Azure Cosmos DB vs MongoDB

Table of Contents

Introduction

- Azure Cosmos DB is comparatively faster than MongoDB in most common use cases, although MongoDB performs better over 1MB payload.
- MongoDB offers more flexibility in terms of setup. It runs on-prem, any cloud provider (Google Cloud Platform, AWS, Azure and more).
- We found Azure Cosmos DB to be 93% cheaper for reads and around 20% cheaper for writes than MongoDB Atlas depending on workload characteristics (see TCO table below).
- At Cazton, we work with different kinds of database technologies and have more than a decade of experience in creating highly scalable and efficient systems that use multiple database technologies that are complementary to each other.
- We have helped many Fortune 500 clients; mid-size companies and startups create a scalable data strategy with high performance.
- Want a detailed report for Cosmos DB, MongoDB, Cassandra, Redis, Postgres, SQL Server, MySQL, ElasticSearch and Solr? Contact us at info@cazton.com.

Just a few years ago, it was acceptable to have offline data processing that enabled employees in analyzing big data. That analysis brought continuous improvement to an existing business and the customer experience; however, there has been a paradigm shift. Customers expect real time predictions, and they expect to act on them instantly. So, we need to use incoming information and map it with our existing data models to make real-time predictions.

Now, more than ever, big data analysis is a necessity for a company to excel and grow. The amount of data shared and transferred between humans is unimaginable and to manage, analyze, make real-time predictions and decisions using that data is a daunting task. Many organizations prefer polyglot persistence over a single data storage technology as they have to deal with different types of data.

Over the last decade the popularity of NoSQL databases has skyrocketed. This is mostly because of the increased number of devices including mobile as well as IOT. NoSQL databases have been a strong contender for data storage and processing requirements of modern applications. These databases can be deployed as distributed systems, which makes them more reliable and faster while querying for data. They are inherently more scalable than SQL databases as they allow you to scale vertically and horizontally whereas most SQL databases are vertically scalable by design.

In the technology industry, there can never be a “one size fits all” solution and hence one must be ready to adapt to different technologies to solve different business and technical problems. At Cazton, we work with different kinds of database technologies and have more than a decade of experience in creating highly scalable and efficient systems that use multiple database technologies that are complementary to each other. We have helped many Fortune 500 clients; mid-size companies and startups create a fool-proof data strategy.

Over the last couple of years, we have seen tremendous growth and demand for both Azure Cosmos DB and MongoDB. Both these technologies are promising and offer features that often complement each other. However, the question remains, which among the two performs better? To analyze and understand which one is better, we ran benchmarking code which explicitly performed CRUD (Create, Read, Update, Delete) operations against both the databases.

What is Azure Cosmos DB?

Azure Cosmos DB is a globally distributed, multi-model NoSQL database service that provides strong consistency, extremely low latency, high availability, throughput and up to five consistency levels backed by solid service level agreements (SLAs). The core idea behind building such a database technology was to make it horizontally scalable and globally distributed. This means that the data gets much closer to the customers as it is replicated and stored across multiple Azure data centers in various regions. Azure Cosmos DB guarantees less than 10ms for reads and less than 10ms latency for writes [MB1] less than 1Kb of data. Note that all reads, and writes are served from a local region, and then replicated according to the selected consistency model. Learn more.

What is MongoDB?

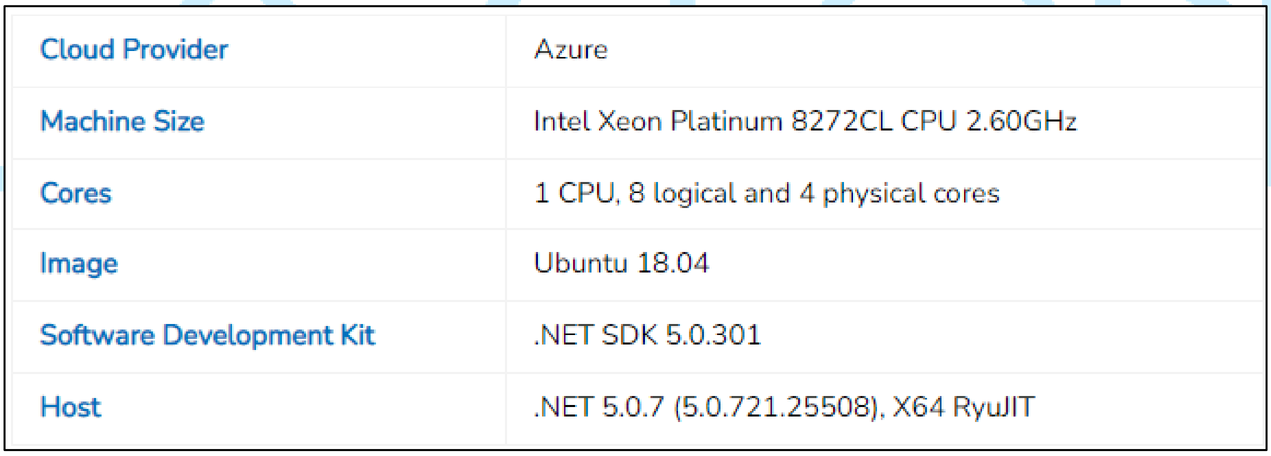
MongoDB is a document-based distributed NoSQL database that can be used for modern, distributed and cloud-based applications. It stores data in BSON format - binary encoding of JSON-like documents and supports arrays and nested JSON objects. It offers SQL-like features like ACID transactions, ad hoc queries, joins, indexing, and much more.

