Real-Time Analytics on MongoDB:

The Ultimate Guide

mongoDB®

ROCKSET

ROCKSET

Table of Contents

About This eBook

In the course of working with MongoDB users, we gathered and shared our experiences in a series of articles that addressed common challenges and solutions when implementing real-time analytics on MongoDB data. In this eBook, we have compiled them into a single resource—a developer's ultimate guide to evaluating the various options for real-time analytics on MongoDB.

This eBook includes:

- An Introduction to MongoDB: This provides its key benefits and use cases.
- The Challenge of Real-Time Analytics on MongoDB: The main roadblock to running analytics directly on MongoDB is the possible performance impact on business-critical operations on a production MongoDB cluster.
- Options for Real-Time Analytics on MongoDB: This covers the advantages and disadvantages of the following approaches:
    - Indexing in MongoDB: Indexing improves read performance but does not solve the issue of analytic queries impacting transactional workloads.
    - Replication in MongoDB: Using read replica sets can separate read and write concerns in the cluster but does not help improve the performance of analytic queries.
    - MongoDB + PostgreSQL: Some advantages of a PostgreSQL-MongoDB pipeline include native SQL queries and the ability to store data in a more relational format. However, it is not well-suited for less-structured data and has limited scalability.
    - MongoDB + Elasticsearch: There are various data-syncing tools available to integrate MongoDB with Elasticsearch and achieve additional analytical functionality. Issues to watch out for include the requirement to manage another cluster and the lack of support for SQL and joins.
    - MongoDB + Rockset: Rockset is a cloud-native, real-time indexing option for MongoDB data. It is best suited for higher-value complex analytics that require fast search, aggregations, and joins.

An Introduction to MongoDB

MongoDB is an open-source NoSQL database that uses JSON-like documents, made up of field-value pairs, to store high-volume data. The size, content, and structure of each document can be different, with no need for a predefined schema. The document model, which intuitively maps directly to objects within code, is a very natural and efficient way for developers to work with data.

Due to its flexibility and scale-out architecture, MongoDB is an excellent fit for many of today's applications in areas like ecommerce, gaming, IoT, and logistics that necessitate high volumes of data and traffic. Often the database of choice for startups, MongoDB is also used by many of the world's largest corporations across a wide range of industries.

The Challenge of Real-Time Analytics on MongoDB

MongoDB is widely used as an operational database, but there is typically a need for real-time analytics on MongoDB data as well. For intelligent applications to make optimal decisions or take the best actions, they need to run complex queries over large-scale data. While MongoDB has native aggregation capabilities that help support these use cases, there are often challenges associated with implementing real-time analytics on MongoDB.

Performance Impact on Transactional Workloads

MongoDB supports aggregation queries, but they are frequently CPU-intensive and can block or delay the execution of other queries. For example, transactional workloads are usually update-intensive or short read operations that have a direct impact on the user experience. If these operations are delayed because a read-heavy aggregation query is running on the MongoDB cluster, users will experience a slowdown. Thus, the unintended adverse performance impact on real-time operational applications running on the same cluster is a real problem for many users running data-intensive, CPU-bound analytics on MongoDB.

Lack of Support for SQL and Joins

MongoDB does not support querying using SQL, which can be a barrier to adoption among developers more familiar with SQL. Powerful SQL functions, like joins, are not available in MongoDB, so developers often have to create workarounds that require effort to code and maintain. Lack of SQL support also means that MongoDB cannot be used in conjunction with the large established ecosystem of SQL BI and visualization tools.

Requires Performance Engineering

While MongoDB provides the functionality required to perform analytic queries, getting good performance often requires performance engineering in the form of creating the right set of indexes, configuring replication and sharding, or modifying the data model. Not only do these tuning and optimization strategies require non-trivial effort, most also require knowledge of query patterns upfront.

Options for Real-Time Analytics on MongoDB

Having explored the challenges prevalent in running analytics on MongoDB, in this section we will consider several alternatives to implementing real-time analytics on MongoDB. Some of these methods are implemented within MongoDB itself, such as indexing and replication. Others are based on offloading heavy read operations, such as aggregations for analytics, to another layer and letting the MongoDB cluster handle only write and short-read operations. Typically, this means syncing the MongoDB data to systems that are optimized for running complex queries at high fidelity, such as PostgreSQL, Elasticsearch, or Rockset. In this section, we describe the advantages and disadvantages of each option.

Indexing in MongoDB

Indexing is one of the most common methods for improving read performance in any database, including relational ones.

When you index a table or collection, the database creates another data structure. This second data structure works like a lookup table for the fields on which you create the index. You can create a MongoDB index on just one document field or use multiple fields to create a complex or compound index.

Improving Read Performance with Indexing

The values of the fields selected for indexing will be used in the index. The database will then mark the location of the documents against those values. Therefore, when you search or query a document using those values, the database will query the lookup table first. The database will then extract the exact location of the document from this lookup table and fetch it directly from the location. Thus, MongoDB will not have to query the entire collection to get a single document. This, of course, saves a great deal of time.

But blindly indexing the data won't cut it. You should ensure you're indexing the data exactly the way you plan to query it. For example, suppose you have two fields, "name" and "email," in a collection called "users," and most of your queries use both fields to filter documents. In such cases, indexing both the "name" and "email" fields is not enough. You must also create a compound index with the fields.

In addition, you need to make sure that the compound index is created in the same order in which the queries filter the records. For example, if the queries are filtering first by "name" then by "email," the compound index needs to be created in the same order. If you reverse the order of the fields in the compound index, the MongoDB query optimizer will not select that index at all.

Furthermore, if there are other queries that use the "email" field alone to filter documents, you will have to create another index only on the "email" field. This is because the query optimizer will not use the compound index you created earlier.

To make sure the query optimizer is selecting the proper index, or the index that you prefer, you can use the hint() method in the query. This method allows you to tell the query optimizer which particular index to select for the query and not to decide on its own.

Key Considerations When Indexing in MongoDB

Despite the advantages of indexing, there are a number of issues that you must take into consideration when using this strategy:

- **Indexes should not be an afterthought:** It is important to design your queries and indexes in the earliest stages of the project. If you already have huge amounts of data in your collections, creating indexes on that data will take a long time, which could end up locking your collections and reducing performance, ultimately harming the performance of the application as a whole.

