WHITE PAPER

Reap the benefits of MongoDB without the worry of production challenges

Building modern applications with ObjectRocket for MongoDB

Table of Contents

ObjectRocket

Introduction

How an organization stores, manages, analyzes, and uses data has a direct impact on its success. Database decisions directly affect how quickly new applications and features go to market, support business growth, and improve customer experience while enhancing business agility and reducing costs.

Over the last decade, companies have shifted their workloads to NoSQL (or non-relational) databases because the old relational model no longer meets the needs of their application or their development teams. There are three very compelling motivations for this change in infrastructure. First, the **need to handle new, multi-structured data types or scale beyond the capacity constraints of existing systems**. Second, the **desire to identify viable alternatives to expensive proprietary database software and hardware**. A third motivation is **agility and speed of development**, as companies look to adapt to the market more quickly and more fully embrace agile development methodologies.

Open source NoSQL databases, such as MongoDB, adapt easily and are best suited to manage the needs of modern application development. However, as with any evolving technology, NoSQL databases require experience and expertise to effectively manage, scale, and optimize the database for the desired application.

This paper aims to help organizations understand what MongoDB is, how it's used, and the complexities that arise as these production instances of MongoDB grow. We describe the importance of expertise as it relates to managing, scaling, upgrading, and optimizing these databases to help organizations determine if they are prepared to manage these database challenges in-house or if it makes more sense to outsource these tasks to a database as a service company.

ObjectRocket
a rackspace company

Who is ObjectRocket?

Since 2012, ObjectRocket has managed thousands of open source databases for companies of all sizes. ObjectRocket provides a database platform that's purpose-built for speed, easy scaling, high availability, and data protection along with world-class experts to solve your most challenging database problems. That way, developers can focus on developing amazing applications and features without worrying about database maintenance. When developers can focus on your core business, you can win in your competitive market.

MongoDB: The Most Wanted Database

Introduced in 2009, MongoDB has become the leading NoSQL document database. Of the 45 document datastores readily available, MongoDB has topped the NoSQL list for the last 4 years and is currently ranked by developers as the number one most wanted database in a Stack Overflow 2018 survey.[1,2]

MongoDB and similar NoSQL databases allow for faster, more flexible development cycles that help lower software and deployment costs. Additionally, developers appreciate that they are not forced into rigid structures like they are with relational databases. MongoDB meets the need for flexible data models that serve unstructured, semi-structured, and structured data types that are in heavy use for modern applications and microservices.

MongoDB is an open source database that uses a document-oriented data model. Instead of using tables and rows as in relational databases, MongoDB is built on an architecture of collections and documents.

NoSQL Document Model

Document databases store data in documents instead of rows and columns. Documents typically use a structure that is similar to JSON (JavaScript Object Notation), a popular format among developers. Documents provide an intuitive way to model data that is closely aligned with object-oriented programming – each document is an object. Documents contain one or more fields, where each field contains a typed value (such as a string, date, binary, or array). Instead of spreading out across multiple columns and tables, each record and its associated data are stored together in a single document. This simplifies data access and, in many cases, eliminates the need for expensive JOIN operations and complex, multi-record transactions.

In a document database, schemas are dynamic, meaning each document can contain different fields. This flexibility can be helpful for modeling unstructured data. It also makes it easy to iterate during development, performing tasks such as adding new fields. Additionally, document databases can be queried based on any combination of fields in a document.

Applications:
Document databases are general purpose, useful for a wide variety of applications due to the flexibility of the data model, the ability to query on any field and the natural mapping of the document data model to objects in modern programming languages.

Examples of Document Databases: MongoDB, CouchDB, Cassandra

Who Uses MongoDB

Due to its versatility and scalability, a wide range of companies rely on MongoDB for a wide variety of workloads and applications:

Why Implement MongoDB?

MongoDB allows for faster, more flexible development cycles that help lower software and deployment costs. Organizations can move from pilot to production in weeks, not months.

