

Comparing MongoDB and DynamoDB

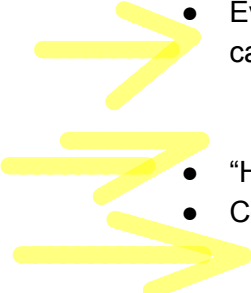
Introduction

DynamoDB is a proprietary key-value database that AWS offers as a fully managed NoSQL service. It is based on the Dynamo datastore design, and was originally launched in January 2012.

Amazon developed Dynamo in the mid-2000s, and published a paper describing its design in 2007. It was developed to address the need for a scalable and highly-available key-value storage system for use in a number of internal Amazon systems, including indexing for S3, shopping carts, best selling lists and session management. What is common about these services is that they require a platform that only needs primary-key access to a data store, and benefit from predictable low latency operations. DynamoDB was developed as a managed service for users with applications requiring similar simple, but scalable, query patterns. DynamoDB use cases are limited, and for more general purpose application requirements, MongoDB is a more flexible choice.

Both MongoDB (through [MongoDB Atlas](#)) and DynamoDB are available as a database as a service in the cloud. However, MongoDB provides much more deployment flexibility, eliminating lock-in to any one platform provider. Unlike DynamoDB, MongoDB can be run within a customer's own data center, or on self-provisioned instances from any cloud provider. Customers can have complete control over their environment, or pass operational responsibility back to MongoDB through the MongoDB Atlas service which runs on AWS, Azure, and Google Cloud Platform. As a customer's application evolves, they can move from a managed service on MongoDB Atlas, through to self-managed database instances on any public cloud, or in their own data centers. DynamoDB can only be run as a managed service on AWS.

In addition to platform lock-in, DynamoDB also presents a number of limitations that push complexity back to engineering teams

- 
- Eventual consistency, restrictive data model, limited query, indexing and aggregation capabilities, coupled with primitive operational tooling.
 - For the development / devops staff, this means more development effort in code, less agility and longer engineering cycles
 - “Hot partitions” dramatically increasing costs
 - Complex pricing and capacity planning, and low market adoption.
 - For the business, this means longer time to market, unpredictable pricing and the risk of deep platform lock-in

The following sections of the analysis compare MongoDB to DynamoDB across key development, operations and business domains.

[Introduction](#)

“This document is Proprietary and Confidential. No part of this document may be disclosed in any manner to a third party without the prior written consent of MongoDB Inc.”

[Technical Considerations](#)

[DynamoDB: Complex Eventual Data Consistency with Restrictive Data Model](#)

[Transactional Data Integrity](#)

[DynamoDB: Limited Query Capabilities](#)

[DynamoDB: Low Operational Maturity & Performance](#)

[Commercial Considerations](#)

[Complex DynamoDB Pricing & Capacity Planning Model](#)

[Platform Lock-In](#)

[Low market adoption](#)

[Conclusion](#)

Technical Considerations

DynamoDB: Complex Eventual Data Consistency with Restrictive Data Model

Eventual Consistency

- By default, DynamoDB is eventually consistent. Any node can accept a write from the application, but reads could be directed to nodes that have not yet applied the write, and therefore return stale or deleted data.
 - As a result, developers must handle the complexity of managing inconsistent data in the application.
- Users can configure read operations to return only consistent data from base tables, but this doubles the cost of each read, and adds latency.
- Unlike read operations made against the base tables, there is no way to configure strong consistency when querying against Global Secondary Indexes (GSIs). Any operations performed against a GSI will be eventually consistent. As stated by [the documentation](#) *"Because of this, your application logic should be capable of handling GSI query results that are potentially out-of-date."*
- DynamoDB has no way of allowing users to configure consistency levels beyond very basic weak and strong consistency

Data Model Restrictions

- DynamoDB is limited to a maximum record size of 400KB, including the key, the value and any [Local Secondary Index \(LSI\)](#) attributes defined against the record.
 - Attempting to store any record that exceeds 400KB will return an error
 - With no support for JOIN operations, users must break larger objects down into chunks that do not exceed the 400KB limit, and then recombine them in application code, therefore adding complexity.