MongoDB offers an option to run it as a service. This means you don't need to worry about setting up physical hardware, installing software or configuring for performance. MongoDB Inc., the company which makes the software for MongoDB, has a service called Atlas. You can leverage MongoDB Database as a Service on any cloud platform like Google Cloud Platform, Azure, or AWS. However, if you wish to keep it local, you can also install it as a service on Windows and Linux platforms. Learn more.

Benchmarking Azure Cosmos DB (SQL API) and MongoDB service on Atlas

The benchmarking effort aims to examine both Azure Cosmos DB (SQL API) and MongoDB service on Atlas and compare their performance using intuitive and easily repeatable scenarios in a cloud-based infrastructure. Using Azure and explicit CRUD operations we were able to create a benchmark that allowed each technology to shine and put its best foot forward. TCO estimates were created using public performance information and a straightforward, data-driven approach to calculations.

- **Machine Specifications:** We ran the benchmarking code on an Azure VM in the same region as the databases. Both Azure Cosmos DB and MongoDB service on Atlas were running in the West coast (US).



Cloud Provider	Azure
Machine Size	Intel Xeon Platinum 8272CL CPU 2.60GHz
Cores	1 CPU, 8 logical and 4 physical cores
Image	Ubuntu 18.04
Software Development Kit	.NET SDK 5.0.301
Host	.NET 5.0.7 (5.0.721.25508), X64 RyuJIT

- **Strategy:** The benchmarking approach included warmup steps to ensure the database instances operated in a representative manner and repeated operations to ensure a statistically significant set of results, including percentile grouping analysis. For the benchmarking effort, we tested CRUD (Create, Read, Update, Delete) operations on Azure with the database technologies and benchmarking code running in the same region. We tested common and uncommon scenarios to assess performance in Microsoft's Azure environment.

We utilized BenchmarkDotNet and NBomber to simulate various load scenarios on the databases. The CRUD operations were performed with the following payloads: 1KB, 10KB, 100KB, 1MB, 2MB. **Azure Cosmos DB has a payload limit of 2MB**. We also made sure that both databases would simultaneously get hits with different user counts ranging from 1 to 1000 users.

Benchmarking Results

- **For 1 KB Payload:** Azure Cosmos DB performed better for create, update, and delete operations whereas MongoDB did well for read operation.

- **For 10 KB Payload:** Again, Azure Cosmos DB performed better for create, update, and delete operations whereas MongoDB did well for read operation.

- **For 100 KB Payload:** Here Azure Cosmos DB performed better for create and read operations whereas MongoDB did well for update and delete operations.

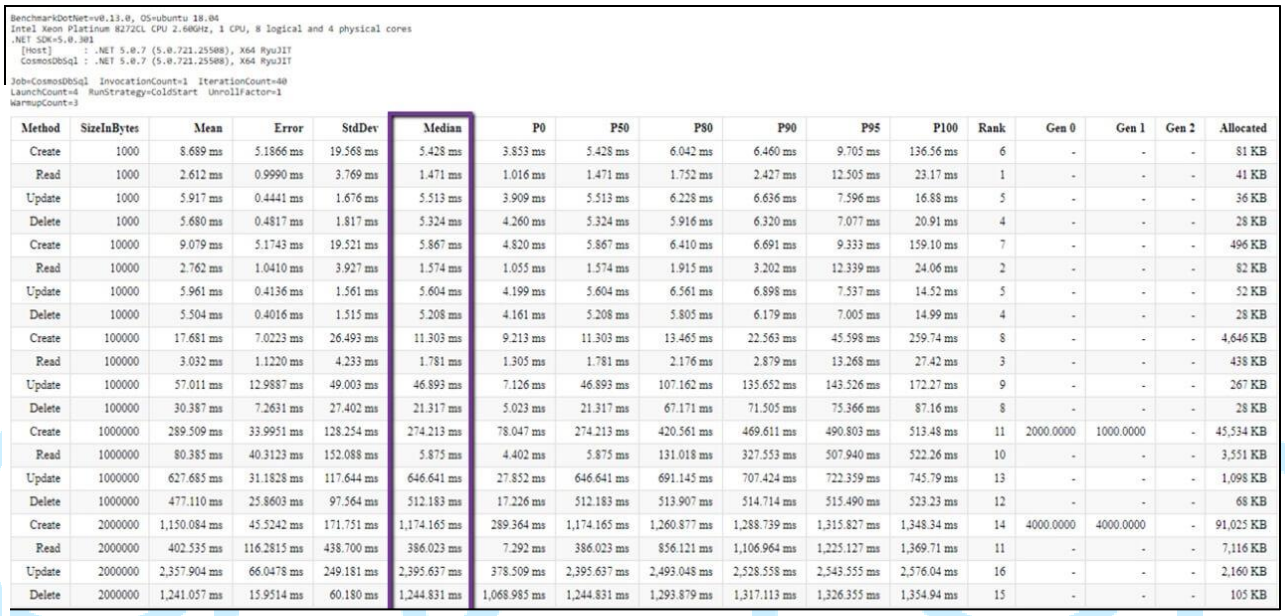
- **For 1 MB Payload:** As the payload size increases, Azure Cosmos DB continues to perform better for read operation whereas MongoDB exceeded in create, update, and delete operations.

- **For 2 MB Payload:** MongoDB completely outperforms Azure Cosmos DB in this scenario.