- **Managing indexes can be challenging:** If you change a query or the order of fields in a query, you'll need to update the indexes as well. While managing all of these indexes may seem easy at first, as your application grows and you add more queries, managing indexes can introduce significant overhead.

- **Indexes require additional memory and storage resources:** A working set in MongoDB comprises the actual document data and all of their associated indexes. Thus, indexes increase the storage resources required. Also, the entire working set has to fit inside the memory. The more indexes you have, the more memory required to ensure that the cluster does not fall out of memory range.

- **Indexes can reduce write performance:** Every update and insert has to look at which indexes are affected by the fields being updated and inserted. The indexes have to be updated for the write operation to take place.

- **New application queries:** If a new application query does not fit existing query patterns and indexes, MongoDB will conduct a full collection scan that can severely degrade the performance of the cluster or even bring it down.
- **Data shape that changes over time:** The MongoDB query planner chooses the index that most quickly returns the first 101 documents. Over time, however, the shape of the data may have changed in a way that a suboptimal index may return the first 101 documents faster but then do so more slowly on the rest of the query. Thus, indexes have to be added all the time to ensure a good fit to the data.

Replication in MongoDB

Another read-performance optimization technique that MongoDB offers out of the box is MongoDB replication. As the name suggests, these are replica nodes that contain the same data as the primary node. A primary node is the node that executes the write operations, and hence, offers the most up-to-date data.

Read replicas, on the other hand, follow the operations that are performed on the primary node and execute those commands to make the same changes to the data they contain. This means it's a given that there will be delays in the data getting updated on the read replicas.

Whenever data is updated on a primary node, it logs the operations performed to a file called the oplog (operations log). The read replica nodes "follow" the oplog to understand the operations performed on the data. Then, the replicas perform these operations on the data they hold, thereby replicating these same operations.

There is always a delay between the time data is written to the primary node and when it gets replicated on the replica nodes. Aside from that, you can command the MongoDB driver to execute all read operations on replica sets. Thus, no matter how busy the primary node is, your reads will be performed quickly, although you do need to ensure that your application is equipped to handle stale data.

MongoDB offers various read preferences when you're working with replica sets. For example, you can configure the driver to always read from the primary node. But when the primary node is unavailable, the MongoDB read preference can be configured to read from a replica set node.

And if you want the least possible network latency for your application, you can configure the driver to read from the "nearest" node. This nearest node could be either a MongoDB replica set node or the primary node. This will minimize any latency in your cluster.

The advantage of using read replica sets is that offloading all read operations to a replica set instead of the primary node can increase speed.

The major disadvantage of this, however, is that you might not always get the latest data. Also, because you are just scaling horizontally here, by way of adding more hardware to your infrastructure, there is no optimization taking place. This means that if you have a complex query performing poorly on your primary node, it would not see a major boost in performance even after adding a replica set. Therefore, it is recommended to use replica sets along with other optimization techniques.

MongoDB + PostgreSQL

PostgreSQL is an open-source relational database that has been around for almost three decades. PostgreSQL has been gaining a lot of traction recently because of its ability to provide both RDBMS-like and NoSQL-like features that enable data to be stored in traditional rows and columns while also providing the option to store complete JSON objects.

PostgreSQL has unique query operators that can be used to query key and value pairs inside JSON objects. This capability allows PostgreSQL to be used as a document database as well. Like MongoDB, it provides support for JSON documents. But, unlike MongoDB, it uses a SQL-like query language to query the JSON documents, allowing seasoned data practitioners to write ad hoc queries when required.

Also unlike MongoDB, PostgreSQL allows you to store data in a more traditional row-and-column arrangement. This way, PostgreSQL can act as a traditional RDBMS with powerful features, such as joins.

The unique ability of PostgreSQL to act as both an RDBMS and a JSON document store makes it a very good companion to MongoDB for offloading read operations.

Connecting PostgreSQL to MongoDB

MongoDB’s oplog is used to maintain a log of all operations being performed on data. It can be used to follow all the changes happening to the data in MongoDB and to replicate or mimic the data in another database, such as PostgreSQL, in order to make the same data available elsewhere for all read operations. Because MongoDB uses its oplog internally to replicate data across all replica sets, it is the easiest and most straightforward way of replicating MongoDB data outside of MongoDB.

If you already have data in MongoDB and want it replicated in PostgreSQL, export the complete database as JSON documents. Then, write a simple service that reads these JSON files and writes their data to PostgreSQL in the required format. If you are starting this replication when MongoDB is still empty, no initial migration is necessary, and you can skip this step.

After you've migrated the existing data to PostgreSQL, you'll have to write a service that creates a data flow pipeline from MongoDB to PostgreSQL. This new service should follow the MongoDB oplog and replicate the same operations in PostgreSQL that were running in MongoDB, similar to the process shown in Figure 1 below. Every change happening to the data stored in MongoDB should eventually be recorded in the oplog. This will be read by the service and applied to the data in PostgreSQL.