Business Benefits of MongoDB

- Accelerates time to market
- Improves customer experience
- Enables innovation
- Reduces costs

Technical Benefits of MongoDB

- Requires no rigid schema
- Provides **robust indexing capabilities** including secondary indexes and index intersection to cover a wide variety of queries
- Alleviates data type limitations
- Provides **high availability**
- Stores structured and unstructured data
- Supports **more programming languages** (27) with its native BSON format and JSON extensions versus competitors such as Cassandra (13) and Couchbase (14)
- Includes a powerful **Aggregation Pipeline** to allow data transformations along the way
- Allows developers to define and format data according to usage
- Delivers **flexible and scalable architectures** engineered from the ground up as fault tolerant clusters with redundancy

How MongoDB is Used

Well suited for a wide range of use cases, MongoDB especially adapts to newer applications that can take advantage of the rich feature set and document style database MongoDB offers, including:

General Use Cases

- Highly unstructured data
- Large volumes of data
- Extensive data write-intensive activities
- Extreme desire for read/write/query balancing
- High demand for duplication or denormalization

Specific Use Cases

- Mobile and mobile analytics
- Catalog data and inventory
- Content management
- Social media user data
- Game user data
- Internet of Things (IoT)
- Augmented reality (AR), Virtual reality (VR), and Mixed reality (MR)
- Core and off-chain datastores that support blockchain initiatives

Here are some of the most common use cases ObjectRocket sees from customers:

1 Customer Analytics

Providing consistently good customer experiences requires data aggregation. Understanding this data is key to enhancing an organization's presence in the customer journey.

Companies now collect massive amounts of data about existing and potential customers while aggregating it with publicly available data. This information provides companies with new data about how customers interact with products (digitally and in person); personal preferences; demographics; and much more. This gathering of disparate data allows companies to build customer profiles and nurture paths with the goal of a more successful customer toward the purchase path.

Since all of this data originates from multiple sources with different schema, tying it all together at such a massive scale becomes the challenge. MongoDB's flexibility and scalability allows for the aggregation of this data and speed to deliver a dynamic experience that evolves based on customer behavior in real time.

2 Product Catalog

Although not new to the evolving digital experience, today's product catalogs have increased in volume and richness of data. MongoDB provides tools that store multiple types of objects with different sets of attributes. The dynamic schema capability allows for product documents to contain attributes relevant to that product. No longer does every product record need to contain every possible attribute. MongoDB developers make changes to their catalogs quickly and easily.

3 Real Time Data Integration

Organizations have vast amounts of data spread across multiple touch points. Data provides value if it is aggregated in a "single view". Previously, companies spent enormous energy and resources on data ingestion, transformation, and schema changes in order to obtain a single source of data in real time. MongoDB's flexible query capabilities easily aggregate data to create more efficient tools. With the addition of change streams, developers can monitor and take action on specific events quickly.

4 Mobility and Scaling

Companies face the daily challenge of storing and analyzing varying data structures across multiple sources with highly dynamic growth, especially from mobile application development. MongoDB's flexibility and scalability provides an ideal database solution for dealing with this type of environment. Leveraging schemas that can evolve over time, mobile application developers no longer need to spend valuable time adjusting the database. Instead, they focus on developing a better customer experience.

MongoDB Challenges

Many developers start building applications with MongoDB quite easily. When a range of complex challenges occurs as the environment grows, organizations need to maximize performance and ensure consistency.

In this section, we'll cover specific challenges of running MongoDB in production at scale. In the next section, we'll cover how ObjectRocket, a MongoDB database as a service provider, solves these issues for companies.

Management Strains Technical Resources

Without dedicated MongoDB DBAs and engineers, developers may spend excessive amounts of time configuring and troubleshooting the database layer instead of writing code and focusing on developing new features. This can slow time-to-market, decrease application quality, and deprive technical contributors of the time they need to pursue new ideas.

Scaling Involves Complex Decisions and High Risk

Although MongoDB is inherently designed to scale, scaling horizontally presents a range of challenges. For a horizontal cluster, architects must plan how to best split data onto different servers, when to shard those workloads, and how to select the best shard keys for each one.