Findings

As you can see for the most common use-cases with a payload size weighing 1KB, 10KB and 100KB, Azure Cosmos DB performed better than MongoDB. But as the payload size increased, MongoDB showed promising performance.

Azure Cosmos DB Benchmarking Result

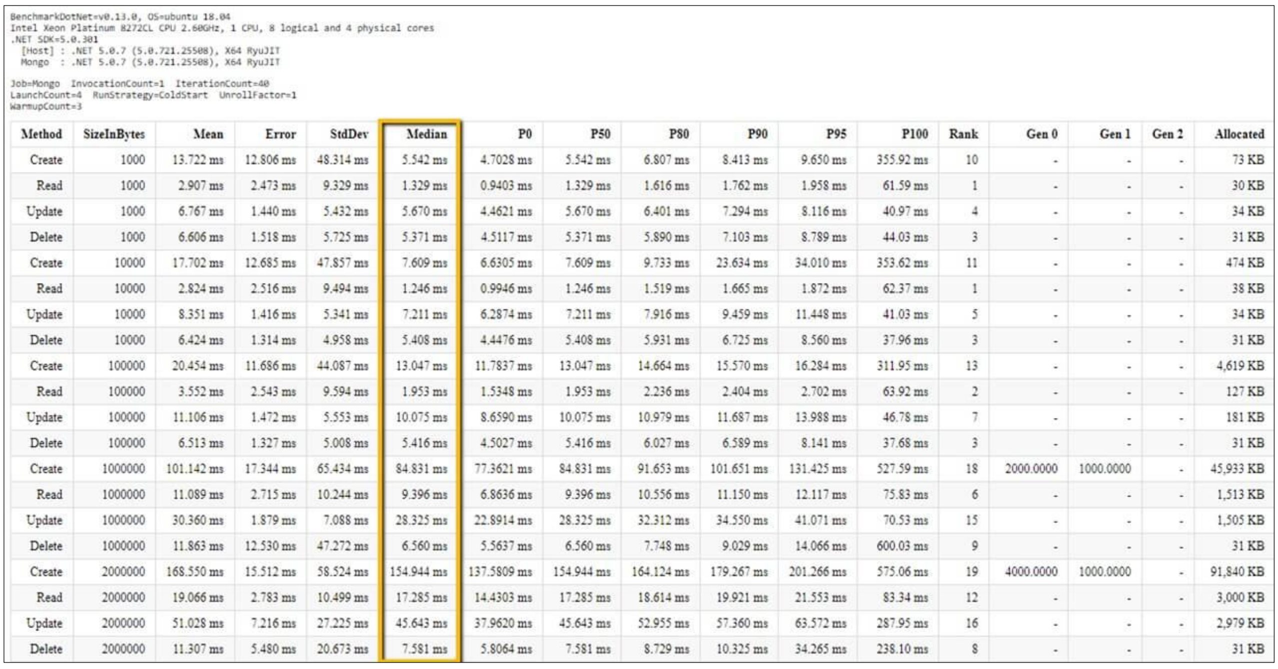


```
BenchmarkDotNet=v0.13.0, OS=ubuntu 18.04
Intel Xeon Platinum 8272CL CPU 2.60GHz, 1 CPU, 8 logical and 4 physical cores
.NET SDK=5.0.301
  [Host]      : .NET 5.0.7 (5.0.721.25508), X64 RyuJIT
  CosmosDbSql : .NET 5.0.7 (5.0.721.25508), X64 RyuJIT

Job=CosmosDbSql  InvocationCount=1  IterationCount=40
LaunchCount=4  RunStrategy=ColdStart  UnrollFactor=1
WarmupCount=3
```