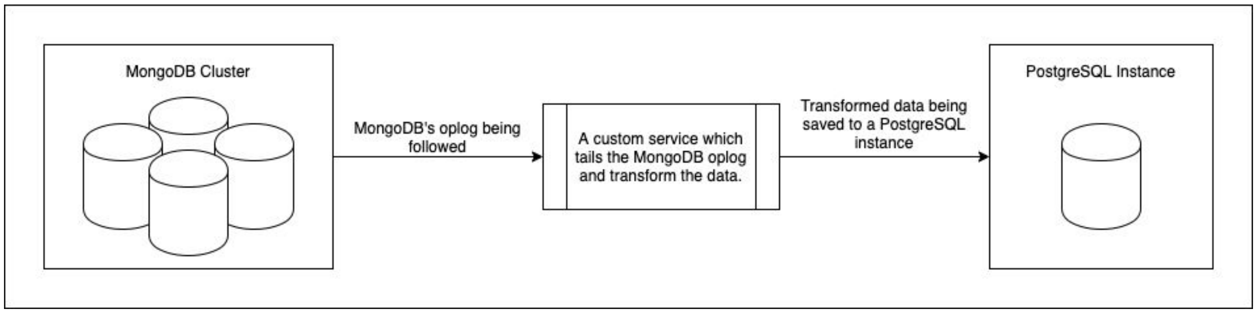
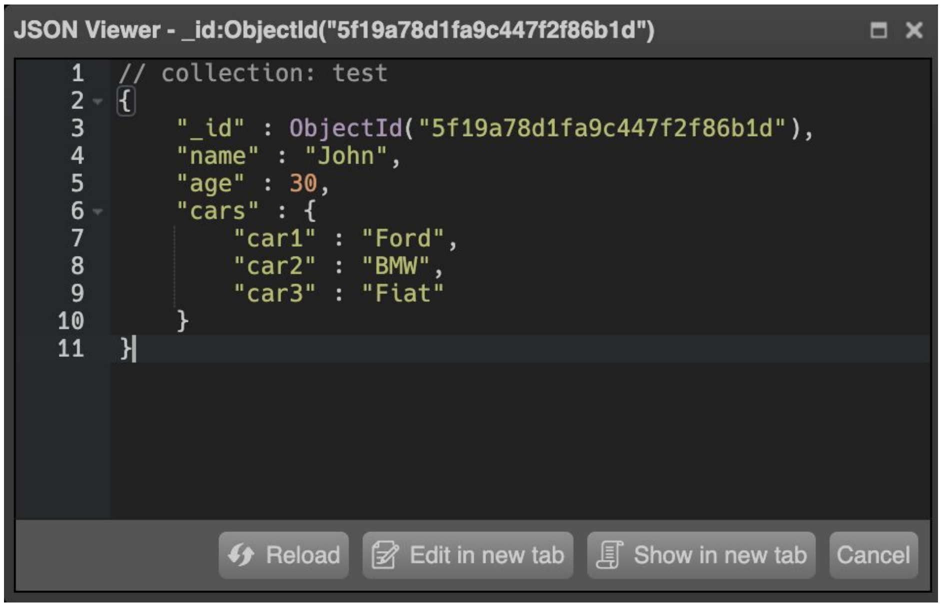

MongoDB Cluster
MongoDB's oplog being followed
A custom service which tails the MongoDB oplog and transform the data.
Transformed data being saved to a PostgreSQL instance
PostgreSQL Instance

Figure 1: A data pipeline that continuously copies data from MongoDB to PostgreSQL

Schema Options in PostgreSQL

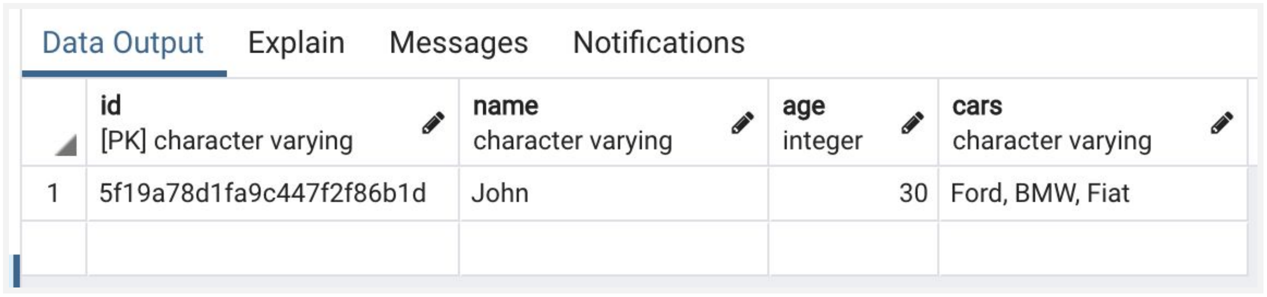
You now need to decide how you'll be storing data in PostgreSQL since the data from MongoDB will be in the form of JSON documents, as shown in Figure 2 below.



```
// collection: test
{
    "_id" : ObjectId("5f19a78d1fa9c447f2f86b1d"),
    "name" : "John",
    "age" : 30,
    "cars" : {
        "car1" : "Ford",
        "car2" : "BMW",
        "car3" : "Fiat"
    }
}
```

Figure 2: An example of data stored in MongoDB

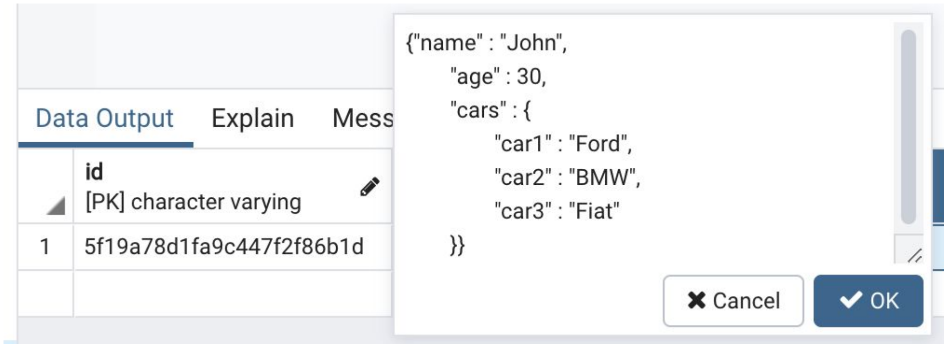
On the PostgreSQL end, you have two options. You can either store the complete JSON object as a column, or you can transform the data into rows and columns and store it in the traditional way, as shown in Figure 3 below. This decision should be based on the requirements of your application; there is no right or wrong way to do things here. PostgreSQL has query operations for both JSON columns and traditional rows and columns.



	id [PK] character varying	name character varying	age integer	cars character varying
1	5f19a78d1fa9c447f2f86b1d	John	30	Ford, BMW, Fiat

Figure 3: An example of data stored in PostgreSQL in tabular format

Once your migration service has the oplog data, it can be transformed according to your business needs. You can split one JSON document from MongoDB into multiple rows and columns or even multiple tables in PostgreSQL. Or, you can just copy the whole JSON document into one column in one table in PostgreSQL, as shown in Figure 4 below. What you do here depends on how you plan to query the data later on.



	id [PK] character varying
1	5f19a78d1fa9c447f2f86b1d

```
{"name" : "John",
      "age" : 30,
      "cars" : {
            "car1" : "Ford",
            "car2" : "BMW",
            "car3" : "Fiat"
      }}
```

Figure 4: An example of data stored in PostgreSQL as a JSON column

Getting Data Ready for Querying in PostgreSQL

Now that your data is being replicated and continuously updated in PostgreSQL, you'll need to make sure that it's ready to take over read operations. To do so, figure out what indexes you need to create by looking at your queries and making sure that all combinations of fields are included in the indexes. This way, whenever there's a read query on your PostgreSQL database, these indexes will be used and the queries will be performant. Once all of this is set up, you're ready to route all of your read queries from MongoDB to PostgreSQL.

