Azure Cosmos DB optimization to reduce cloud cost and increase scalability

1. Abstract:

The Canada e-commerce IMS team was incurring high DB cloud costs of around $1 million USD in Azure Cosmos annually until Q4 FY’21 which we reduced to 150K by Q2 FY'23. Apart from the high cloud costs, we were also facing scalability issues, which made the system struggle to support hot-selling items such as PS5/Xbox, resulting in a poor customer experience and a significant business impact. Therefore, the issue was not just the high cloud cost but also the scalability to meet the growth. High cloud costs are a problem that every other team in Walmart is facing, but if the right steps are taken, they can be avoided. After the cloud cost optimization activity, we not only reduced the DB cost by 6X but also increased the overall scalability of IMS by more than 10X. To achieve these results, we took a series of steps, such as **having the right strategy to purge data, migrating the cosmos containers to a more optimized containers**, **and using an in-house auto-scale service instead of the Microsoft-provided auto-scale service.** After each step, there was a significant reduction in cloud costs and improvements in scalability. Additionally, other teams can leverage our learnings and tools to achieve similar results.

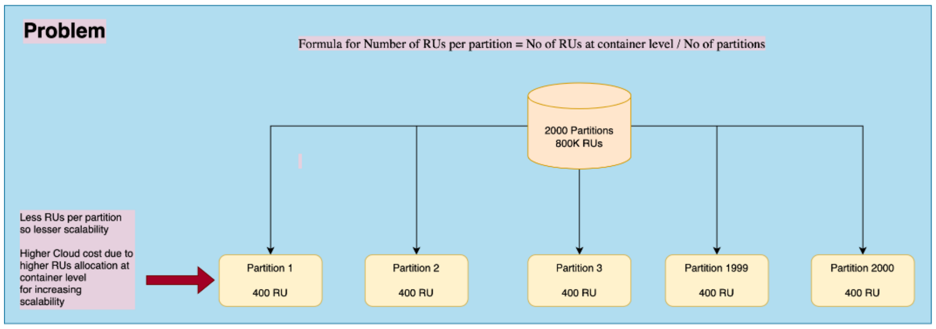
In this Tech paper we mainly want to highlight the work which we did to migrate the **Cosmos container from one container to another container** by building an in-house solution(copy pipeline) as Microsoft doesn't provide a feature to automatically shrunk the partitions or transfer the data from one container to another container. We will also talk about data purging and leveraging in-house autoscaling service which also contributed in cloud cost reduction.