Method	SizeInBytes	Mean	Error	StdDev	Median	P0	P50	P80	P90	P95	P100	Rank	Gen 0	Gen 1	Gen 2	Allocated
Create	1000	8.689 ms	5.1866 ms	19.568 ms	5.428 ms	3.853 ms	5.428 ms	6.042 ms	6.460 ms	9.705 ms	136.56 ms	6	-	-	-	81 KB
Read	1000	2.612 ms	0.9990 ms	3.769 ms	1.471 ms	1.016 ms	1.471 ms	1.752 ms	2.427 ms	12.505 ms	23.17 ms	1	-	-	-	41 KB
Update	1000	5.917 ms	0.4441 ms	1.676 ms	5.513 ms	3.909 ms	5.513 ms	6.228 ms	6.636 ms	7.596 ms	16.88 ms	5	-	-	-	36 KB
Delete	1000	5.680 ms	0.4817 ms	1.817 ms	5.324 ms	4.260 ms	5.324 ms	5.916 ms	6.320 ms	7.077 ms	20.91 ms	4	-	-	-	28 KB
Create	10000	9.079 ms	5.1743 ms	19.521 ms	5.867 ms	4.820 ms	5.867 ms	6.410 ms	6.691 ms	9.333 ms	159.10 ms	7	-	-	-	496 KB
Read	10000	2.762 ms	1.0410 ms	3.927 ms	1.574 ms	1.055 ms	1.574 ms	1.915 ms	3.202 ms	12.339 ms	24.06 ms	2	-	-	-	82 KB
Update	10000	5.961 ms	0.4136 ms	1.561 ms	5.604 ms	4.199 ms	5.604 ms	6.561 ms	6.898 ms	7.537 ms	14.52 ms	5	-	-	-	52 KB
Delete	10000	5.504 ms	0.4016 ms	1.515 ms	5.208 ms	4.161 ms	5.208 ms	5.805 ms	6.179 ms	7.005 ms	14.99 ms	4	-	-	-	28 KB
Create	100000	17.681 ms	7.0223 ms	26.493 ms	11.303 ms	9.213 ms	11.303 ms	13.465 ms	22.563 ms	45.598 ms	259.74 ms	8	-	-	-	4,646 KB
Read	100000	3.032 ms	1.1220 ms	4.233 ms	1.781 ms	1.305 ms	1.781 ms	2.176 ms	2.879 ms	13.268 ms	27.42 ms	3	-	-	-	438 KB
Update	100000	57.011 ms	12.9887 ms	49.003 ms	46.893 ms	7.126 ms	46.893 ms	107.162 ms	135.652 ms	143.526 ms	172.27 ms	9	-	-	-	267 KB
Delete	100000	30.387 ms	7.2631 ms	27.402 ms	21.317 ms	5.023 ms	21.317 ms	67.171 ms	71.505 ms	75.366 ms	87.16 ms	8	-	-	-	28 KB
Create	1000000	289.509 ms	33.9951 ms	128.254 ms	274.213 ms	78.047 ms	274.213 ms	420.561 ms	469.611 ms	490.803 ms	513.48 ms	11	2000.0000	1000.0000	-	45,534 KB
Read	1000000	80.385 ms	40.3123 ms	152.088 ms	5.875 ms	4.402 ms	5.875 ms	131.018 ms	327.553 ms	507.940 ms	522.26 ms	10	-	-	-	3,551 KB
Update	1000000	627.685 ms	31.1828 ms	117.644 ms	646.641 ms	27.852 ms	646.641 ms	691.145 ms	707.424 ms	722.359 ms	745.79 ms	13	-	-	-	1,098 KB
Delete	1000000	477.110 ms	25.8603 ms	97.564 ms	512.183 ms	17.226 ms	512.183 ms	513.907 ms	514.714 ms	515.490 ms	523.23 ms	12	-	-	-	68 KB
Create	2000000	1,150.084 ms	45.5242 ms	171.751 ms	1,174.165 ms	289.364 ms	1,174.165 ms	1,260.877 ms	1,288.739 ms	1,315.827 ms	1,348.34 ms	14	4000.0000	4000.0000	-	91,025 KB
Read	2000000	402.535 ms	116.2815 ms	438.700 ms	386.023 ms	7.292 ms	386.023 ms	856.121 ms	1,106.964 ms	1,225.127 ms	1,369.71 ms	11	-	-	-	7,116 KB
Update	2000000	2,357.904 ms	66.0478 ms	249.181 ms	2,395.637 ms	378.509 ms	2,395.637 ms	2,493.048 ms	2,528.558 ms	2,543.555 ms	2,576.04 ms	16	-	-	-	2,160 KB
Delete	2000000	1,241.057 ms	15.9514 ms	60.180 ms	1,244.831 ms	1,068.985 ms	1,244.831 ms	1,293.879 ms	1,317.113 ms	1,326.355 ms	1,354.94 ms	15	-	-	-	105 KB

MongoDB Benchmarking Result



```
BenchmarkDotNet=v0.13.0, OS=ubuntu 18.04
Intel Xeon Platinum 8272CL CPU 2.60GHz, 1 CPU, 8 logical and 4 physical cores
.NET SDK=5.0.301
  [Host] : .NET 5.0.7 (5.0.721.25508), X64 RyuJIT
  Mongo  : .NET 5.0.7 (5.0.721.25508), X64 RyuJIT

Job=Mongo  InvocationCount=1  IterationCount=40
LaunchCount=4  RunStrategy=ColdStart  UnrollFactor=1
WarmupCount=3
```