The Advantages of Using PostgreSQL for Real-Time Reporting and Analytics

There are many advantages of using PostgreSQL to offload read operations from MongoDB. To begin with, you can leverage the power of the SQL query language. Even though there are some third-party services that provide a MongoDB SQL solution, they often lack features that are essential either for MongoDB users or SQL queries.

Another advantage, if you decide to transform your MongoDB data into rows and columns, is the option of splitting your data into multiple tables in PostgreSQL to store it in a more relational format. Doing this will allow you to use PostgreSQL's native SQL queries instead of MongoDB queries. Once you split your data into multiple tables, you'll obviously have the option to join tables in your queries to do more with a single query. And, if you have joins and relational data, you can run complex SQL queries to perform a variety of aggregations. You can also create multiple indexes on your tables in PostgreSQL for better-performing read operations. Keep in mind that there is no elegant way to join collections in MongoDB. However, this doesn't mean that MongoDB aggregations are weak or are missing features.

Once you have a complete pipeline set up in PostgreSQL, you can easily switch the database from MongoDB to PostgreSQL for all of your aggregation operations. At this point, your analytic queries won't affect the performance of your primary MongoDB database because you'll have a completely separate setup for analytic and transactional workloads.

The Disadvantages of Using PostgreSQL for Real-Time Reporting and Analytics

While there are many advantages to offloading your read operations to PostgreSQL, a number of trade-offs come along with the decision to take this step.

To begin with, there's obviously the new moving part in the architecture you will have to build and maintain—the data pipeline that follows MongoDB's oplog and recreates it at the PostgreSQL end. If this one pipeline fails, data replication to PostgreSQL stops, creating a situation where the data in MongoDB and the data in PostgreSQL are not the same. Depending on the number of write operations happening in your MongoDB cluster, you might want to think about scaling this pipeline to avoid it becoming a bottleneck. The pipeline has the potential to become the single point of failure in your application.

There can also be issues with data consistency because it takes anywhere from a few milliseconds to several seconds for the data changes in MongoDB to be replicated in PostgreSQL. This lag time could easily go up to minutes if your MongoDB write operations experience a lot of traffic.

Because PostgreSQL, which is an RDBMS, is your read layer, it might not be the best fit for all applications. For example, in applications that process data originating from a variety of sources, you might have to use a tabular data structure in some tables and JSON columns in others. Some of the advantageous features of an RDBMS, such as joins, might not work as expected in these situations. In addition, offloading reads to PostgreSQL might not be the best option when the data you're dealing with is highly unstructured. In this case, you'll again end up replicating the absence of structure even in PostgreSQL.

Finally, it's important to note that PostgreSQL was not designed to be a distributed database. This means there's no way to natively distribute your data across multiple nodes. If your data is reaching the limits of your node's storage, you'll have to scale up vertically by adding more storage to the same node instead of adding more commodity nodes and creating a cluster. This necessity might prevent PostgreSQL from being your best solution.

Before making the decision to offload your read operations to PostgreSQL—or any other SQL database, for that matter—make sure that SQL and RDBMS are good options for your data.

Considerations for Offloading to PostgreSQL

If your application works mostly with relational data and SQL queries, offloading all of your read queries to PostgreSQL allows you to take full advantage of the power of SQL queries, aggregations, joins, and all the other features described in this eBook. But, if your application deals with a lot of unstructured data coming from a variety of sources, this option might not be a good fit.

It's important to decide whether or not you want to add an extra read-optimized layer early on in the development of your project. Otherwise, you'll likely end up spending a significant amount of time and money creating indexes and migrating data from MongoDB to PostgreSQL at a later stage. The best way to handle the migration to PostgreSQL is by first moving small pieces of your data to PostgreSQL and testing your application's performance. If it works as expected, you can continue the migration in small pieces until, eventually, the complete project has been migrated.

If you're collecting structured or semi-structured data that works well with PostgreSQL, offloading read operations to PostgreSQL is a great way to avoid impacting the performance of your primary MongoDB database.

If you've made the decision to offload reporting and analytics from MongoDB for the reasons discussed above but have more-complex scalability requirements or less-structured data, you may want to consider other real-time databases, such as Elasticsearch and Rockset. Both Elasticsearch and Rockset are scale-out alternatives that allow schemaless data ingestion and leverage indexing to speed up analytics.

MongoDB + Elasticsearch

Offloading analytics from MongoDB establishes a clear isolation between write-intensive and read-intensive operations. Elasticsearch is one tool to which reads can be offloaded, and, because both MongoDB and Elasticsearch are NoSQL in nature and offer similar document structure and data types, Elasticsearch can be a very effective choice for this purpose. In most scenarios, MongoDB can be used as the primary data storage for write-only operations and as support for quick data ingestion. In this situation, you only need to sync the required fields in Elasticsearch with custom mappings and settings to get all the advantages of indexing.

Here, we'll examine the various tools that can be used to sync data between MongoDB and Elasticsearch. We will also discuss the various advantages and disadvantages of establishing data pipelines between MongoDB and Elasticsearch to offload read operations from MongoDB.

Tools to Sync Data Between Elasticsearch and MongoDB

When setting up a data pipeline between MongoDB and Elasticsearch, it's important to choose the right tool.

First of all, you need to determine if the tool is compatible with the MongoDB and Elasticsearch versions you are using. Additionally, your use case might affect the way you set up the pipeline. If you have static data in MongoDB, you may need a one-time sync. However, a real-time sync will be required if continuous operations are being performed in MongoDB and all of them need to be synced. Finally, you'll need to consider whether or not data manipulation or normalization is needed before data is written to Elasticsearch.