- DynamoDB has **no direct support for dates** and has **only one numeric type**. It therefore **cannot distinguish between regular and long integers, floating point numbers or decimal values**
 - As a result, **developers must preserve types on the client**, which adds **further application complexity** to ensure data can be properly sorted and compared, and reduces data re-use across different applications. This especially restrictive for dates and times, used in many time-series based applications.
- Beyond simple Key-value objects, **AWS has added support for JSON** to provide document-like data structures that better match objects in application code. However, **these are also restricted**:
 - Subject to the same **400KB record size limit**. This severely limits the ability to store the rich, embedded JSON data structures that many apps generate.
 - **Indexes can only be applied to top level JSON keys**. Sub-documents and arrays cannot be indexed, limiting the ability to efficiently query the embedded elements of a document.
 - **Updating any JSON value requires the entire JSON object to be rewritten**
- **No validation of data being stored**. Applications can write data of any type or structure into the database.
 - **Developers would need to add validation logic to the application to enforce governance control**
- DynamoDB offers **no way for its drivers to automatically retry writes** in the event of a **system failure or a network timeout**. As a result, developers must write complex error handling code in their applications.

MongoDB offers much greater simplicity and flexibility with strong consistency and its document data model.

- Compared to DynamoDB, MongoDB is **strongly consistent by default**. This helps **reduce the amount of time developers spend dealing with errors related to consistency** and focus on designing for **business requirements**. Developers can tune consistency levels in MongoDB, with **support for strict linearizable and causal guarantees**.
- For additional flexibility, **consistency requirements for read operations can be relaxed**. Through secondary consistency control, read queries can be routed **only to replicas that fall within acceptable consistency limits with the primary server**.
- Documents can be up to **16MB** in size.
- With **GridFS**, MongoDB can support storing **multiple content assets** (i.e. pdfs, images, movies) **natively within the database – with no limit to file sizes** (subject to underlying storage).
- **Indexes can be applied to any attribute in a document, down to individual values in arrays**.
- Through Binary Encoded JSON (**BSON**), MongoDB supports **a much wider array of data types**, **further reducing application complexity**.
- MongoDB can **selectively update any field in a document**, including **multiple matching elements in an array, in a single atomic update**, without rewriting the entire document in the client.

- **Schema validation** enables MongoDB to enforce governance controls on data written to the database
- The addition of retryable writes to MongoDB moves the complexity of handling temporary system failures from the application to the database. Now, rather than the developer having to implement custom, client-side code, the MongoDB driver can automatically retry writes in the event of transient network failures or a primary replica election, while the MongoDB server enforces exactly-once processing semantics to handle both idempotent and non-idempotent operations

Transactional Data Integrity

Many non-relational “NoSQL” databases provide the **data integrity guarantees** required by most applications. In the case of MongoDB, documents bring together related data that would otherwise be modeled across separate parent-child tables in a relational schema, **Atomic single-document operations** enable one or more fields to be written in a single operation, including updates to multiple sub-documents and elements of an array. The guarantees provided by MongoDB ensure complete isolation as a document is updated; any errors cause the operation to roll back so that clients receive a consistent view of the document.

MongoDB 4.0, scheduled for Summer 2018*, will add support for multi-document transactions. This will make it the only database to combine the ACID guarantees of traditional relational databases, the speed, flexibility, and power of the document model, with the intelligent distributed systems design to scale-out and place data where you need it.

DynamoDB does not support multi-record ACID transactions in the database. As a result, developers need to implement the functionality in application code, which adds complexity, cost, and time to development, testing and maintenance processes.

Multi-document ACID transactions in MongoDB will feel just like the transactions developers are familiar with from relational databases – multi-statement, similar syntax, and easy to add to any application. Through snapshot isolation, transactions provide a globally consistent view of data, enforce all-or-nothing execution, and they will not impact performance for workloads that do not require them. For many developers and DBAs, simply knowing that they are available will provide critical peace of mind. The addition of multi-document transactions makes it even easier for organizations to address a complete range of use-cases with MongoDB. You can learn more about transactions, and [sign up for the beta program today.](#)