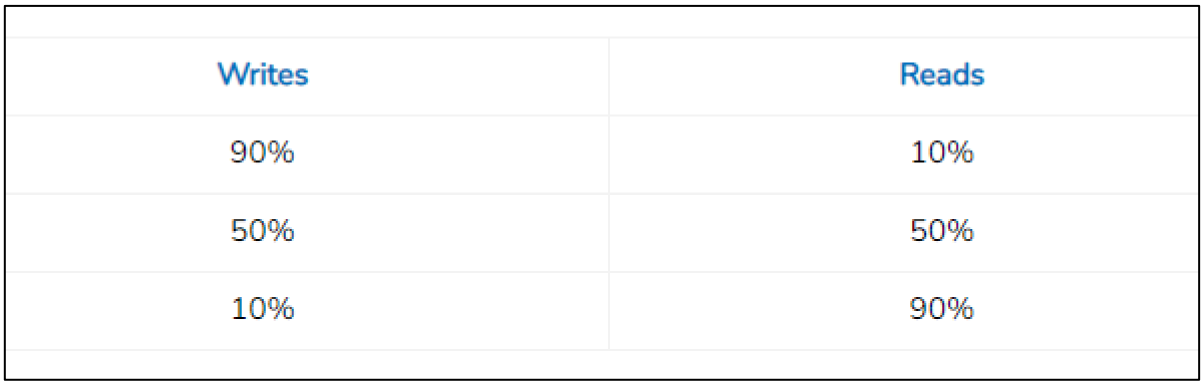
Method	SizeInBytes	Mean	Error	StdDev	Median	P0	P50	P80	P90	P95	P100	Rank	Gen 0	Gen 1	Gen 2	Allocated
Create	1000	13.722 ms	12.806 ms	48.314 ms	5.542 ms	4.7028 ms	5.542 ms	6.807 ms	8.413 ms	9.650 ms	355.92 ms	10	-	-	-	73 KB
Read	1000	2.907 ms	2.473 ms	9.329 ms	1.329 ms	0.9403 ms	1.329 ms	1.616 ms	1.762 ms	1.958 ms	61.59 ms	1	-	-	-	30 KB
Update	1000	6.767 ms	1.440 ms	5.432 ms	5.670 ms	4.4621 ms	5.670 ms	6.401 ms	7.294 ms	8.116 ms	40.97 ms	4	-	-	-	34 KB
Delete	1000	6.606 ms	1.518 ms	5.725 ms	5.371 ms	4.5117 ms	5.371 ms	5.890 ms	7.103 ms	8.789 ms	44.03 ms	3	-	-	-	31 KB
Create	10000	17.702 ms	12.685 ms	47.857 ms	7.609 ms	6.6305 ms	7.609 ms	9.733 ms	23.634 ms	34.010 ms	353.62 ms	11	-	-	-	474 KB
Read	10000	2.824 ms	2.516 ms	9.494 ms	1.246 ms	0.9946 ms	1.246 ms	1.519 ms	1.665 ms	1.872 ms	62.37 ms	1	-	-	-	38 KB
Update	10000	8.351 ms	1.416 ms	5.341 ms	7.211 ms	6.2874 ms	7.211 ms	7.916 ms	9.459 ms	11.448 ms	41.03 ms	5	-	-	-	34 KB
Delete	10000	6.424 ms	1.314 ms	4.958 ms	5.408 ms	4.4476 ms	5.408 ms	5.931 ms	6.725 ms	8.560 ms	37.96 ms	3	-	-	-	31 KB
Create	100000	20.454 ms	11.686 ms	44.087 ms	13.047 ms	11.7837 ms	13.047 ms	14.664 ms	15.570 ms	16.284 ms	311.95 ms	13	-	-	-	4,619 KB
Read	100000	3.552 ms	2.543 ms	9.594 ms	1.953 ms	1.5348 ms	1.953 ms	2.236 ms	2.404 ms	2.702 ms	63.92 ms	2	-	-	-	127 KB
Update	100000	11.106 ms	1.472 ms	5.553 ms	10.075 ms	8.6590 ms	10.075 ms	10.979 ms	11.687 ms	13.988 ms	46.78 ms	7	-	-	-	181 KB
Delete	100000	6.513 ms	1.327 ms	5.008 ms	5.416 ms	4.5027 ms	5.416 ms	6.027 ms	6.589 ms	8.141 ms	37.68 ms	3	-	-	-	31 KB
Create	1000000	101.142 ms	17.344 ms	65.434 ms	84.831 ms	77.3621 ms	84.831 ms	91.653 ms	101.651 ms	131.425 ms	527.59 ms	18	2000.0000	1000.0000	-	45,933 KB
Read	1000000	11.089 ms	2.715 ms	10.244 ms	9.396 ms	6.8636 ms	9.396 ms	10.556 ms	11.150 ms	12.117 ms	75.83 ms	6	-	-	-	1,513 KB
Update	1000000	30.360 ms	1.879 ms	7.088 ms	28.325 ms	22.8914 ms	28.325 ms	32.312 ms	34.550 ms	41.071 ms	70.53 ms	15	-	-	-	1,505 KB
Delete	1000000	11.863 ms	12.530 ms	47.272 ms	6.560 ms	5.5637 ms	6.560 ms	7.748 ms	9.029 ms	14.066 ms	600.03 ms	9	-	-	-	31 KB
Create	2000000	168.550 ms	15.512 ms	58.524 ms	154.944 ms	137.5809 ms	154.944 ms	164.124 ms	179.267 ms	201.266 ms	575.06 ms	19	4000.0000	1000.0000	-	91,840 KB
Read	2000000	19.066 ms	2.783 ms	10.499 ms	17.285 ms	14.4303 ms	17.285 ms	18.614 ms	19.921 ms	21.553 ms	83.34 ms	12	-	-	-	3,000 KB
Update	2000000	51.028 ms	7.216 ms	27.225 ms	45.643 ms	37.9620 ms	45.643 ms	52.955 ms	57.360 ms	63.572 ms	287.95 ms	16	-	-	-	2,979 KB
Delete	2000000	11.307 ms	5.480 ms	20.673 ms	7.581 ms	5.8064 ms	7.581 ms	8.729 ms	10.325 ms	34.265 ms	238.10 ms	8	-	-	-	31 KB

Total Cost of Ownership (TCO)

Beyond throughput, there is the practical question of Total Cost of Ownership. We reviewed an analysis that maps Azure Cosmos RU/s to native MongoDB cores for various workloads with the primary goal of estimating Total Cost of Ownership (TCO). Results suggest Azure Cosmos DB API for MongoDB is best utilized for high-storage, read-heavy, variable-traffic workloads.

- **Approach:** To determine the RU/s for equivalent performance across Azure Cosmos DB for MongoDB API and given MongoDB Atlas SKUs, utilize Yahoo! Cloud Service Benchmark (YCSB), publicly available pricing data, and instances running on Azure Cloud. For MongoDB Atlas, investigate the impact of consistency configuration options on performance and TCO. The performance analysis targeted Azure Cosmos DB for SQL API, compared to this analysis which focuses on the MongoDB API and thereby accounting for a software solution that may migrate from one to the other based on the results.

- **Workloads:** Considering the different distributions of workloads, throughput is estimated based on following breakdowns of traffic:



Writes	Reads
90%	10%
50%	50%
10%	90%

- **Throughput:** The estimate is based on workload on an Atlas M40 instance.