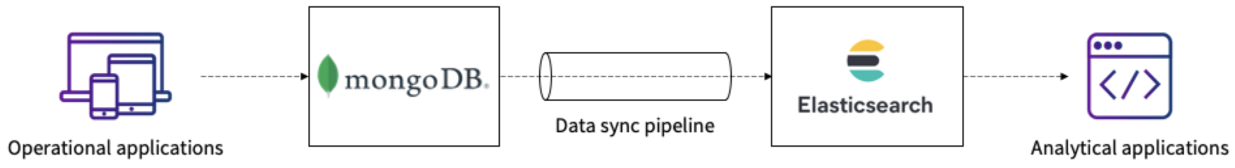


Figure 5: Using a pipeline to sync MongoDB to Elasticsearch

If you need to replicate every MongoDB operation in Elasticsearch, you'll need to rely on MongoDB oplogs (which are capped collections), and you'll need to run MongoDB in cluster mode with replication on. Alternatively, you can configure your application in such a way that all operations are written to both MongoDB and Elasticsearch instances with guaranteed atomicity and consistency.

With these considerations in mind, let's take a look at some tools that can be used to replicate MongoDB data to Elasticsearch:

- Monstache is one of the most comprehensive libraries available to sync MongoDB data to Elasticsearch. Written in Go, it supports up to and including the latest versions of MongoDB and Elasticsearch. Monstache is also available as a sync daemon and a container.

- Mongo-Connector, which is written in Python, is a widely used tool for syncing data between MongoDB and Elasticsearch. However, it only supports Elasticsearch through version 5.x and MongoDB through version 3.6.

- Mongoosastic, written in NodeJS, is a plugin for Mongoose, a popular MongoDB data modeling tool based on ORM. Mongoosastic simultaneously writes data in MongoDB and Elasticsearch. No additional processes are needed for it to sync data.

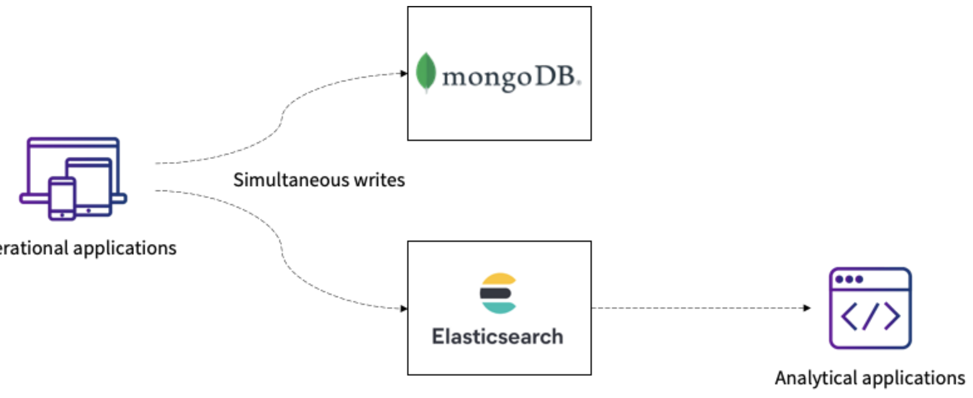


Figure 6: Writing simultaneously to MongoDB and Elasticsearch

- Logstash JDBC Input Plugin is Elastic's official tool for integrating multiple input sources and facilitating data syncing with Elasticsearch. To use MongoDB as an input, you can employ the JDBC input plugin, which uses the MongoDB JDBC driver as a prerequisite.

If the tools described above don't meet your requirements, you can write custom scripts in any of the preferred languages. Remember that sound knowledge of both the technologies and their management is necessary to write custom scripts.

The Advantages of Offloading Analytics to Elasticsearch

By syncing data from MongoDB to Elasticsearch, you remove load from your primary MongoDB database and leverage several other advantages offered by Elasticsearch, some of which are listed here below:

- **Reads Don’t Interfere with Writes:** In most scenarios, reading data requires more resources than writing. For faster query execution, you may need to build indexes in MongoDB, which not only consumes a lot of memory but also slows down write speed.

- **Additional Analytical Functionality:** Elasticsearch is a search server built on top of Lucene that stores data in a unique structure known as an inverted index. Inverted indexes are particularly helpful for full-text search and document retrievals at scale. They can also perform aggregations and analytics and, in some cases, provide additional services not offered by MongoDB. Common use cases for Elasticsearch analytics include real-time monitoring, APM, anomaly detection, and security analytics.

- **Multiple Options to Store and Search Data:** Another advantage of putting data into Elasticsearch is the possibility of indexing a single field in multiple ways by using mapping configurations. This feature assists in storing multiple variations of a field that can be used for different types of analytical queries.

- **Better Support for Time Series Data:** In applications that generate a huge volume of data, such as IoT applications, achieving high performance for both reads and writes can be a challenging task. Using MongoDB and Elasticsearch in combination can be a useful approach in these scenarios since it is then very easy to store the time series data in multiple indices (such as daily or monthly indices) and search those indices’ data via aliases.

- **Flexible Data Storage and an Incremental Backup Strategy:** Elasticsearch supports incremental data backups using the _snapshot API. These backups can be performed on the file system or on cloud storage directly from the cluster. This feature deletes the old data from the Elasticsearch cluster once the backup is taken. Whenever access to old data is necessary, it can easily be restored from the backups using the _restore API. This allows you to determine how much data should be kept in the live cluster and also facilitates better resource assignments for the read operations in Elasticsearch.

- **Integration with Kibana:** Once you put data into Elasticsearch, it can be connected to Kibana, which makes it easy to explore the data as well as build visualizations and dashboards.

The Disadvantages of Offloading Analytics to Elasticsearch

While there are a lot of advantages to indexing MongoDB data into Elasticsearch, there are some potential disadvantages you should be aware of as well:

- **Building and Maintaining a Data Sync Pipeline:** Whether you use a tool or write a custom script to build your data sync pipeline, maintaining consistency between the two data stores is always a challenging job. The pipeline can go down or simply become hard to manage due to several reasons, such as either of the data stores shutting down or any data format changes in the MongoDB collections. If the data sync relies on MongoDB oplogs, optimal oplog parameters should be configured to make sure that data is synced before it disappears from the oplogs. Also, when you need to use many Elasticsearch features, complexity can increase if the tool you're using is not customizable enough to support the necessary configurations, such as custom routing, parent-child or nested relationships, indexing referenced models, and converting dates to formats recognizable by Elasticsearch.

- **Data Type Conflicts:** Both MongoDB and Elasticsearch are document-based and NoSQL data stores. Both of these data stores allow dynamic field ingestion. However, MongoDB is completely schemaless in nature, and Elasticsearch, despite being schemaless, does not allow different data types of a single field across documents inside an index. This can be a major challenge if the schema of MongoDB collections is not fixed. Thus, it's always advisable to define the schema in advance for Elasticsearch to avoid conflicts that can occur while indexing the data.