DynamoDB: Limited Query Capabilities

Key-Value Queries Only

- **DynamoDB only supports key-value queries**

Other

- For any queries requiring aggregations, graph traversals or search required for real-time analytics or rich user experience, DynamoDB data must be first copied into additional AWS technologies such as EMR, RedShift or Elasticsearch.
 - Transferring data adds latency to analytics and search queries as data must first be moved out of DynamoDB to these services, increasing time to insight and imposing greater business risk by operating on aged data
 - Users face additional costs in having to buy access to these services, in addition to their DynamoDB costs
 - Developers face additional complexity and longer development cycles as they now need to use completely different sets of APIs and SDKs to query these services
- Queries to DynamoDB can only return result sets up to 1MB. Once the limit is reached, the query (for example a scan) terminates, and the application must issue another query to perform subsequent retrievals, each limited to 1MB of values
- Drivers are provided in [only six languages](#)
 - There are no drivers for C++, Perl or Scala

Stale and Rigid Indexes

DynamoDB offers both Global Secondary Indexes (GSI) and Local Secondary Indexes (LSI). These are essentially materialized views and come with [significant restrictions](#)

- A maximum of [only five indexes can be created per table](#), with a maximum of 20 attributes per index
- Indexes can be defined as hash or hash-range indexes only. More advanced indexing strategies to support rich query requirements such as compound, unique, array, geospatial, partial, text or TTL are not available
- Index keys can only be String, Number or Binary data types
- As noted above, GSIs are eventually consistent with the underlying data
- GSIs do not support ad-hoc queries, instead the developer must know access patterns in advance. As GSIs are essentially materialized views, they do not fetch attributes from the parent table, therefore support covered queries only.
- LSIs can be queried to return strongly consistent data, but only from the node where the LSI is defined.
 - LSIs must be defined when the table is created. They cannot be added to existing tables. They cannot be removed without dropping the entire table
- DynamoDB indexes further complicate capacity planning as they are sized and provisioned separately from underlying tables, causing potentially non-deterministic performance issues at run time. As cautioned by the documentation
 - *"Because some or all writes to a DynamoDB table result in writes to related GSIs, it is possible that a GSI's provisioned throughput can be exhausted. In such a scenario, subsequent writes to the table will be throttled. This can occur even if the table has available write capacity units."*

The MongoDB query language and rich array of secondary indexes provide much richer application functionality

- Users should be able to access and manipulate their data in sophisticated ways to support both operational and analytical applications. Indexes play a critical role in

providing efficient access to data, supported natively by the database rather than maintained in application code.

- Expressive queries. The MongoDB query language enables developers to build applications that can query and analyze the data in multiple ways – by single keys, ranges, faceted search, graph traversals, and geospatial queries through to complex aggregations, JOINS, and subqueries.. Complex queries are executed natively in the database without having to use additional analytics frameworks or tools, and avoiding the latency that comes from moving data between operational and analytical engines.
- Rich secondary indexes. Providing fast filtering and access to data by any attribute, MongoDB supports compound, unique, array, partial, TTL, geospatial, sparse, hash and text indexes to optimize for multiple query patterns, data types and application requirements. Indexes are strongly consistent with the underlying data
- BI & analytics integration. The [MongoDB Connector for BI](#) enables industry leading analytical and visualization tools such as Tableau to efficiently access data stored in MongoDB using standard SQL.
- MongoDB supports a dozen drivers for different programming languages, with availability of over 30 community-supported drivers for maximum developer flexibility.

DynamoDB: Low Operational Maturity & Performance

Performance Limits & “Hot” Partitions

[Only a single AWS region](#) (US East) supports DynamoDB performance exceeding 10,000 read or write units.

Additionally, design of the data model is essential to ensure provisioned throughput can be realized. As per [the documentation](#)

- *“Amazon DynamoDB assumes a relatively random access pattern across all primary keys. You should set up your data model so that your requests result in a fairly even distribution of traffic across primary keys. If you have a highly uneven or skewed access pattern, you may not be able to achieve your level of provisioned throughput.”*
- *“it is recommended that you considering modifying your request pattern or your data model in order to achieve a relatively random access pattern across primary keys.”*

It is important keep in mind that DynamoDB's storage limits actually dictates the number of partitions (individual partitions are limited to 10GB with 3,000 read capacity units or 1,000 write capacity units), not total desired throughput. Since users don't have any control over partitioning, if any individual partition is saturated, one would have to **double capacity (and therefore cost) to split partitions** rather than scale linearly. Very careful design of the data model is essential to ensure that provisioned throughput can be realized.