- **Storage:** For sharding, consider large M50 shards and small M30 shards (Azure

Cosmos DB, however, is always sharded with 50GB shards and has no extra cost for sharding).

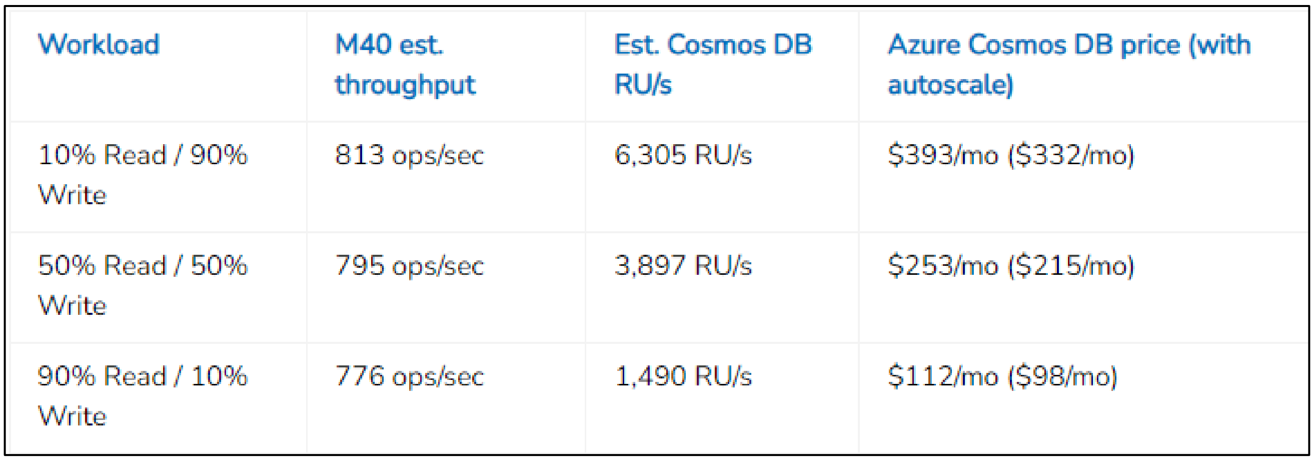
- **Low storage:** 100GB - throughput dominates TCO
- **Balanced storage:** 4TB - max size of unsharded MongoDB Atlas
- **High storage:** 25TB - sharded; storage dominates TCO

- **Elasticity:** For "high-throughput/low-storage" and "balanced scenarios" where throughput impacts TCO substantially, utilize traffic distribution:
  - 66% of the time, throughput is 33% of peak.
  - 33% of the time, throughput is 100% of peak.

Total Cost of Ownership (TCO) Results

Low Storage:

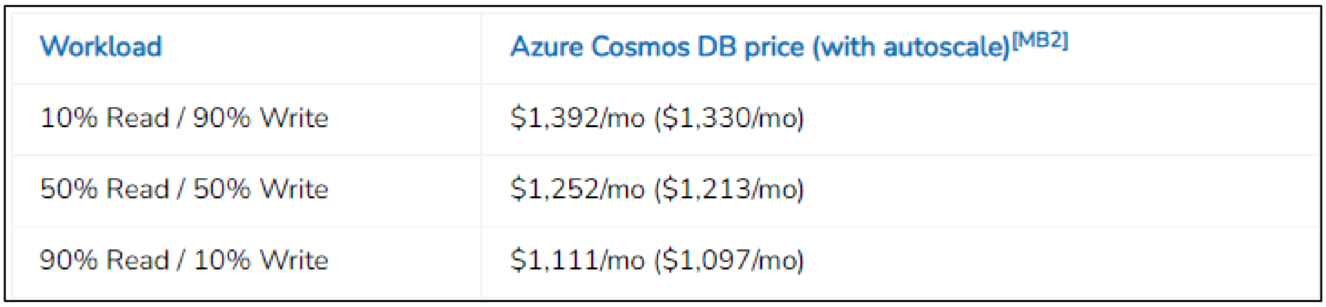
With low storage (~100GB), utilizing an unsharded *Mongo M40* instance. *MongoDB Atlas Azure M40 256GB* estimated price was **$1,095/mo**.



Workload	M40 est. throughput	Est. Cosmos DB RU/s	Azure Cosmos DB price (with autoscale)
10% Read / 90% Write	813 ops/sec	6,305 RU/s	$393/mo ($332/mo)
50% Read / 50% Write	795 ops/sec	3,897 RU/s	$253/mo ($215/mo)
90% Read / 10% Write	776 ops/sec	1,490 RU/s	$112/mo ($98/mo)

Balanced throughput / storage:

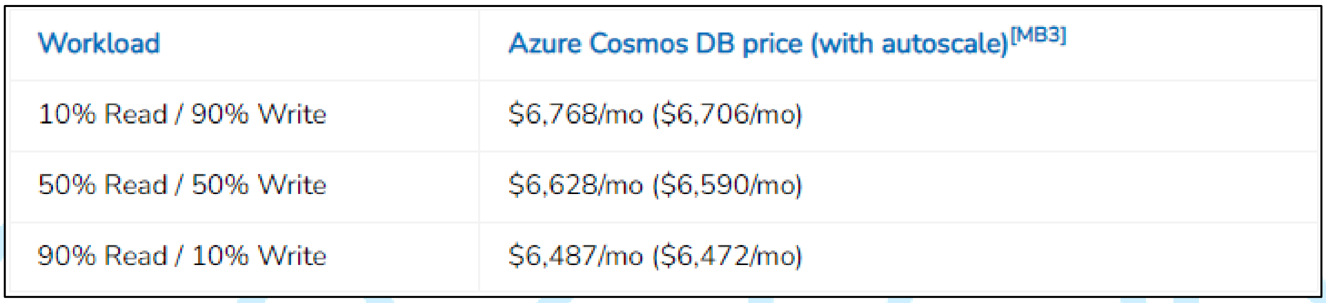
Using the same throughput estimations as above in terms of ops/sec and RU/s, but with a larger customer data size of 4TB, which requires an increased cluster tier to **M50**. *MongoDB Atlas Azure M50 4TB* estimated price: **$3,869/mo**.



