/second. RavenDB enjoys single-digit millisecond performance right up until you hit the limits of your hardware.

The RavenDB team keeps improving the database's performance from version to version, sustaining its rich and complex functionality along with ACID guarantees. RavenDB makes handling Big Data a small challenge. Using RavenDB on the cloud, whether in your own cloud, as a service with RavenDB Cloud, or in a hybrid architecture will save you time, resources, and money.

The standard performance test for RavenDB is to load the entire Stack Overflow dataset, which includes tens of millions of questions totaling over 50 GB of data. Currently, RavenDB accomplishes this task in less than five minutes, and our team keeps improving its performance.

Similar to RavenDB, MongoDB supports CRUD operations, simple updates, simple indexing, and both simple and aggregation queries at peak performance speeds. MongoDB is typically faster for non-concurrent, plain write-oriented tasks with no indexing. However, there are heavy prices for these shortcuts. With MongoDB, to achieve the best performance you need to sacrifice data integrity for speed by avoiding transactions. No indexing means slow queries, especially on large datasets. Your users feel slow queries. On the cloud, your wallet does as well.

MongoDB lags behind RavenDB in performance for aggregation queries, transaction support, and real data integrity. Furthermore, there are many operations that MongoDB does not support: Pre-computed map-reduce operations, complex patch operations, and non-trivial indexes and queries. All of these features make your system as a whole more efficient, agile, and able to provide your users with fast, complex data management. As the features you need become more advanced, the performance costs in MongoDB rise significantly.

How does each database's processing method push performance to the max?

The RavenDB native format is called Blittable JSON, a zero-overhead format designed for storing and processing JSON data. The RavenDB team developed the Blittable format to enable efficient document processing. To make this happen, we restructured how we save things to memory and on disk to make reading documents dirt cheap. This is one of the advantages of creating an all-in-one database, where each component is custom-made to seamlessly work in tandem with one another, maximizing overall performance.

The Blittable format allows RavenDB to avoid deserializing JSON objects when reading them from persistent storage. This saves lots of memory and CPU – especially on the cloud. Blittable can process data without first parsing the entire document into its object form. This reduces the costs of most operations in RavenDB significantly.

The Blittable format was designed to take advantage of the way RavenDB's storage engine works to streamline document processing and significantly simplify the amount of work RavenDB needs to do.

For example, instead of writing our own caching subsystems, RavenDB was designed to leverage the operating system's own page cache and access documents in such a way as to make optimal usage of the kernel's behavior. The kernel has more information about the state of the whole system, which allows RavenDB to be a better team player and share the system's resources instead of hogging them all.

MongoDB uses a format called BSON (Binary JSON) to store documents. Processing BSON is somewhat easier for a computer than processing JSON textual data. However, it is still a format that requires deserializing documents whenever you load them, increasing memory and CPU usage.

RavenDB's Blittable format means that it can access the documents directly in the operating system's page cache; MongoDB uses a separate memory (in addition to the page cache) and needs to deserialize the BSON documents whenever it accesses them.

Since its inception in 2009, and up to version 6.0, RavenDB has been using its own version of Lucene.net, which was modified to be ACID-compliant. Lucene is a proven and mature indexing engine, but in its essence, it is a full-text indexing solution optimized for processing single documents. In version 6.0, RavenDB introduced Corax, a new indexing engine built from scratch and tailored for batch processing of documents, which better suits real-life database usage scenarios. Unlike Lucene, which computes and holds data structures in memory, Corax stores them on disk. Consequently, Corax uses significantly less memory while eliminating long execution time on cold queries.

Scaling Out Your Database

How does each database maintain high availability and distribution of work?

RavenDB recommends setting up clusters of at least three servers, or nodes, to properly distribute your system's workload and provide zero downtime capability when a server goes down. RavenDB has a multi-master topology, where each server in a database group is always kept fully updated and is thus able to instantly and seamlessly take over all of the tasks required if the need arises.