A number of DynamoDB users have been caught by this limitation:

- [You probably shouldn't use DynamoDB:](#)
 - *“This seems simple enough but an issue arises in how dynamo decides to distribute the requested capacity. It distributes it evenly across your nodes. So if you provision 100 write capacity units and you have 4 nodes, each node gets 25 write capacity units. The trouble arises because you cannot control*

how many nodes you have. That is decided by your data volume. For every 10GBs of data you get a new node. So if you have some number of terabytes of data you are going to have hundreds of nodes in your DynamoDB cluster. The requested throughput is then distributed to these nodes. So if you request a write capacity of 1000 and you have 200 nodes what this really means is that each node only has a write capacity of 5. That is pitifully small and will cause throughput exceptions.”

- [The million dollar engineering problem:](#)
 - *“The implication here is that you aren’t paying for total throughput, but rather partition count. And if you happen to have a few keys which saturate the same individual partitions, you have to double capacity to split a single hot partition onto their own partitions rather than scale the capacity linearly. And even there you are limited to the throughput for a single partition.”*

Note that once data has been split into new partitions to handle increased loads, it can not be repartitioned to scale back down

- [Scaling costs with DynamoDB](#)
 - *“If you want to handle spikes in traffic, the immediate course of action is to just increase the number of instances to handle the surge, and bring it back down to save money. You can scale the read and write capacity freely, but once you split partitions, partitions are split forever. This is one of the biggest caveats, which is actually mentioned in their docs. Whether it's a technical limitation or profitable business strategy, it's definitely a cash sink for companies using this service. 😞”*

Slow Auto Scaling

DynamoDB has introduced auto-scaling to automatically provision more throughput in response to growth in the workload. Note that auto-scaling needs to be [configured separately for tables and indexes](#). There are several limitations discussed in [The problems with DynamoDB Auto Scaling and how it might be improved](#) article:

- Scaling actions can lag bursts in capacity by up to 15 minutes, making it too slow to respond to sudden increases in application usage
- Throughput increases are based on consumed capacity, rather than actual usage. Therefore for auto-scaling to be effective, users must predict in advance what their throughput demands will be, limiting the ability to react to sudden demands for throughput growth.

As a result of these limitations, auto-scaling will only serve a subset of use cases where users can predict in advance both what, and how much, additional capacity they will need.

A sticking plaster for performance: Dynamo Application Accelerator (DAX)

DAX provides a caching layer for DynamoDB, delivering a claimed 10x latency improvement without application changes. There are however, some major limitations in DAX, which, according to [AWS documentation](#) is “because it is not tightly coupled to DynamoDB.”

- Does not support workloads that require strong consistency of data, or write intensive workloads. It only serves read-centric apps that can tolerate eventually consistent data

- Any write operations are persisted first to the base DynamoDB table, which are then pushed into the DAX cache
 - Any cache misses are retrieved from the base table using an eventually consistent read. Users cannot configure from a strongly consistent read
- No encryption – does not support TLS. IAM roles created for DynamoDB tables are [not honored by DAX](#). Instead, it uses its own access policies that are configured independently
- Only supports applications written in Java and node.js

Limited Monitoring and Backup

While DynamoDB enables simple provisioning, other key operational tasks are not well served by DynamoDB

- On-demand and continuous backups. It took AWS 5 years to bring a managed backup service to DynamoDB. However, there are some considerations that users need to be aware of
 - Backups are not complete. Auto-scaling policies, IAM permissions, cloud watch metrics and alerts, TTL, and global table settings all need to be recreated on the restored table, adding performance overhead and potential security risk
 - Users pay for both backing up and restoring data, whether they use on-demand or continuous backups. Each is priced separately and at different rates
- Less than 20 DynamoDB metrics are reported by [AWS CloudWatch](#), which limits visibility into real time database behavior. Some metrics are collected every 60 seconds, while others are collected only at 5 minute intervals
- There are no tools that allow developers or DBAs to optimize performance by visualizing the DynamoDB schema or graphically profile query performance

Incomplete Security

Beyond field level access controls, DynamoDB is missing key functionality needed to power applications subject to regulatory compliance:

- Limited encryption of data at rest
 - AWS has been late in adding this [security feature](#), only releasing it in February 2018.
 - Encryption at rest can only be configured for new tables, when they are created. It cannot be applied to existing tables.
 - It is not possible to enable Encryption at Rest for DynamoDB Streams, which are used for change data capture and global replication
 - Users are charged separately for all calls to the AWS Key Management Service, increasing costs and pricing complexity
- AWS CloudTrail can be used to create audit trails, but it only tracks [a small subset of DDL \(administrative\) actions](#) to the database, not all user access to individual tables or records. It also incurs additional costs for collecting and storing events

Complex Multi-Region Replication

DynamoDB supports cross region replication with [multi-master global tables](#), however these add further application complexity and cost, with eventual consistency and no automatic client failover:

- Global tables connected with multi-region replication introduce yet another layer of eventual consistency, and impose risk of data loss through hard-coded conflict resolution
 - There is no way to enforce guarantees that writes are persisted across multiple regions before being acknowledged back to the application. Writes in one region and read from another region can include stale data that doesn't reflect the results of recently-completed writes in the other region.
 - If the same item is simultaneously modified in two regions, then DynamoDB's hard-coded Last Writes Win policy will throw one copy away.
- No data sovereignty controls for regulatory compliance. A multi-master architecture means the entire database is replicated across all regions. As a result all of your customer data – including that data which should be contained to one region for regulatory compliance – is replicated across every region. So, like MongoDB zoned sharding, multi-master helps meet demands for low latency and availability, it doesn't offer the data placement policies demanded by modern data privacy regulations.
- Incomplete replication. Global Secondary Indexes are not replicated, and must instead be manually created on each replica. Without doing this, users will experience radically different query latency, or even failed queries, when they access replica tables in other regions.
- No auto-failover in the event of a region failure.
 - To continue to provide service in the event of a complete region failure, AWS advises developers to add their own custom business logic to determine when to redirect requests to alternative regions.
- Inflexible – no way to respond to evolving business requirements without dropping and then recreating tables.
 - New replica tables can only be added to global tables as long as the global table itself is empty.
- Added cost. With the release of global tables, AWS added yet another new pricing metric – the Replicated Write Capacity Unit (rWCU)
- Global tables are dependent on DynamoDB Streams. These add more cost and complexity to pricing calculations, and also do not support encryption at rest, even if the base tables are encrypted. This creates a significant security gap for sensitive data, and prevents the use of global tables for apps in regulated industries

MongoDB provides rich operational tooling with end-to-end security protection and support for active/active cross region deployment

- MongoDB Atlas provides a managed MongoDB service, with automated provisioning, upgrades, monitoring, backup, and security available from an intuitive UI
- Cloud Manager gives the user the additional flexibility of running MongoDB on the cloud infrastructure of their choice, while providing much of the operational automation and convenience provided by a managed DBaaS platform

- Through the Ops Manager and Cloud Manager operational platforms, users can automatically provision, configure and upgrade MongoDB instances with a few clicks from a simple GUI
- Administrators can monitor and alert from over 100 different instance metrics, providing deep visibility into MongoDB
- MongoDB Atlas, Ops Manager and Cloud Manager provide the only tools that allow continuous, consistent point in time backup and recovery of MongoDB clusters, ensuring users have robust disaster recovery capabilities that reduce the risk of data loss. In addition, MongoDB backups are queryable, without users having to first restore the data. This is especially powerful when just needing to restore a small piece of the database, and for reviewing how data and document schema has changed over time. This functionality is available to DynamoDB.
- The MongoDB In-Memory storage engine eliminates the requirement to install a caching layer such as DAX in front of MongoDB when predictable low latency is a primary application requirement. It supports both read and write-centric workloads, with strong consistency data guarantees.
- MongoDB Atlas can auto-scale underlying storage capacity, based on actual consumption
- MongoDB Enterprise Advanced provides end to end security controls for applications managing sensitive data, including extensive access controls, authentication and encryption of data in-flight and at-rest on disks and in backups
- MongoDB's truly distributed design enables single clusters to span racks, data centers and continents. With replica sets supporting up to 50 members and zone sharding across regions (self-managed MongoDB), administrators can provision clusters that support active/active data center deployments, with write local/read global access patterns, strong data consistency, and 99.999% availability. The MongoDB Atlas service supports up to 7 replicas that can be distributed across regions, allowing for resilience to region failures and for data locality. Remote replicas can be provisioned at any time from the Atlas GUI and API, and are managed as part of a single cluster.
- With zoned sharding (note, support for MongoDB Atlas coming soon), developers can deploy "write anywhere" active-active clusters, allowing data to be localized to any region. With each region mastering its own data, the risks of data loss and eventual consistency imposed by multi-master DynamoDB are eliminated, and customers can meet the data sovereignty demands of new privacy regulations
 - To learn more, [read our blog](#) comparing multi-master and zoned sharding for active-active deployments