Workload	Azure Cosmos DB price (with autoscale)[MB2]
10% Read / 90% Write	$1,392/mo ($1,330/mo)
50% Read / 50% Write	$1,252/mo ($1,213/mo)
90% Read / 10% Write	$1,111/mo ($1,097/mo)

High Storage:

Using the same throughput estimations as above but scaling with an even larger customer data size of **25TB**, which necessitates **7 shards of an M50 tier**. *MongoDB Atlas 7x sharded M50 w/ 8 vcpu, 4TB* data disk per shard price: **$27,536/mo**.



Workload	Azure Cosmos DB price (with autoscale)[MB3]
10% Read / 90% Write	$6,768/mo ($6,706/mo)
50% Read / 50% Write	$6,628/mo ($6,590/mo)
90% Read / 10% Write	$6,487/mo ($6,472/mo)

Findings:

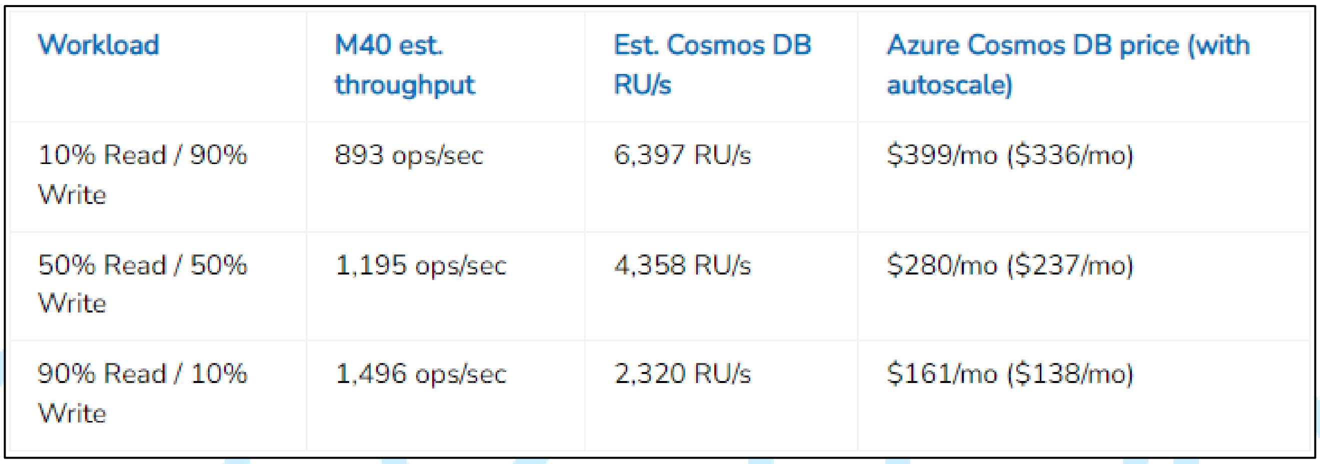
What these results show is a marked difference in price per month for an equivalent Azure Cosmos DB solution vs MongoDB Atlas for a variety of storage and workload scenarios. Low storage and high writes favor MongoDB while higher storage and higher reads favor Azure Cosmos DB in terms of how the prices scale, but the overall finding is that in this estimation Azure Cosmos DB is significantly cheaper for Total Cost of Ownership.

Azure Cosmos DB has five options for consistency. Eventual was used for the testing above. MongoDB separates the concerns into write concern and read preference, which adjust performance and throughput, and therefore also result in price changes in our scenarios due to changes in operations per second and estimated RU/s.

Read Preference:

For example, switching read preference to *SecondaryPreferred [MB4]* on a 3x replicated MongoDB Atlas setup allows clients to read from either secondary replica, potentially halving read cost and therefore doubling read throughput for the same cluster tier.

With that setup in our low storage scenario: *MongoDB Atlas Azure M40 256GB* estimated price: **$1,095/mo**.