RavenDB makes it easy to set up a cluster of multiple servers to act as nodes for your database group. Setting up a cluster is as simple as point and click in the RavenDB studio, and it's even easier with RavenDB Cloud, the hosted version. The cluster takes care of all the details of replicating data between nodes, ensuring sufficient copies of your data are kept, and dynamic load balancing and failover between the nodes in the cluster. No special network configuration or intricate load-balancing setup is required.

If one node fails, other nodes will continue to operate, and your users will have continuous access to your database with zero downtime! Once the faulted node is up again, one of the other nodes will replicate the most current state of data to it, keeping your information highly available with multiple copies.

When running in a cluster, RavenDB uses the multi-master model. When you make a write to any node in the cluster, that write will be accepted and then replicated to the rest of the cluster. RavenDB's multi-master model handles failure more gracefully because each node writes independently of the rest of

the cluster, and there is no period of unavailability if the cluster leader fails.

RavenDB features a built-in monitoring dashboard that highlights nodes that have gone down and provides insights into the root cause of the problem, enabling you to perform maintenance on your system faster and more efficiently. Assignment failover ensures that all outstanding tasks assigned to a downed node are evenly redistributed among the operational nodes in your cluster.

RavenDB supports Sharding your data across multiple nodes. Documents in a sharded database are stored in buckets, and each server is assigned a range of buckets. When storing documents, the cluster will execute a hash algorithm over the document ID, and based on the computed hash, it will automatically determine the bucket for a document. Replication is applied to sharded databases, so each shard will be replicated across multiple servers, ensuring automatic seamless failover with zero downtime.

RavenDB Sharding is completely server-bound, and when working with such a database, your experience will be identical to a non-sharded one. All implementation details, such as buckets, are hidden, allowing you to interact with documents just as you would in a non-sharded database. The cluster features an Orchestrator, serving as the first point of contact for your clients. The Orchestrator manages all complexities of coordination and execution, so from the perspective of your application, the database still appears to be one whole and seamless, even when it's broken up into multiple servers.

During the lifetime of your database, an even distribution of data and workload between all shards maintains a steadier overall usage of resources like disk space, memory, and bandwidth, improves availability, and eases database management. RavenDB provides users a resharding option - moving one bucket or range of buckets from one shard to another.

In other databases, users are typically forced to select a Partition Key upon database creation. The Partition Key consists of one or more fields and determines the shard where the document will be located. Once selected, it cannot be changed on the fly, which introduces significant risks as the database grows and your application adapts to requirements changes.

By default, RavenDB uses a document identifier as a Partition Key, but users also have the option to customize the partitioning strategy by "anchoring" similar documents together. Let's look at documents with the following IDs:

`customers/5` - customer
`orders/1$customers/5` - order for customer
`shipments/1$customers/5` - shipment for customer

RavenDB identifier is a string that can contain a dollar character. In a sharded database, the hashing function will consume only the part after the $ sign. As a result, all three documents will have the same hash code and end up in the same bucket. This anchoring approach is beneficial when executing queries, indexing, or fetching related documents, as it keeps them together and eases the load on the database.
MongoDB uses the primary-secondary replication process, where data is initially written to a single node, which then propagates the data to other nodes in the cluster. This can create a single choke point in the data architecture, and a primary node failure can stall the entire system while a different node is selected as the new primary.
To replicate, MongoDB uses the OpLog. This log captures the operations required for secondary nodes to execute, ensuring the replication of data to the

master state. If a failure occurs and there are enough writes to fill the OpLog, this can put your cluster into a permanently bad state and require admin intervention to recover.
MongoDB also offers sharding, but configuring, managing, and maintaining a sharded cluster is more complex than a standalone server or even a replica set. The process involves setting up multiple servers, mongos routers, and config servers. Choosing the right shard key is critical and can be tricky. A poor choice can lead to unbalanced data distribution (some shards having much more data than others), creating "hotspots". Once you've chosen a shard-key for a collection, you can only change it with a significant effort to dump, drop, and reload the data.

Features

How many third-party applications and plugins will you have to install along with each database?

RavenDB is a synergy of tailored components developed in-house to serve all your data needs in one place. It aims to minimize cost and complexity by providing a feature-rich database that covers all common scenarios directly. If you have a solution using various third-party components, you might need to go to several places for help, often hearing the support engineers say, "This is not our problem. Talk to the people who developed that."