- **Data Security:** MongoDB is a core database and comes with fine-grained security controls, such as built-in authentication and user creations based on built-in or configurable roles. Elasticsearch does not provide such controls by default. Although it is achievable in the X-Pack version of Elastic Stack, it's hard to implement the security features in free versions.

- **The Difficulty of Operating an Elasticsearch Cluster:** Elasticsearch is hard to manage at scale, especially if you're already running a MongoDB cluster and setting up the data sync pipeline. Cluster management, horizontal scaling, and capacity planning come with some limitations. Challenges arise when the application is write-intensive and the Elasticsearch cluster does not have enough resources to cope with that load. Once shards are created, they can't be increased on the fly. Instead, you need to create a new index with a new number of shards and perform reindexing, which is tedious.

- **Memory-Intensive Process:** Elasticsearch is written in Java and writes data in the form of immutable Lucene segments. This underlying data structure causes these segments to continue merging in the background, which requires a significant amount of resources. Heavy aggregations also cause high memory utilization and may cause out-of-memory (OOM) errors. When these errors appear, cluster scaling is typically required, which can be a difficult task if you have a limited number of shards per index or budgetary concerns.

- **No Support for Joins:** Elasticsearch does not support full-fledged relationships and joins. It does support nested and parent-child relationships, but they are usually slow to perform or require additional resources to operate. If your MongoDB data is based on references, it may be difficult to sync the data in Elasticsearch and write queries on top of them.

- **Deep Pagination Is Discouraged:** One of the biggest advantages of using a core database is that you can create a cursor and iterate through the data while performing sort operations. However, Elasticsearch's normal search queries don't allow you to fetch more than 10,000 documents from the total search result. Elasticsearch does have a dedicated scroll API to achieve this task, although it, too, comes with limitations.

- **Uses Elasticsearch DSL:** Elasticsearch has its own query DSL, but you need a good hands-on understanding of its pitfalls to write optimized queries. While you can also write queries using Lucene Syntax, its grammar is tough to learn, and it lacks input sanitization. Elasticsearch DSL is not compatible with SQL visualization tools and, therefore, offers limited capabilities for performing analytics and building reports.

Considerations for Offloading to Elasticsearch

If your application is primarily performing text search, Elasticsearch can be a good option for offloading reads from MongoDB. However, this architecture requires an investment in building and maintaining a data pipeline between the two tools.

The Elasticsearch cluster also requires considerable effort to manage and scale. If your use case involves analytics—such as filters, aggregations, and joins—then Elasticsearch may not be your best solution. In these situations, Rockset, a real-time indexing database, may be a better fit. It provides both a native connector to MongoDB and full SQL analytics, and it's offered as a fully managed cloud service.

MongoDB + Rockset

Rockset is a real-time indexing database that is able to sync data from common data sources, like MongoDB, and automatically build indexes on your documents. Rockset provides a fully managed connector that continuously ingests and incrementally indexes data from MongoDB using MongoDB change streams.

All documents stored in a Rockset collection are mutable and can be updated at the field level, even if these fields are deeply nested inside arrays and objects. Updates only reindex those fields in a document that are modified while keeping the rest of the fields in the document untouched. This is a highly efficient way to perform a fast ingestion from MongoDB change streams.

Below, we'll describe what the developer experience might look like when using the Rockset + MongoDB combination.

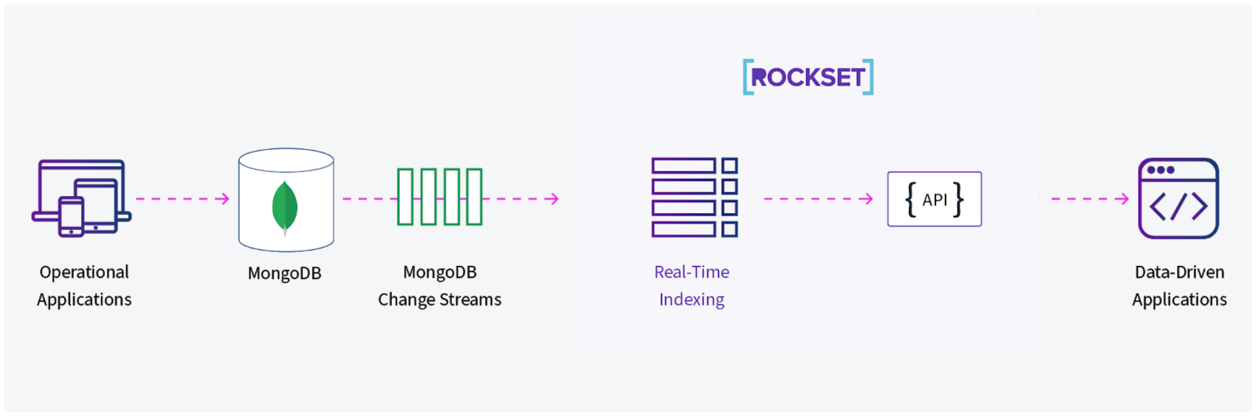
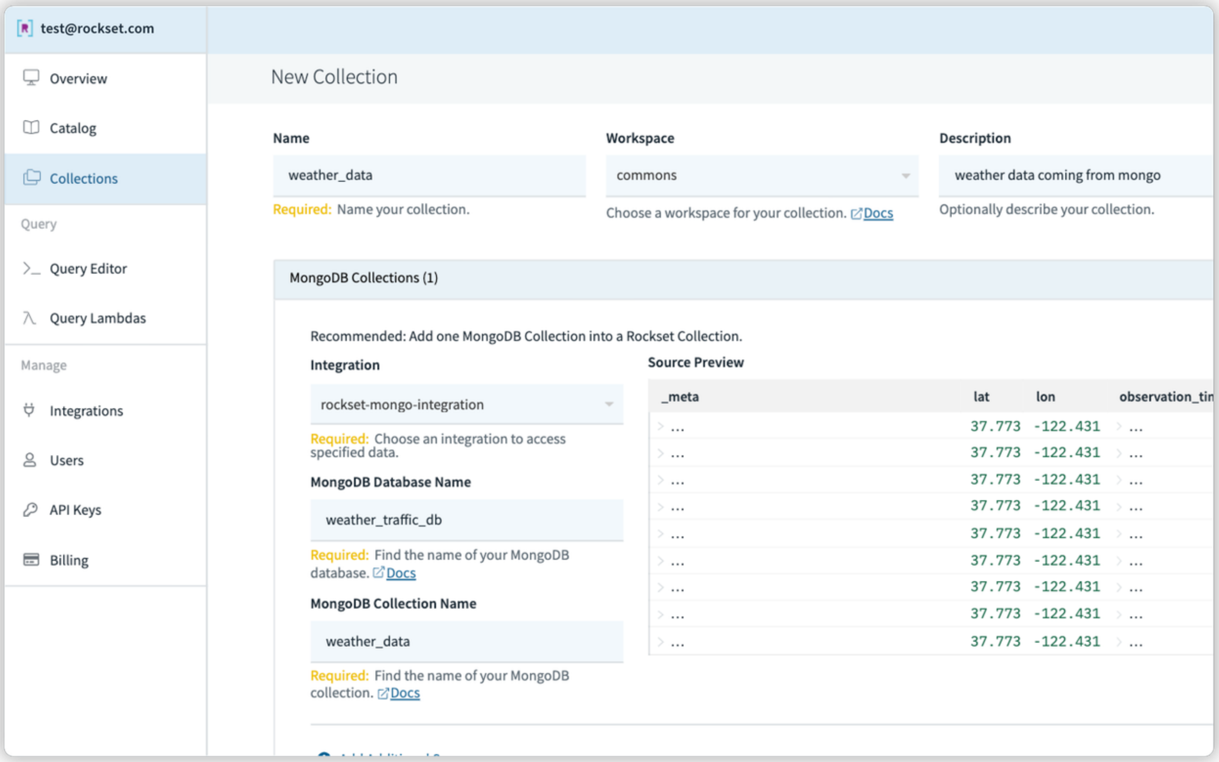