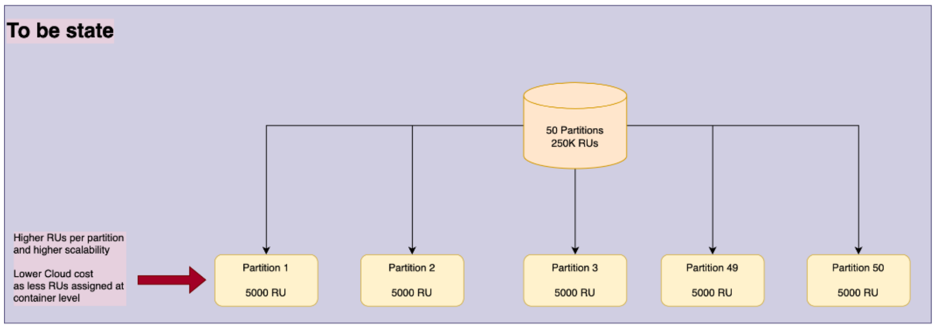
2. Introduction

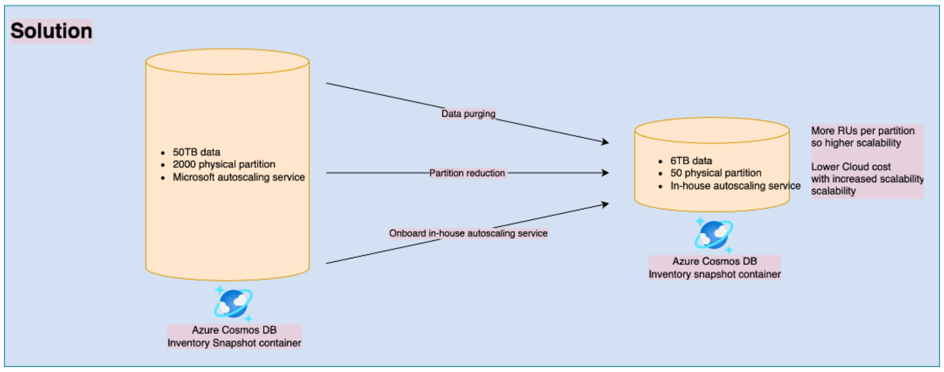
The Canada e-commerce IMS uses Azure Cosmos DB as a database which is a fully managed NoSQL non relational database for modern app development. Azure Cosmos DB offers single-digit millisecond response times, automatic and instant scalability, along with guaranteed speed at any scale. Azure Cosmos DB abstracts the IO, Memory, and CPU required to perform an operation(CRUD) into an abstract unit called Request Units (RUs). The more RUs an operation consumes, the more expensive it is to run on the database engine.

Now switching to Canada IMS, consider a scenario where the actual useful data is only 3TB, but inefficient data purging policies have led to the data size growing to 50TB. Storing such a large amount of data in Cosmos DB would require 2000 physical partitions. However, having a large number of partitions means that each partition will be assigned fewer Request Units (RUs), which can lead to scalability issues during peak traffic periods such as sales events or holiday seasons. As a result, the database may start throttling due to insufficient RUs at the partition level. To prevent this throttling, we would need to scale the RUs at the container level, which can result in higher cloud costs for providing the necessary scalability.

As a part of this optimization our main goal was to fix the data purging policy, migrate containers to newer containers having less partitions and leverage in-house autoscaling service instead of Microsoft auto-scaling service as that is more costly and less efficient.


Problem
Formula for Number of RUs per partition = No of RUs at container level / No of partitions
2000 Partitions
800K RUs
Less RUs per partition so lesser scalability
Higher Cloud cost due to higher RUs allocation at container level for increasing scalability
Partition 1
400 RU
Partition 2
400 RU
Partition 3
400 RU
Partition 1999
400 RU
Partition 2000
400 RU




Solution
50TB data
2000 physical partition
Microsoft autoscaling service
Azure Cosmos DB
Inventory Snapshot container
Data purging
Partition reduction
Onboard in-house autoscaling service
6TB data
50 physical partition
In-house autoscaling service
Azure Cosmos DB
Inventory snapshot container
More RUs per partition so higher scalability
Lower Cloud cost with increased scalability scalability

3. Background

This section outlines some key problems in the Canada eCommerce IMS setup

- Containers were not optimized with TBs of redundant data and lot of physical partitions to store this redundant data.
- Due to large number of partitions even though we were assigning enough RUs at container level, but each partition was effectively getting very less RUs
- This in turn impacted the ability for each partition to serve high number of requests particularly during sale events.

Below table summarizes the whole problem how we were incurring high cloud cost with low scalability.

	Datasize	Number of RUs at container level	Number of Partitions	Number of RUs per partition	Max request per sec on Hot partition	Cloud cost (1 day)	Scalability
Problem	50TB	800K	2000	400	133	1536 $	x
Solution	6TB	250K	50	5000	1666	480 $	12x

As above table summarizes we were facing problem of low scalability even after paying high cloud cost so we took actions to optimize our usage of Cosmos DB which resulted in higher scalability at lower cloud cost.

4. Implementation

To address above problems, we took three steps:

Purged old and redundant data to reduce the overall data size. A new physical partition gets created when either of these two values is crossed which is more than 50GB data or greater than 10,000 RUs per partition. In case of Canada IMS, it was the huge data which led to many physical partitions. On analysing further, it was found once data is created it was not getting deleted as TTL (Time To Live) was not enabled at container level although the TTL value was set as 90 days.

Migrated to a more optimised Cosmos DB container that required fewer partitions, which helps to reduce the minimum RU threshold for a given container and achieve maximum throughput and scalability at a lower cost.

While Microsoft does not provide a built-in capability to migrate containers between databases, our Canada IMS team developed an in-house solution to handle the migration. Given that our service is a Tier 0 service, we took extra precautions to prevent any potential data loss or downtime during the migration process.