Features you would usually have to plug in from somewhere else, like fast aggregation, ETL and Hub/Sink replication, full-text search, time-series, revisions, messaging or event sourcing, memory management, and more, are already part of RavenDB.

The RavenDB API includes server and client-side caching, adheres to best practices by default, and incorporates design patterns like Identity Map and Unit of Work. RavenDB packages everything in a well-defined and easy-to-use location, with all operations available both as scriptable commands and as part of a tailor-made GUI.

This is great for smaller businesses with limited development resources and time for IT. RavenDB's comprehensive solution ensures adherence to best practices and maintains a consistent and verifiable approach, making it well-suited for larger organizations. This is ideal for the cloud, where every add on can cost you money for every moment you use it.

MongoDB is more like a do-it-yourself solution. You'll need to purchase or find a MongoDB Admin GUI and get a BSON utility library to deal with everything BSON. About a dozen tools are provided with just the MongoDB installation alone. There are several dozen components for things like shell or backup and restore tooling. The number of tools you are required to compose to perform certain operations with MongoDB can be overwhelming, making it challenging to figure out the best approach to use in a given situation and identify available resources for specific tasks.

As a simple example, full-text search is typically handled by integrating MongoDB and Elastic together. That works, but at more than double the operational overhead and cost. In contrast, RavenDB offers complete full-text search capabilities out of the box, which means that you don't need to have a separate product to purchase, integrate, and monitor.

Integration With Your Organization's Current Systems

How well does each non-relational database work with SQL relational databases and the cloud to support hybrid data architectures?

There are many use cases where organizations based on a relational database chose RavenDB because it offers hyper-fast complex queries and is easier to integrate with their existing system. RavenDB's query language, RQL, is intentionally very similar to SQL. This makes it easy to learn, but they can also use the same logic that their system is built on with minimal translation. This saves a lot of development work, debugging, and maintenance.

RavenDB also has an SQL migration wizard that takes your relational data and creates a basic document template to collect it. It's a starting point to take in data from your relational database and enable you to model it in a non-relational form. Although it's usually unnecessary, documents can reference each other, and related documents can be called into the same session in only one trip to the server.

RavenDB supports automatic ETL (extract, transform, load) processes to relational and non-relational databases and databases on all cloud platforms. You don't need an outside application; its a core part of RavenDB. You can replicate the documents from your Non-Relational RavenDB Database to a relational SQL database. This empowers you to perform various analyses and reports on your data in a familiar environment using your existing reporting toolset.

A typical deployment pattern introduces RavenDB as a write-behind cache to a relational database. Another common deployment is to use RavenDB as part of a polyglot microservices architecture.

MongoDB has "connectors" that support pulling data from a relational database. The MongoDB BI connector translates SQL queries into MongoDB queries and returns them to its BI systems. MongoDB does not have a native "push" oriented service that continuously transfers data to a relational database for OLAP and reporting purposes. While external ETL services are available, it is important to note that external plugins can be buggy and prone to issues, especially after version updates.

MongoDB released its "Relational Migrator" in 2022, which imports data from tables, transforming them into a document model.

MongoDB's query language differs significantly from SQL, meaning integration will likely require a substantial amount of translation and re-coding.

Thanks to its commitment to smooth integration with relational systems for many years, RavenDB offers an import from SQL to documents, an ETL process to SQL, and a SQL-like query language called RQL. Altogether, these features make integrating with a relational database simple, efficient, and smooth.

How does each integrate with other top data services in a complex system?

OLAP

RavenDB has a native OLAP ETL, an ongoing task that automatically pushes changes in data to OLAP

databases and data lakes for additional business intelligence functionality.

MongoDB has a native connector for BI, which acts as a MySQL server for MongoDB data. MongoDB Connectors do not include the option of adding transform scripts in the process (the T in RavenDB's ETL).

Kafka / RabbitMQ

RavenDB provides bi-directional support for integration with Kafka & RabbitMQ. You can have RavenDB pull events from sinks and queues, transform them into documents, or publish events and messages from documents inside of the database. You need to define the policy on how that is done, and then RavenDB takes over the entire process, ensuring high availability, ongoing monitoring, and your peace of mind.