Figure 7: Rockset indexes data from MongoDB to power data apps

Building an External Index on a MongoDB Collection

Once an integration between MongoDB and Rockset is created, you can create Rockset collections backed by MongoDB. Specify the source database and collection in MongoDB—each MongoDB collection maps to one Rockset collection—and you will see a preview of your data, as shown in Figure 8.



New Collection

Name: weather_data
Required: Name your collection.

Workspace: commons
Choose a workspace for your collection. Docs

Description: weather data coming from mongo
Optionally describe your collection.

MongoDB Collections (1)

Recommended: Add one MongoDB Collection into a Rockset Collection.

Integration: rockset-mongo-integration
Required: Choose an integration to access specified data.

MongoDB Database Name: weather_traffic_db
Required: Find the name of your MongoDB database. Docs

MongoDB Collection Name: weather_data
Required: Find the name of your MongoDB collection. Docs

Source Preview

_meta	lat	lon	observation_tin
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...
> ...	37.773	-122.431	> ...

Figure 8: Preview of MongoDB collection in Rockset

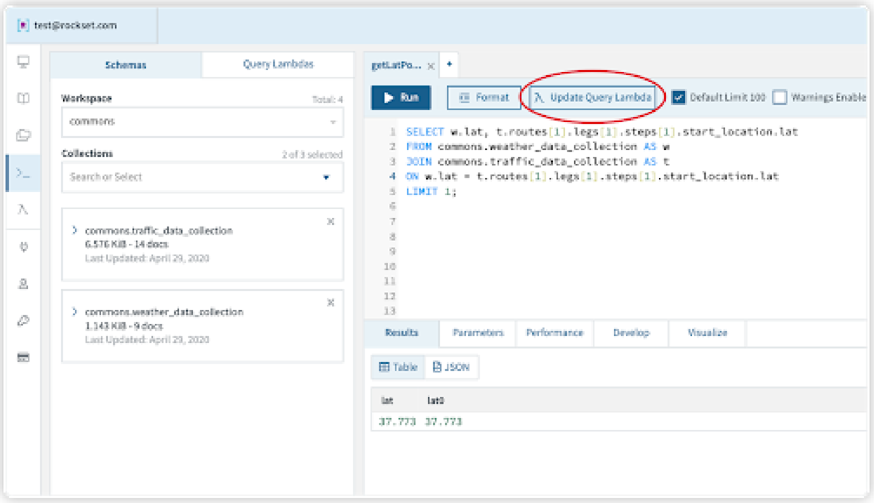
Rockset will first perform a full scan of the MongoDB data and, subsequently, stay in sync with any updates to the collection through a MongoDB change stream.

Rockset automatically indexes all the ingested data, acting as an external index for your MongoDB data. Rockset uses a concept called a Converged Index™ to build multiple indexes—a search index, a column index, and a row index—on every field of your data. This allows for fast SQL queries, including search-style queries, aggregations, and joins across different data sets, all without having to manually create a schema.

Creating APIs Powered by SQL on MongoDB Data

Now that your data from MongoDB is being continuously ingested and indexed by Rockset, you can construct your application using Query Lambdas—saved SQL queries accessed via REST endpoints.

Start by constructing queries on the collection you created from your MongoDB integration. You can even bring in data from other Rockset collections, created from other MongoDB collections or data sources. We use an SQL JOIN to do so in the example shown in Figure 9. From there, you can create an API endpoint from the query you wrote.