Improved the auto scaling capabilities for better efficiency and customisability, without incurring any additional cost. For this we used an in-house auto-scale service that overcame limitations of the Microsoft-provided auto-scaling service.

Reducing the data size was crucial to reduce the number of physical partitions and lower the minimum RU threshold for the container. This, in turn, helped to achieve better throughput and scalability at a lower cost. Our in-house auto-scale service also proved to be more efficient and customizable, which further reduced our cloud costs.

The rest of this document covers above actions in more detail.

3.1. *Purge old and redundant data to reduce the overall data size*

Observation - Purging was disabled for the Canada DB at container level because of which assigned TTL value at document level was not taking into effect.

Action - We enabled purging at container level and set appropriate TTL value based upon discussion with business team. This reduced data drastically from 50TB to 6TB i.e. 8X storage reduction. The old unused data was purged and fresh data growth was capped, as data older than 'X' days was removed.

Result - Data size reduced from 50TB to 6TB.

3.2. *Migrate to a more optimised Cosmos DB container with fewer partitions*

Observation:

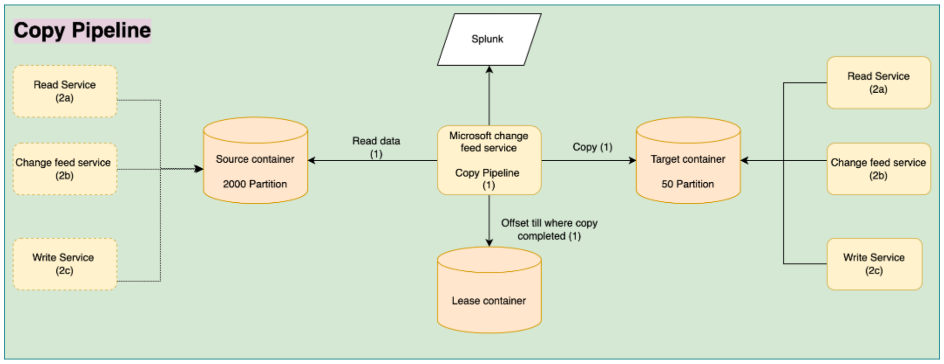
Initially total data size was was 50TB with 2000 physical partitions in some of the containers, after data purging it reduced to 6TB but physical partition count remained same. So, although the data footprint was reduced, the physical partitions allocated to original container remained same.

⚠️ Physical partitions could not be shrunk as Microsoft doesn't support physical partition auto-merging automatically.

Action:

To reduce the number of physical partitions at the container level, rather then merge partitions in-place, we decided to migrate the data to a new container with fewer partitions. We used the Cosmos ChangeFeed service to accomplish this migration.

This is highlighted in figure x .



- Source container: Source container from where data will be migrated
- Target container: New container to which data will be migrated
- Lease container : It is used for keeping a track till where migration is completed so that again and again same data is not copied

The migration steps are as follows:

1. **Set up a data copy pipeline** from the source to the target container.
2. **Cutover dependent services**, once all the data is copied from the source to the target container,
    1. Switch all the read services to the target container.
    2. Switch the change feed services to the target container.
    3. Switch the write services to the target container and note the timestamp.
3. **Ensure the source container is always in sync with the target container**. Hence, in case there is a need to rollback to the source container, there will not be any downtime.
    1. This is achieved by stopping the data copy pipeline and start a reverse copy pipeline from the target to the source container (using a new auxiliary container). However, instead of starting the copy pipeline from the beginning, use the timestamp noted in step 2c.

Read cutover (2a above): In the case of Canada IMS, there were two production regions with a 50-50% traffic split in both regions. For performing the read traffic switch, one of the regions was pointed to the new container and the traffic was slowly released into that region by controlling it from the GSLB. Only when there were no errors observed, 100% traffic was released into the new container, and then the other region was also pointed to the new container.

Write cutover (2c above): In the case of writes, there was only a single write region, so the secondary region was pointed to the new target container, and the primary was brought down manually. GSLB health checks failed for approximately 30 seconds, and then all the traffic was served from the new container. To perform the write switch, a time-window was selected during non-peak hours when almost no customer traffic was received so that even in those 30 seconds, no write call fails.