MongoDB also has connectors to Kafka and RabbitMQ. Again, you cannot include transform scripts, and a significant amount of integration time and effort to set up properly is required.

PowerBI

RavenDB has native PowerBI integration. You can export raw data from RavenDB collections to PowerBI. RQL queries can be executed directly in PowerBI to retrieve only selected data from RavenDB, which means PowerBI users can take advantage of the complete set of query capabilities offered by RavenDB, such as automatic indexing and aggregation queries. .

On top of that you can then apply the PowerBI features to slice and dice your data and create meaningful reports from the data directly.

MongoDB's BI connector tool can export data to PowerBI but does not support query execution.

Elasticsearch

MongoDB offers full-text search capabilities only as part of MongoDB Atlas, their cloud offering. Typical MongoDB deployments that necessitate full-text search often integrate with Elasticsearch.

The MongoDB connector to Elasticsearch can copy documents from MongoDB to Elasticsearch but will only send the full document contents, not just the details you care to enable search on.

RavenDB, on the other hand, provides full-text search capabilities directly out of the box, requiring no additional integration. If you are already using Elasticsearch and want to expose data from RavenDB to your existing search cluster, you can use the Elasticsearch ETL inside of RavenDB to replicate data to Elasticsearch.

In addition to specifying which collections will be sent, you have the flexibility to send just the relevant data to the search or even push aggregated information to the other side to reduce overall costs.

Caching and Concurrency

How does each database cache its data?

RavenDB incorporates both automatic and aggressive caching. RavenDB clients can cache data locally and let the server know that they have a cached version of the query they are making. An optimized code path

inside RavenDB then checks whenever there have been any modifications to the query. If there haven't been, the client will use the cached version.

This allows RavenDB to save a lot of bandwidth and cost since many queries stay mostly the same and can be served completely from the client cache while ensuring you always serve fresh information to the users. This saves you massive amounts of latency, especially on the cloud. Specifically on the cloud, this also reduces the data transfer costs in a measurable manner.

RavenDB also supports aggressive caching. Instead of the client asking the server if something has changed for every query, the client will ask the server to inform it only when there are any modifications on the server side. Until the client gets such a notification, it can serve results purely from its own cache, thus vastly reducing trips to the server. RavenDB doesn't just save you the query cost and the bandwidth transfer but eliminates the network round trip costs as well. Automatic and aggressive caching are also parts of the RavenDB Cloud (DBaaS) Managed Service.

MongoDB doesn't have comparable features to client-integrated caching or aggressive caching. External workarounds can be done to create a client-side cache with MongoDB. Doing so will improve your query speed, and assuming that the cache is successfully kept fresh, it is practical. Still, transferring data to various services and maintaining these integrations slow down DevOps and server/client performance.

How does each database handle concurrency?

RavenDB's in-house storage engine, Voron, was built explicitly to increase performance as concurrency grows. It uses an MVCC architecture to ensure that writers do not block readers and vice versa. With concurrent writes, RavenDB can merge multiple concurrent operations into a single disk operation, significantly reducing I/O costs and improving performance by large margins. RavenDB has been ACID by default from its inception. Data integrity is and has always been a key value in RavenDB.

For reads, RavenDB scales up directly to the number of nodes available to the system, as there is no need for locks or other concurrency controls to waste cycles on.

MongoDB uses multi-granularity locking. Locks are handled at multiple levels (server, database, collection, and document) and must be managed by MongoDB to ensure proper behavior. Lock management is typically expensive in databases, with locking & latching taking over 30% of the overall cost. The number of locks and their management can take up significant time when dealing with production scale load.

Data Security

How does each database protect your data?
How certain can you be that nobody will hack into your private information or destroy your reputation?

RavenDB can natively encrypt information in transit and at rest in your database. To safely guard your data on disk, RavenDB uses the XChaCha20, with 256-bit encryption. RavenDB uses X.509 certificates for authenticating access to your data and TLS 1.2 or higher for encrypting all communication between clients and servers.