Workspace Total: 4
commons
Collections 2 of 3 selected
Search or Select

commons.traffic_data_collection
6.576 KiB - 14 docs
Last Updated: April 29, 2020

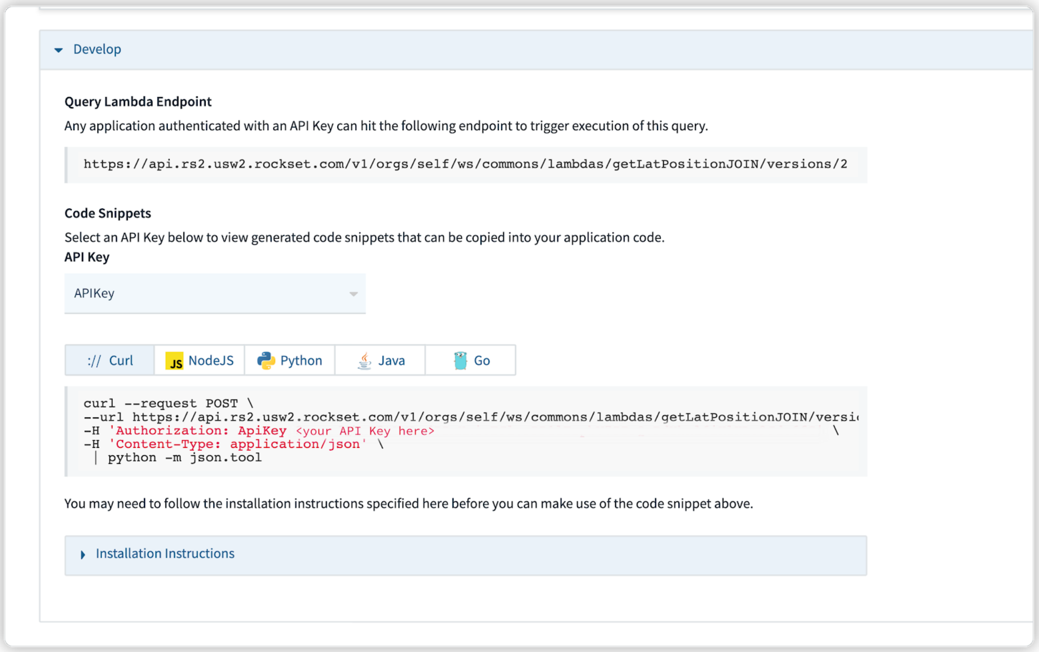
commons.weather_data_collection
1.143 KiB - 9 docs
Last Updated: April 29, 2020

```sql
SELECT w.lat, t.routes[1].legs[1].steps[1].start_location.lat
FROM commons.weather_data_collection AS w
JOIN commons.traffic_data_collection AS t
ON w.lat = t.routes[1].legs[1].steps[1].start_location.lat
LIMIT 1;
```

```
lat     lat0
37.773  37.773
```

Figure 9: Sample Lambda Query using JOIN to aggregate collections

Name and describe the Query Lambda so that you know what it's being used for. You'll also get code snippets to embed in your application to make this API call, as shown in Figure 10.



Query Lambda Endpoint

Any application authenticated with an API Key can hit the following endpoint to trigger execution of this query.

```
https://api.rs2.usw2.rockset.com/v1/orgs/self/ws/commons/lambdas/getLatPositionJOIN/versions/2
```

Code Snippets

Select an API Key below to view generated code snippets that can be copied into your application code.

API Key

APIKey

```
curl --request POST \
--url https://api.rs2.usw2.rockset.com/v1/orgs/self/ws/commons/lambdas/getLatPositionJOIN/versi
-H 'Authorization: ApiKey <your API Key here>                                                  \
-H 'Content-Type: application/json' \
 | python -m json.tool
```

You may need to follow the installation instructions specified here before you can make use of the code snippet above.

Figure 10: Code snippets to be copied into the application code

Executing a Query Lambda in your Python application will look something like this:

```
import requests, json

def executeLambda():
    r = requests.post('https://api.rs2.usw2.rockset.com/v1/orgs/self/ws/commons/lambdas/...',
    headers={'Authorization': 'ApiKey API_KEY'})
    return r.json()
```

Considerations for Offloading to Rockset

Using Rockset to index and create APIs on MongoDB data allows you to:

- **Perform real-time search, aggregations, and joins:** Use the full capabilities of SQL to express different types of queries. Since all of your data is indexed, you get millisecond-latency APIs to power your applications.

- **Simplify application logic:** Save on writing application-side logic when you are able to join MongoDB and other data sets using SQL JOINs. In addition, you just need to call a REST endpoint in your code to trigger a query, instead of embedding the entire query.

- **Ship features faster:** Productionize your queries as version-controlled REST endpoints as part of Rockset's standard query development flow.

- **Manage less infrastructure:** Build serverless applications that auto-scale as needed.

Like in the other scenarios where data is replicated to another system or read replica, Rockset may not be appropriate for use cases that cannot tolerate a delay between MongoDB and the secondary, analytical system. While Elasticsearch and Rockset both take an indexing approach to speeding up analytics, Rockset automatically builds multiple indexes on its collections as opposed to Elasticsearch, which is primarily a search index. Thus, Elasticsearch may be a better fit if your use case is heavily biased towards text search and log analytics. Rockset is also a cloud-only product, so you would need to consider other alternatives if your use case requires an on-premises solution.

For most use cases, Rockset is a good fit for offloading analytics from MongoDB in order to provide performance isolation between transactional and analytical workloads. It offers the ability to query using SQL and join functionality, like PostgreSQL does, while being able to scale out like Elasticsearch. A major benefit of being in the cloud is that Rockset can be offered as a fully managed solution, so users are spared the overhead of managing infrastructure, scaling, performance, and upgrades. The availability of a built-in MongoDB-Rockset connector also obviates the need for users to build and maintain their own data pipeline between the two systems.

Conclusion

While it is possible to run analytics on MongoDB, it is often not straightforward to do so on production MongoDB clusters that support transactional workloads, which are typically update-intensive or short read operations. If read-heavy analytical queries are run directly on the MongoDB cluster, users will often experience a slowdown in the transactional workloads, which they would normally expect to be real-time.

There are two solutions that can be implemented in MongoDB itself

- Indexing, which speeds up analytics but does not mitigate the performance impact of analytical workloads interfering with transactional operations.
- Replication, which uses read replica sets to separate read and write concerns in the cluster but does not improve the performance of analytic queries.

Other solutions involve the replication of MongoDB data so that analytic workloads can be performed in an external system such as PostgreSQL, Elaticsearch, or Rockset. Pairing MongoDB with a secondary system dedicated to analytics achieves better performance isolation and is a best practice in situations where MongoDB reliability is mission-critical. When choosing how to offload analytics for a specific application, the developer must consider their pros and cons along the dimensions of simplicity, scalability, and the performance of queries on complex analytics.

Learn more about how Rockset's cloud-native, real-time indexing accelerates complex analytics on MongoDB data, without impacting the performance of transactional operation.

About Rockset

Rockset is a real-time indexing database service for serving low latency, high concurrency analytical queries at scale. It builds a Converged IndexTM on structured and semi-structured data from OLTP databases, streams and lakes in real time and exposes a RESTful SQL interface.

Find out more at rockset.com