Poor architecture decisions — or a failure to plan for sharding early on — can have catastrophic results as the system grows. But the necessary preparation can further slow time-to-market, exacerbate the expense of retaining MongoDB expertise, and place even more strain on other technical resources.

Optimizing MongoDB Demands Expensive Experience

Because of its popularity and increasingly high demand, engineers with meaningful experience managing MongoDB production environments at scale can be expensive[3] and the hiring process extremely time-consuming. When it comes to planning for scalability, automating routine management tasks, and optimizing the database layer for specific application needs, organizations need to invest the most experienced partner possible.

Scaling MongoDB

Scaling refers to the ability of a database to expand as needed for additional storage/disk, RAM/memory, CPUs/compute cycles, networking, or other resources. To successfully scale MongoDB, every organization must consider the entire stack.

First, a few definitions:

Scale
The ability of a system to function within a pre-defined acceptable level as workloads increase, where the pre-defined acceptable level varies per application

Workload
The number of active end users

Entire Stack
The application, the network, and the database

Approaches to Scaling

Many organizations create elaborate multi-year marketing or business plans trying to anticipate future trends and customer habits, i.e., proactive planning. On the other end of the spectrum, some organizations respond to the market after the impact of trends has occurred, i.e., reactive planning. There is also a whole lot of strategies that fall in between these approaches. For example, during architectural design, you may only proactively consider which actions you take if X event occurs. The decision to shard up front can have profound ramifications on how you will scale at different set pain points.

Organizations that are most successful with MongoDB create a strategy for their MongoDB solution and iterate based on their workload and on usage spikes.

Scaling proactively

Organizations can schedule, plan, and create a proactive scaling project in anticipation of an impending spike or increased activity, characterized by two general patterns:

Marketing push: Adding a significant number of customers and/or amount of data

Cyclical spikes in usage: Extreme increase in activity or events that require data capture and high volume of data such as holiday purchases, updated product catalog, new product launches, etc

Scaling reactively

As an organization begins experiencing bottlenecks or performance degradation, they must reactively (after the fact) scale.

Trouble signs to watch for indicating a potential need to scale:

- Increased query times for end users
- Increased login times
- Increased load on hosts
- Requests and servers freezing up
- Slower server response times
- Database slow" as noted by developers
- Out-of-memory errors
- Unintended elections
- Errors in the logs

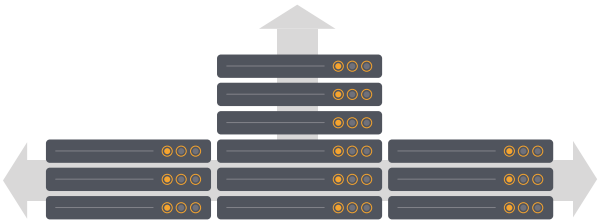
If any of these trouble signs appear, organizations must scale immediately to keep pace with the ensuing demand and understand any impact to customer experience.

Learn when to scale MongoDB instances

Read blog >

Methods to Scaling: Vertical vs. Horizontal

Organizations have two ways to scale: **Up (Vertically)** or **Out (Horizontally).**



Scaling Vertically

Scaling vertically increases computing resources, such as the number and type of CPUs, the amount of RAM, or amount disk space required.

Today, there are better options for commodity hardware, cheaper disks, and better storage options, affordable memory, software, and networking that provide a more gracefully means to manage failovers and interruptions.

The main benefits of vertical scaling include:

- Reduced architectural complexity
- Fewer hosts to maintain
- Less maintenance when resources are unavailable

Scaling up works well for many applications and needs. However, keep in mind that sometimes hidden costs to scaling vertically arise when using larger replica sets. If the environment grows rapidly, an organization may constantly be moving to larger machines; or they must add resources to hosts until that option is no longer viable. Plus, upgrade cycles are less efficient on a single larger host versus a horizontally scaled environment.