Industry-standard security takes minutes to set up in an on-premises cluster and is a built-in and required feature on RavenDB Cloud. Still, you can start coding an application and run it in development without security, listening only on the local loopback device. As long as the database is used inside the local machine, you can usually ignore all security concerns (accepting no outside connections) and require no authentication.

If you set your database to listen to connections outside your local machine but didn't set up security properly, RavenDB will immediately block the vulnerable configuration and require the administrator to properly set up the security and access control to prevent unauthorized access to the data.

RavenDB will only let you expose your data outside your local machine once you adequately provide security. RavenDB also makes it easy to set itself up securely. You don't have to jump through hoops or go through reams of documentation. A friendly wizard will take you through the process of setting up a secure RavenDB cluster according to our best practices. That, combined with proactively preventing vulnerable configurations, ensures your data isn't left unsecured and exposed on the public Internet.

To set up MongoDB securely is a complex process. Evidence suggests that there are steps that are routinely skipped. Over 100,000 MongoDB databases have been compromised in recent years! There have been multiple high-profile instances of MongoDB databases being hijacked, where hackers wiped out client data and held businesses hostage by demanding payment to restore their databases.

Unlike RavenDB, MongoDB is not safe by default. To enable developers to download their database and start coding quickly, the configuration considers every database user an administrator. This is effective in allowing developers to focus on building their applications. However, once the application goes beyond one local machine and remote users can connect to the database, it is easy to miss a step and forget to lock the door. The outcome of this complexity has been lost records, critical data held for ransom, and leakage of sensitive user information from many organizations.

Data breaches typically cost millions of dollars to repair and are a blow to users' trust in an organization. Is data security something you'd rather believe that no human will make a mistake, or would you prefer a data platform designed and proven safe by default?

Getting Started

What do you get in the free version of each product?

RavenDB gives you a free cloud instance or an on-prem license suitable for low-throughput scenarios. Obtaining a free cloud instance is quick and straightforward, allowing you to start using RavenDB within minutes. RavenDB will handle all the back-end operations and enable you to focus exclusively on how your data works to enhance your application. To get started, claim your free instance at: *https://cloud.ravendb.net/*

A free on-prem license includes 3 cores, 3 nodes, and 6 gigabytes of RAM memory for your data. Your free license also comes with the RavenDB Studio, providing you with a GUI that makes RavenDB easy to use. RavenDB has a simple on-premises cluster setup wizard that can get you started in minutes. You can take a free license at: *https://ravendb.net/free*.

RavenDB bears most of the financial burden for on boarding and tech support. Also, the tech support team is staffed by the same engineers who built RavenDB. This is done on purpose to ensure that RavenDB remains as intuitive and trouble-free as possible.

While MongoDB allows you to set up a cluster in their free version, accessing critical features necessitates the purchase of premium packages. Many of their installation, configuration, and maintenance processes are complex, requiring tech support. Mentioned by both Gartner and Forrester Research, RavenDB is a pioneer in NoSQL database technology with over 2 million downloads and thousands of customers from startups to Fortune 100 Large Enterprises. Over 1,000 businesses use RavenDB for IoT, Big Data, Microservices Architecture, fast performance, a distributed data network, and everything needed to support a modern application stack for today's user. For more information, please visit *ravendb.net* or contact *info@ravendb.net*.

About RavenDB

RavenDB is a pioneer in NoSQL database technology with over 2 million downloads and thousands of customers from startups to Fortune 100 Large Enterprises.

Mentioned in both Gartner and Forrester research, over 1,000 businesses use RavenDB for IoT, Big Data, Microservices Architecture, fast performance, a distributed data network, and everything you need to support a modern application stack for today's user.

For more information please visit

ravendb.net

Contact us at

info@ravendb.net

Documentation
https://ravendb.net/learn/docs-guide

Use Cases
https://ravendb.net/news/use-cases

Free Online Training
https://ravendb.net/learn/bootcamp

Webinars
https://ravendb.net/learn/webinars

RavenDB Download
https://ravendb.net/download

RavenDB Cloud Database as a Service
https://cloud.ravendb.net/


RavenDB