Commercial Considerations

Complex DynamoDB Pricing & Capacity Planning Model

While DynamoDB is a managed service, it does not abstract away the need for careful capacity planning to ensure sufficient resources are available when the application requires them.

- Users need to consider that DynamoDB supports up to [10,000 operations per second](#). Any requirements to expand capacity must be made by a web form, and only granted once an AWS representative responds to the request.
- Users can increase capacity multiple times per day, but resources can take up to several hours to become available. As a result, DynamoDB may not provision capacity sufficiently quickly to handle sudden spikes in load
- Only four downsizing events are supported per day
 - MongoDB Atlas can be scaled on-demand, both vertically and horizontally, providing true elasticity to support highly dynamic workloads

To calculate pricing, users need to assess multiple factors for each DynamoDB table they provision, including

- size of the data set per month
- size of each object
- number of reads per second (pricing is based on reading a 4KB object)
- whether those reads need to be strongly consistent, or eventually consistent (the latter operations are 2x less expensive)
- if accessing a JSON object, the entire document must be retrieved, even if the application needs to read only a single element
- number of writes per second (writes are based on a 1KB object)
- The number of Replicated Write Capacity Unit (rWCU) required if cross-region replication is needed
- size and throughput requirements for each index created against the table
- Costs for backup and restore. AWS offers on-demand and continuous backups – both are charged separately, at different rates for both the backup and restore operation
- data transferred by Dynamo streams per month
- data transfers both in and out of the database per month
- calls to the AWS KMS if encryption at rest is enabled
- cross-regional data transfers, EC2 instances and SQS queues needed for cross-regional deployments
- the use of additional AWS services to plug the gaps in DynamoDB's limited key value query model
- the region you are running the database in
- use of on-demand or reserved instances
- number of metrics pushed into CloudWatch for monitoring
- number of events pushed into CloudTrail for database auditing

- any additional AWS support requirements

Note the “hot partitions” issues discussed above. The limitation of DynamoDB’s data partitioning scheme can cause rapid cost escalation.

Platform Lock-In

- While a local DynamoDB build is available for prototyping on a developer laptop, DynamoDB can only be run in production on AWS
- As enterprises are starting to unshackle themselves from decades of lock-in to proprietary software, many are fearful of repeating mistakes from the past with new platform lock-in from cloud vendors
 - Comparethemarket.com, the UK’s leading price comparison service, recently completed a transition from on-prem deployments with Microsoft SQL Server to AWS and MongoDB. When asked why they hadn’t selected DynamoDB, a company representative [was quoted as saying](#) “DynamoDB was eschewed to help avoid AWS vendor lock-in.”
- DynamoDB is based on a proprietary code base. Developers are not able to access the underlying source code

MongoDB can be run anywhere – from developer laptop to on-prem data center to public cloud platforms. MongoDB Atlas provides a database as a service from the makers of MongoDB, eliminating the operational heavy lifting required with any application. Pricing is much simpler than DynamoDB:

- users select the size of instance and storage they need
- the number of replicas and shards that will make up the cluster
- whether to include backups
- the region(s) the cluster needs to run in.

Users then receive a price for consumption, against which they add data transfer costs.

MongoDB Atlas is available on-demand through a pay-as-you-go model and billed on an hourly basis. It’s easy to get started – use a simple GUI to select the instance size, region, and features you need. MongoDB Atlas provides:

- Security features to protect your data, with fine-grained access control and end-to-end encryption
- Built in replication for always-on availability. Cross-region replication within a public cloud can be enabled to help tolerate the failure of an entire cloud region.
- Fully managed, continuous and consistent backups with point in time recovery to protect against data corruption, and the ability to query backups in-place without full restores
- Fine-grained monitoring and customizable alerts for comprehensive performance visibility
- One-click scale up, out, or down on demand. MongoDB Atlas can provision additional storage capacity as needed without manual intervention.