Scaling Horizontally

Scaling horizontally distribute computing resources across a network, typically by adding servers. To distribute data across multiple nodes as they are added, MongoDB utilizes a database architecture called sharding, which uses key ranges to partition data that is distributed among multiple database instances.

Sharding is horizontal scaling. It stores data across multiple nodes, distributing the load and the processes across the hosts.

By scaling horizontally, organizations can add additional resources with physical or virtual hosts:

- Physical – availability of low-cost commodity hardware
- Virtual – ability to add additional CPU cores or nodes via VMs or cloud
- Networking – capability to add load balancers, additional mongoS processes, etc.

However, sharding comes with a trade-off in some increased overall complexity. Yet sharding also provides the benefit of simplifying maintenance by allowing for rolling upgrades and the ability to perform operations such as index builds in parallel at the same time across shard/nodes.

Some comparisons between using larger Replica Sets vs. Sharded Clusters:

Replica Set	Sharding
Simple	Expertise needed
Lots of Reads across a wide data set (Don't want to scatter gathers)	Lots of Writes/Updates (Want to go directly to exact shards for results)
Lots of data, lower activity rates	Lots of data, lots of activity
Need more "normal" resources – ex. just disks	Need more of all resources – Disks, RAM, CPU, write scopes

Scaling the Application

Drivers

Always use the latest drivers. The latest drivers provide the option to upgrade to the latest version of MongoDB to take advantage of the latest capabilities. Plus, the latest drivers also provide improved performance through bug fixes.

Driver settings are also important. Defaults work well most of the time but to maximize performance, tweak driver parameters, most commonly the "timeout" and "connection pool" settings.

Throttling

Slowing down applications are sometimes beneficial, but not for all applications. Use a throttling mechanism to cope with traffic bursts. Using the example of the product catalog note above, assume all visits are logged on the website to build the user experience. Throttling those events on the database will not be catastrophic for business but could be beneficial in creating a better customer experience.

> Note: Throttling events requires a queuing mechanism such as **Redis** or **Kafka**.

Caching

The vast majority of modern applications take advantage of caching, a temporary fast storage usually an in-memory database like **Redis**. Caching maintains frequently accessed data as close to the application as possible in order to minimize read/writes on the permanent storage, MongoDB.

Using a product catalog as an example: summertime drives users to frequently search for AC units; caching the AC unit search on a Redis database will minimize the hits on MongoDB. Caching is a good fit for static data or data that does not change frequently.

Scaling the Network

Latency

Network latency is an expression of how much time it takes for a packet of data to get from one designated point to another. The lower the latency, the better.

As a result, it is very important to know the network latency between the application tier and the database tier. In order to improve the latency, servers must be as close as possible the application data if not in the same datacenter.

Bandwidth

The amount of data that can be transmitted in a fixed amount of time defines an organization's bandwidth, usually measured in bits per second (bps). Most applications measure their units with megabits (mps) or gigabits per second (gbs).

Bandwidth may have a negative effect on performance by delaying response times. Know and monitor the network capacity between the database and the application tier very closely; or use ObjectRocket, to monitor network capacity.

Projections

Whenever an application issues a read request (query), MongoDB sends the entire matching document(s). If applications need only a portion and not the full document, project on the query only the fields you want transferred.

Example:

> A user searches a product catalog for "air conditioning unit".

> More than 100 specs (fields) associated with "air conditioning unit" display; it is not possible to display all 100 as it will be hard for the user to view and compare the results.

> Instead, fetch the price, the brand name, the btu, and the stock availability. In this case, the customer is interested in a particular model and can click to get the full details.

> If the entire document is a few Kbs, the application may end up transferring only a few bytes over the wire using this technique.

Shorten Field Names

A read request transfers an entire document over the wire. A document consists of field names and their values. Making the field names shorter saves bandwidth as it decreases each document's size. For example, in the product catalog described above, instead of "product_name", the field can be named "n_p" and the price can be named as "p".