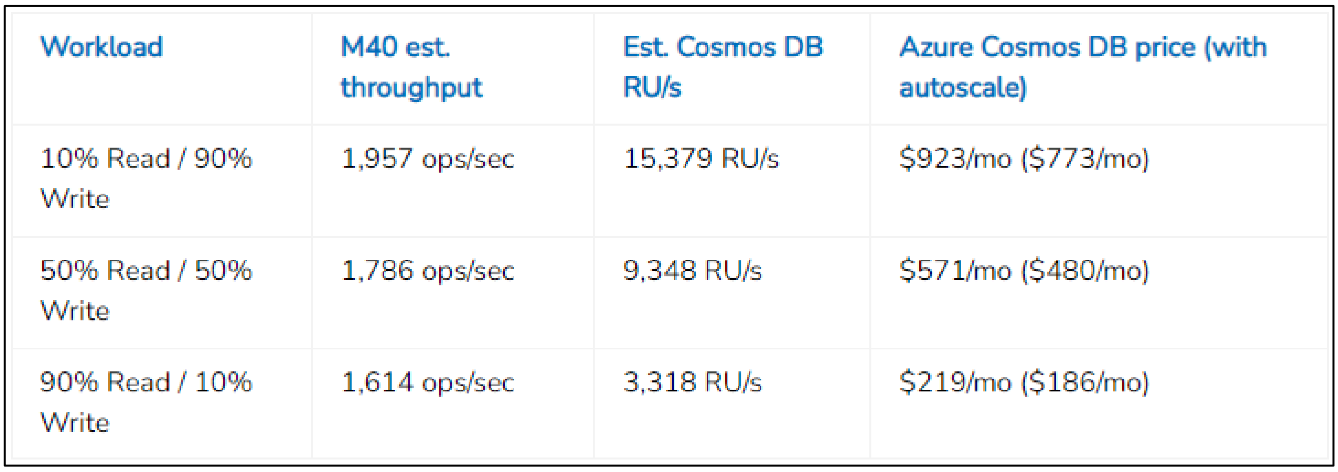
Workload	M40 est. throughput	Est. Cosmos DB RU/s	Azure Cosmos DB price (with autoscale)
10% Read / 90% Write	893 ops/sec	6,397 RU/s	$399/mo ($336/mo)
50% Read / 50% Write	1,195 ops/sec	4,358 RU/s	$280/mo ($237/mo)
90% Read / 10% Write	1,496 ops/sec	2,320 RU/s	$161/mo ($138/mo)

The increased operations/second (and RU/s), leads to an increased Azure Cosmos DB price but the pricing in comparison still favors Azure Cosmos DB. This is true for the balanced and high storage scenarios as well.

Write Preference:

We can go one step further and adjust write concern. Write concern governs the proportion of nodes in a cluster that must acknowledge a write to that cluster. This can directly lead to a lowering of insert cost in a similar 3x replicated cluster as above.

With write concern 1 and read preference SecondaryPreferred in our low storage scenario: *MongoDB Atlas Azure M40 256GB* estimated price: **$1,095/mo**.



Workload	M40 est. throughput	Est. Cosmos DB RU/s	Azure Cosmos DB price (with autoscale)
10% Read / 90% Write	1,957 ops/sec	15,379 RU/s	$923/mo ($773/mo)
50% Read / 50% Write	1,786 ops/sec	9,348 RU/s	$571/mo ($480/mo)
90% Read / 10% Write	1,614 ops/sec	3,318 RU/s	$219/mo ($186/mo)

Note the further increase in operations/second (and RU/s) and resulting change in the Azure Cosmos DB price. Again, note the comparison continues to favor Cosmos DB, which is also true for the balanced and high storage scenarios.

Conclusion

Overall, Azure Cosmos DB had admirable performance and succeeded at maintaining high throughput across the various scenarios. TCO remained modest as scale increased, compared to fixed costing of other solutions. These findings indicate that Cosmos is the more performant and cost-effective solution, except for scenarios over 1 MB (which is not a typical use case). Obviously different needs are filled by different technologies, but Azure Cosmos DB has shown it is an effective solution for a wide range of needs and will satisfy high throughput workloads.

Disclaimer: *These results are based on the benchmarking done on* ***July 15, 2021****, using intuitive and easily repeatable scenarios in a cloud-based infrastructure. Results and pricing can vary over time.*

Our team has gained expertise all over the world and in all fields of the tech industry. They can quickly identify, predict, and satisfy our clients' current and future needs and that has enabled us to build our own framework, which is robust, flexible, scalable, increases speed of development and implements best practices. At Cazton, we believe that it is easy to do the right thing in business because there's not much competition in doing so. We help all types of companies ranging from SMBs to fortune 500s in their digital transformation. And our clients trust us to provide them with the knowledge and skill to tackle every challenge and succeed at every opportunity.

If your company is building applications using Big Data and Artificial Intelligence or creating web or mobile applications on cloud, microservices or on-premises, please check our services to find out why Fortune 500 companies choose Cazton for building multi-billion-dollar revenue generating applications.
Cazton is composed of technical professionals with expertise gained all over the world and in all fields of the tech industry and we put this expertise to work for you. We serve all

industries, including banking, finance, legal services, life sciences & healthcare, technology, media, and the public sector. Check out some of our services:

- Artificial Intelligence
- Big Data
- Web Development
- Mobile Development
- Desktop Development
- API Development
- Database Development
- Cloud
- DevOps
- Enterprise Search
- Blockchain
- Enterprise Architecture

Cazton has expanded into a global company, servicing clients not only across the **United States**, but in **Oslo, Norway; Stockholm, Sweden; London, England; Berlin, Germany; Frankfurt, Germany; Paris, France; Amsterdam, Netherlands; Brussels, Belgium; Rome, Italy; Sydney, Melbourne, Australia; Quebec City, Toronto Vancouver, Montreal, Ottawa, Calgary, Edmonton, Victoria,** and **Winnipeg** as well. In the United States, we provide our consulting and training services across various cities like **Austin, Dallas, Houston, New York, New Jersey, Irvine, Los Angeles, Denver, Boulder, Charlotte, Atlanta, Orlando, Miami, San Antonio, San Diego, San Francisco, San Jose, Stamford**, and others. Contact us today to learn more about what our experts can do for you.