- Automated patching and single-click upgrades for new major versions of the database, enabling you to take advantage of the latest and greatest MongoDB features
- Live migration to move your self-managed MongoDB clusters into the Atlas service with minimal downtime

Low market adoption

- Based on [DB-Engines ratings](#) of database popularity and adoption, DynamoDB has only reached 21st position in rankings, compared to MongoDB in 5th position, with a score showing over 10x higher usage
- The 451 Group's [NoSQL Skills Index](#) has DynamoDB at 7th position, a long way behind MongoDB in number 1 position, with higher skills availability than all NoSQL database vendors combined

MongoDB is by far the leading non-relational database, enjoying adoption by a wide number of organizations across a wide number of use cases, with higher skills availability and proven best practices than DynamoDB. Prior to AWS reInvent 17, Andy Jasay, CEO of AWS, [acknowledged](#) MongoDB's popularity over DynamoDB:

- *"When we built DynamoDB, because our customers really wanted us to have a native, nonrelational database offering, but MongoDB continues to kick butt on top of AWS."*

In its latest [Big Data NoSQL Wave](#), Forrester rates MongoDB ahead of DynamoDB for current offering

FIGURE 3 Forrester Wave™: Big Data NoSQL, Q3 2016

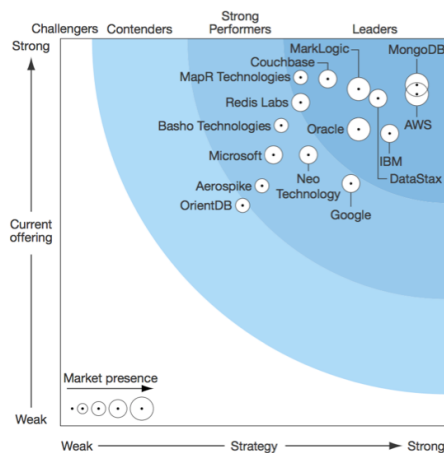


FIGURE 3 Forrester Wave™: Big Data NoSQL, Q3 2016 (Cont.)

	Forrester's Weighting	Aerospike	Amazon Web Services	Basho Technologies	Couchbase	DataStax	Google	IBM	MapR Technologies	MarkLogic	Microsoft	MongoDB	Neo Technology	Oracle	OrientDB	Redis Labs
CURRENT OFFERING	50%	3.10	4.28	3.88	4.48	4.23	3.13	3.78	4.50	4.35	3.50	4.40	3.50	3.83	2.85	4.18
Development	50%	2.30	4.10	3.15	4.15	3.65	3.30	3.60	4.50	4.50	3.40	3.90	3.45	3.15	2.85	3.80
Deployment	50%	3.90	4.45	4.60	4.80	4.80	2.95	3.95	4.50	4.20	3.60	4.90	3.55	4.50	2.85	4.55
STRATEGY	50%	2.60	4.60	2.85	3.45	4.10	3.75	4.25	3.10	3.85	2.75	4.60	3.20	3.85	2.35	3.10
Ability to execute	25%	3.00	5.00	3.00	4.00	5.00	4.00	5.00	4.00	4.00	4.00	5.00	4.00	5.00	2.00	3.00
Road map	25%	3.00	5.00	4.00	5.00	5.00	5.00	4.00	4.00	5.00	3.00	5.00	4.00	4.00	3.00	3.00
Open source and licensing	25%	3.00	5.00	3.00	3.00	5.00	3.00	3.00	3.00	3.00	1.00	5.00	3.00	3.00	3.00	5.00
Professional services	20%	1.00	3.00	1.00	1.00	1.00	3.00	5.00	1.00	3.00	3.00	3.00	1.00	3.00	1.00	1.00
Support	5%	3.00	5.00	3.00	5.00	3.00	3.00	5.00	3.00	5.00	3.00	5.00	5.00	5.00	3.00	3.00
MARKET PRESENCE	0%	2.10	5.00	2.65	3.50	3.25	4.00	3.10	2.80	4.40	3.80	5.00	3.35	4.40	2.25	3.55
Product revenue	30%	1.00	5.00	2.00	3.00	3.00	4.00	3.00	3.00	5.00	3.00	5.00	5.00	2.00	4.00	1.00
Install base	25%	2.00	5.00	3.00	4.00	3.00	5.00	3.00	2.00	5.00	5.00	5.00	3.00	5.00	3.00	5.00
Market awareness	30%	3.00	5.00	3.00	4.00	4.00	3.00	3.00	3.00	4.00	3.00	5.00	5.00	4.00	3.00	4.00
Partnerships	10%	2.00	5.00	2.00	2.00	2.00	2.00	3.00	2.00	5.00	5.00	5.00	3.00	5.00	1.00	3.00
Reach	5%	4.00	5.00	4.00	4.00	4.00	3.00	5.00	4.00	5.00	5.00	5.00	4.00	5.00	4.00	4.00