Getting Ahead Of Usage Spikes in Sharded MongoDB Clusters

Usage spikes happen. MongoDB workloads can encounter spikes for a number of reasons:

- Seasonal patterns
- New features
- New customer growth
- Big marketing events

Being prepared to handle the increase in operations, connections, and storage can make the difference for growing successfully.

Scale Out in Advance

For sharded collections with proper shard keys and targeted operations, scaling out can achieve near linear scaling—the recommended way to scale MongoDB. When adding shards, scale both write and read throughput versus adding additional secondary members, which will only scale read performance when all members are available.

For those running MongoDB components in an environment where the same virtual machine or container can be deployed on the same physical hardware, it is important to distribute the components across different physical devices as much as possible. Distribution prevents components from exhausting shared resources such as storage and networking.

Understand limits

Know the limits and the latency threshold of current clusters and workloads to calculate how many additional shards required.

Example:

Company Zed has 10 shards. They expect a 50% increase in traffic during the back-to-school rush.

They currently handle 20,000 requests per second while maintaining less than or equal to 10 milliseconds per request.

Then →

In order to keep up with traffic, they'll need an additional five shards during the rush to handle the expected increase to maintain the same query speed.

Plan accordingly to allow adequate time for balancing chunks between shards. Starting in MongoDB 3.4, multiple chunk migrations can run in parallel that can significantly decrease the balancing time.

Additionally, scaling out is not limited to data nodes. The extra workload also puts more demand on the mongos tier, especially when scaling up the application tier for the added traffic. The mongos layer will need to:

- Handle more client and shard connections
- Process more operations
- Return more results per second

In some cases, the mongos can become a bottleneck and add latency to requests when concurrency increases.

Add additional mongos servers in advance

More mongos servers can reduce concurrency for an individual mongos and reduce the resources consumed by each mongos server. Also, deploy a subset of mongos servers to an application server. Selecting and using a subset prevents the driver from establishing connections to all mongos servers, in turn, keeping connection counts lower per mongos server.

Scale Up in Advance

Sometimes, scaling clusters out may also require an organization to scale up. The increased workload and added components increase overall work and resources consumed by the process. The processes and the storage engine also have internal data structures and threads that are responsible for specific actions, all of which require additional CPU and memory when the workload increases. Adding other components will also increase connection counts. The **mongod** process will now have more connections between each shard in addition to the connections established by the mongos.

> ***Remember:*** Each active connection can use up to one megabyte of memory.

An alternative to scaling out reads is scaling up reads by adding additional secondaries. MongoDB allows for users to configure a read preference for operations. This allows them to utilize secondary members of clusters to distribute reads. Read preferences that make use of secondaries is fine for reads that are okay with eventual consistency, ie, no need to read data that was just written to the primary. It is essential to provision enough secondaries so that if one is down for planned or unplanned maintenance, the remaining secondaries can handle the workload.

Unused Data and Indexing

For read-intensive spikes, it is necessary to **remove any unneeded documents** and **have proper indexing in place**.

Over time, collections can grow quickly. These documents are great candidates for purging. They can be the result of an application bug, data that is no longer needed, or data from testing.

To prevent this data from creating unnecessarily large indexes, getting paged from disk, or uncompressed in memory, remove it from the collections in advance. If a significant number of documents are removed, it is important to initial sync all members to rebuild the data files and remove any fragmentation for both WiredTiger and MMAPv1.

Unused indexes can also add overhead for writes (as index entries are also written) and for reads (if the optimizer had to evaluate them as a potential plan).

Queue and Caching Systems

Lastly, implementing a caching system and queue system can help manage a usage spike. Each method serves a different purpose during a spike in traffic.

- A caching system prevents unnecessary and redundant calls to the database. Caching helps avoid degradation in performance because caching systems often outperform databases for static data.
- A queue system allows for throttling the rate of requests against the database, where a queue approach enables the application to process requests as expected but not overwhelm the database with requests.

Maintaining and Optimizing MongoDB

Some of the most time-consuming tasks for large Mongo environments is selecting the right shard keys, optimizing queries and indexes, and upgrades. Here are some of the many services ObjectRocket helps with to optimize MongoDB and ensure that companies get the most out of this technology.

Shard Key Selection

The shard key is either an indexed field or indexed compound fields that exist in every document in the collection. MongoDB partitions data in the collection using ranges of shard key values. Each range defines a non-overlapping range of shard key values and is associated with a chunk.

A good shard key has:

- Sufficient cardinality
- Distributed writes
- Targeted reads (query isolation)

Shard keys should be in every query, if possible.

Defining Shard Keys

After setting up the physical sharding infrastructure, focus on the logical aspect of sharding. Shard keys represent the various fields within a collection that MongoDB uses to partition data. Conveniently, Mongo allows the user to define these keys.

To identify the fields that involves implementing shard keys:

- Identify the collections to shard. Collections that are larger than 200 MB and whose data can be distributed evenly are good candidates.
- Generate an appropriate shard key.

When creating a shard key, consider the following recommendations and questions:

- Gain insight into the manner in which the application interacts with the database.
- Consider whether the application is more read- or write-heavy, or whether it is balanced evenly.
- Determine the most important activity against the database.
- For example, the application might write large amounts of data to the database, but the most important activity might involve queries returning data in less than 100 ms.
- Define the expected weekly and monthly patterns of data growth.
- Address any pain or problem areas such as slow queries
- Identify when the application tends to get busy: during certain hours of the day, week, month, or year; or if it is constantly busy all the time

After investigating these issues, you can begin performing a more detailed analysis.

Index Issues

The most common issue for slowing down MongoDB stems from index issues.

The key to using a large technical reference manual, such as a cookbook, is a good index at the back of the book. Without an index that indicates on what page the "chicken parmesan" recipe is on, a user may need to look through the entire book before finding the desired 'data'. The same analogy holds true for MongoDB indexes. The larger the 'cookbook' – i.e., the MongoDB instance – the more important it is to have a good index to find things quickly.

MongoDB indexes help queries by narrowing down the search scope when looking up a document. Poor performing queries are usually the result of not having an index at all or not setting up the appropriate indexes for your queries.

Conversely, too many indexes also impact write performance. If every "the" and every "and" is indexed, this slows down MongoDB instances because each time data is inserted into results, the indexes get updated.

Queries also have to be formulated in such a way that they are able to use the correct index. In many cases, queries need to include the predicate field or the first field of a compound/concatenated index for it to be able to use the index.

Upgrading MongoDB

Upgrades for MongoDB can be painful. Often, companies that use managed services for MongoDB are forced to upgrade to keep their level of support. These managed service companies often do not help customers plan for these upgrades to ensure minimal downtime.

There are definite advantages to upgrading. Deprecated versions are not backported for bug fixes, especially since each version involves hundreds of bugs. For example, as many as 150 bug fixes related to the database engine and more than 280 other enhancements and improvements have been identified and resolved for version 3.4 alone.

Here are some reasons it can be painful to upgrade your MongoDB instances:

Downtime

Both minor and major version upgrades can usually be completed without any downtime, if your driver is set up to be resilient during a step down and mongos restart. Timelines for upgrades depend on many factors, such as which version you're currently running, your applications' needs, and which features will offer the most benefit for your app. Most customers can expect a 30-minute maintenance window with a 15 second period of downtime.

Rewriting Code & Deprecated Features

It's not uncommon to have to change your code in order to upgrade. Each new major version may or may not deprecate query operators and tools. If you're using a lot of features, there's a good chance some of your queries will need to change. Features may also be shelved but in most cases there is a replacement feature or a workaround that can be implemented.

Updating Drivers

Since you are likely using an older version of MongoDB, your driver might already be out of date. New drivers offer new features and can improve your application and user experience. You might have to update your driver in advance. Sometimes, this can be painful because a driver upgrade involves dependencies so other components/ frameworks might need an upgrade as well.

For all the reasons listed above, it is always recommend to test a new major version before upgrading.