All scores are based on a scale of 0 (weak) to 5 (strong).

Further demonstrating MongoDB's leadership over DynamoDB, Forrester rated MongoDB ahead in the latest [Document Stores wave](#), beating DynamoDB across key categories including development, deployment, roadmap, professional services and support.

FIGURE 3 Forrester Wave™: Document Stores, Q3 '16

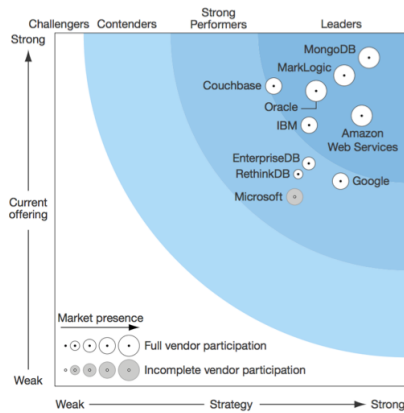


FIGURE 3 Forrester Wave™: Document Stores, Q3 '16 (Cont.)

	Forrester's Weighting	Amazon Web Services	Couchbase	EnterpriseDB	Google	IBM	MarkLogic	Microsoft	MongoDB	Oracle	RethinkDB
CURRENT OFFERING											
Development	50%	3.83	4.25	3.15	2.90	3.55	4.70	2.65	4.55	4.05	3.20
Deployment	50%	4.00	4.10	3.00	2.90	3.85	4.10	2.70	4.75	4.30	2.80
STRATEGY											
Ability to execute	35%	5.00	3.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00	4.00
Road map	35%	4.00	3.00	4.00	5.00	3.00	5.00	3.00	5.00	4.00	3.00
Open source and licensing	15%	4.00	4.00	3.00	3.00	3.00	3.00	3.00	5.00	3.00	5.00
Professional services	10%	4.00	2.00	2.00	3.00	5.00	3.00	3.00	3.00	3.00	1.00
Support	5%	4.00	4.00	3.00	3.00	4.00	4.00	4.00	5.00	3.00	3.00
MARKET PRESENCE											
Product revenue	30%	5.00	3.00	3.00	4.00	3.00	5.00	3.00	5.00	4.00	1.00
Install base	30%	5.00	4.00	3.00	5.00	3.00	5.00	5.00	5.00	5.00	2.00
Market awareness	30%	5.00	4.00	3.00	3.00	4.00	3.00	5.00	5.00	4.00	2.00
Partnerships	5%	5.00	2.00	3.00	2.00	3.00	2.00	5.00	5.00	5.00	1.00
Reach	5%	5.00	4.00	3.00	4.00	5.00	5.00	5.00	5.00	5.00	2.00

All scores are based on a scale of 0 (weak) to 5 (strong).

Conclusion

MongoDB offers a number of technical and business advantages over DynamoDB. The consistency model, indexing, and query language of MongoDB make developers more productive on a wide range of software projects. The addition of multi-document ACID transactions will make it easier than ever for developers to address a complete range of use-cases with MongoDB. Combined with much greater deployment flexibility – from on-prem, to cloud, to managed service coupled with the wide availability of skills and market traction means MongoDB is a safe choice for your next software project.

* Safe Harbour Statement

The development, release, and timing of any features or functionality described for our products remains at our sole discretion. This information is merely intended to outline our general product direction and it should not be relied on in making a purchasing decision nor is this a commitment, promise or legal obligation to deliver any material, code, or functionality.