Upgrade Considerations

Here are some things we look at when figuring out how long an upgrade to an instance will take:

- Code language
- Driver version
- Code compatibility, like deprecated operators
- MongoDB version constraints (dataset compatibility checks)

Managed MongoDB from ObjectRocket

Reap all the benefits of MongoDB without the worrying about the production challenges.

Managing, scaling, optimizing, and upgrading MongoDB takes time, money, and focus away from your core application.

ObjectRocket manages thousands of production instances of MongoDB, including one of the world's largest at over 1000 shards that supports over 1.25 billion active users each month.[4] Since 2012, we have offered support for larger replica sets as well as being one of the first providers to offer support for large sharded MongoDB clusters. The platform hosts petabytes of data, supporting billions of users on behalf of a variety of customers, large and small.

Add the right kind of modern database expertise to your team without adding headcount.

Everything is included regardless of plan size

A big difference between ObjectRocket and other Database as a Service (DBaaS) companies is that everything is included at every plan size. There are no upsells and no surprises. It's just straightforward, predictable pricing. Other companies may force you to upgrade to a more expensive plan if you need access to a DBA or help with query optimizations. With ObjectRocket, every customer gets the support they need, regardless of plan size.

An important part of the ObjectRocket mission is to ensure the delivery of data management ease and understanding to customers so that they can focus on building compelling products instead of focusing on database maintenance.

Whether you have a simple database or a large, complex application with distributed databases, we make it easy to migrate to the ObjectRocket platform by working with you one-on-one.

> ***ObjectRocket is core to Braze's technical success. Their infrastructure and team can support the way our application is built and also allow us to continue to scale MongoDB to such a massive level.***
>
> ***Jon Hyman***
> *CTO and Cofounder, Braze*

> ***What's really helpful to me as CTO is we don't really have to have that non-relational database expertise in-house. We rely on ObjectRocket to help us optimize our queries at what is sometimes an absolutely insane scale.***
>
> ***Alen Durbuzovic***
> *CTO, Chive Media Group*

Data Migration & Test Planning and Support

Every ObjectRocket plan includes complimentary migration service by our Data Migration Team. ObjectRocket migrates workloads across a variety of technologies, providers, and architectures. ObjectRocket does the heavy lifting so that you can focus on your application.

Our Account Executives and Technical Account Managers (TAMs) work together with your team to ensure a smooth transition to supporting your data. Your TAM will set up a regularly scheduled call to ensure you're hitting all the milestones necessary to be successful with testing and the data migration process.

You'll have access to a dedicated Slack channel so that our teams can communicate in real time. This includes conversations with our DBAs and CDEs on everything from troubleshooting to optimization.

Our complimentary migration service includes:

- Help preparing connectivity from your current environment to ours
- Adding replica members on ObjectRocket to your existing instance
- Migrating your existing configuration files to our infrastructure
- One-on-one guidance as you update your application
- Ensuring a short maintenance window and clean data migration
- Assurance that your application has been appropriately migrated
- Help removing old replica members, if no longer needed

> ***ObjectRocket is the gold standard in support. The entire tech staff are just the smartest people in the world. They're just so good. When you see an organization like this that truly cares about the right solution, it's amazing.***
>
> ***BJ Fox***
> *VP of Engineering, Thunder Industries*

Monitoring

Our industry-leading automated monitoring checks 254 individual metrics each minute on every instance to ensure the health of our customers' data. ObjectRocket engineers are instantly alerted if any of these health checks fail and they go right to work to triage any issue that arises.

Optimization

One of the most time-consuming tasks for large Mongo environments is optimizing queries and indexes. It's difficult and expensive to find experienced MongoDB experts to join your team. ObjectRocket provides this service with every plan. Our experts know a ton of tips and tricks to help your development team fully take advantage of all the benefits of Mongo features, plus they know strategies to greatly reduce the size of the cluster, saving you money and improving performance at the same time. Our expertise pays for itself many times over.

Upgrades

In the spirit of sharing expertise and supporting customers' database needs, ObjectRocket helps plan upgrades for MongoDB instances on the customer timeline so they can reap the benefits of new features and avoid the pitfalls that sometimes accompany deprecated versions.

24/7 White Glove Support

Never get stuck in a black hole of support again. Our global support team is responsive, consultative, and communicative.

Our support team is empowered to ensure the right resources are brought in by escalating to senior team members and/or utilizing our on-call process. This ensures that no matter what time it is, no matter where you're located, we'll respond and work with your team to find the cause of any issue, even if the problem isn't on our end. First line responders are highly qualified engineers and DBAs that know all the ins and outs of running MongoDB in production. Our support team has over 85 MongoDB certifications.

Polyglot Persistence

The database technology market is a rapidly changing space that features new use cases and requirements. As a result, the ability to capture, store, and leverage data evolves as engineers develop innovations to find the value of captured data.

Polyglot persistence involves leveraging different database technologies to address different database needs within an application. Because each database possesses its own strengths and weaknesses, the days of utilizing a single database per application are nearly gone. Polyglot persistence helps to ensure that the right tool is used for the right job at the right time.

ObjectRocket supports the following open source NoSQL databases:

- MongoDB
- Elasticsearch with Kibana
- Redis

ObjectRocket has supported workloads of all sizes since 2012 and aims to support more modern databases on any infrastructure. As the market quickly moves to clouds like AWS and Azure, ObjectRocket will be there to support our customers' open source database needs now and in the future.

If you want to connect your database with MongoDB, remember that ObjectRocket can help you through it. We offer fully managed database-as-a-service solutions, and we can free up your developers to focus on building your app by taking the database maintenance piece off your hands.

> ***We use MySQL databases for primary data store and then we supplement with MongoDB, Redis, and Elasticsearch for different pieces of our application. People often ask me why we don't just move all the way to Mongo or move all the way to Redis. The simple answer is that you don't have to put all your eggs in one basket. Use what's going to work best for you.***

Greg Avola
CTO and Co-founder, Untappd

> ***From a pure financial play, it's a no-brainer because it's a significant cost savings and saves us a lot of time. We get everything we need and it's reliable. And it's scalable. And it's taken care of by someone else. Show me where to sign—ObjectRocket makes it so easy.***

Jeff Hansen
Technical Director, Digital Engagement, Single Stone Consulting

Ways to contact ObjectRocket Support

We've got your back 24/7/365.

- Dedicated Slack channel
- Phone
- Email
- Chat

Enjoy the best hands-on support, hands-down. It's the right response, 24/7/365.

There are no hidden charges, and much more is included than you're likely to get anywhere else.

- Hosting
- Monitoring
- Data Migration Planning & Support
- Version upgrades
- 24/7/365 support
- DBA services
- Performance tuning
- Configuration advice
- Support for advanced configurations
- Schema design
- Architecting
- Security
- Query & index optimization
- Financially-backed SLAs

ObjectRocket
a rackspace company

About ObjectRocket

ObjectRocket's technology and expertise helps businesses build better apps, faster so developers can concentrate on creating applications and features without having to worry about managing databases. We'll migrate your data at no cost and with little-to-no downtime. Our DBAs do all the heavy lifting for you so you can focus on your builds. We provide 24x7x365 expert support and architecture services for MongoDB, Elasticsearch, Redis, and Hadoop instances in data centers across the globe.

Contact ObjectRocket for a more information about our Database as a Service (DBaaS) offerings. We're always happy to provide a free, no-hassle consultation of the leading rocket-fueled platform that is purpose-built for running production workloads of MongoDB at scale.

LET'S TALK DATA

[1] According to DB-Engines ranking, https://db-engines.com/en/ranking.

[2] According to a recent Stack Overflow survey, https://insights.stackoverflow.com/survey/2018/

[3] Based on Indeed.com industry salary trends as of 8/16/18: $158,000/year is the median salary for a senior data engineer in San Francisco, CA., capping on the high end at $282,000.

[4] 2018 Braze case study. https://www.objectrocket.com/braze-case-study/

Date Modified: 08/23/2018