Results:

As a result of these steps, we were able to achieve:

- A reduction of 20 times the number of physical partitions
- An increase in scalability by 10 times
- Cloud cost savings of 150K USD
- Much smoother sales events, resulting in a great customer experience.

3.3. In-house Cosmos Auto-scale Service:

Observations:

There are two ways to provision throughput for a container: Manual and Autoscale mode. The RUs provisioned for a container is distributed equally across all the physical partitions for that container. In manual mode, the maximum RUs remains unchanged, whereas in autoscale mode, it varies depending on the usage, from the minimum RUs threshold to the maximum RUs required in case of throttling.

There are critical cointainers in IMS which directly serves Tier-0 inventory lookup services. Autoscaling was enabled for these critical containers so that whenever there is throttling, RUs are increased. To use this feature, Canada IMS was paying 1.5X the regular Cosmos cost. However, it was observed that the Microsoft Autoscale service was not very efficient in scaling the RUs under high throttling scenarios, as we can't define by how much percentage RUs has to be increased. Therefore, due to the above reasons, we leveraged and enhanced an in-house Cosmos autoscale service to monitors Microsoft Cosmos containers and scale provisioned RUs up or down depending on the load.

Actions:

The service retrieves metrics from the Azure Metrics API and scales a container's provisioned RUs based on two criteria:

Scaling up condition:

- Scale up if throttled DB requests are > 1% of total DB requests (CONFIGURABLE)
- New RU throughput is increased as per this formula → New RU = Existing RUs + (existing RUs * % of throttling * scaling factor)

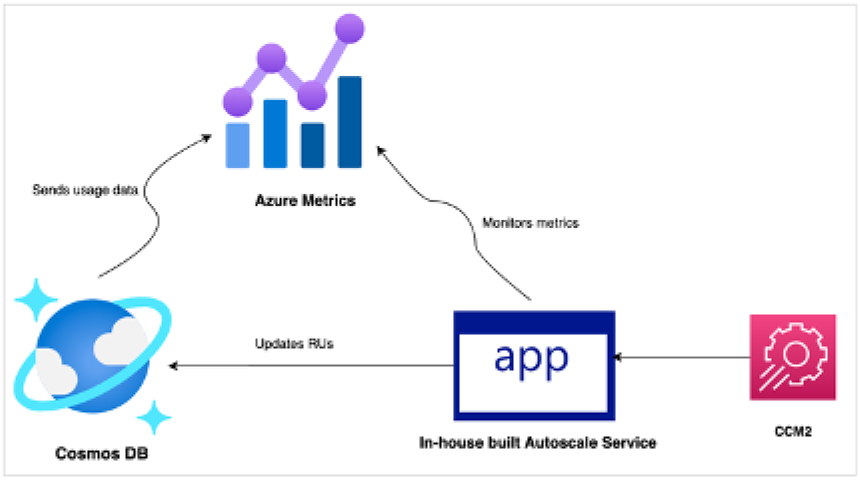
Scaling down condition:

- Scale down if normalized RU consumption is < 95% of configured RU (CONFIGURABLE)
- New RU throughput is % decrease on the unconsumed RUs
- Scaling down occurs on a cron after the 57th minute of every hour (You pay for the full hour)

Container Information: The autoscaling service pulls container information from a JSON record in CCM, which is retrieved on startup. The service will monitor any container from this list. Each container contains four required fields:

CCM config:

```
{
    "database": "mbaku",
    "container": "mbaku-snapshots-v2",
    "minRUThreshold": "30000",
    "maxRUThreshold": "199000",
    "throttledPctThreshold": "0.1", // Increase the RUs when ever .1% of
total DB request are throttled
    "scaleUpFactor": "10" // Scaling factor for the formula
}
```



To use this service, we need to provision throughput in manual mode and need wmt-cosmos-scaler role access to the cosmos account.

Results:

After enabling in-house autoscale service we had following impact:

- 4X Cloud Cost Savings
- Savings 620K USD annually

5. Conclusions

After doing the above mentioned optimizations we brought the cloud cost for Canada e-commerce IMS from 1M to 200K. Microsoft business team reached out to us to understand how we migrated the data from one container to another container as Microsoft Cosmos DB was not providing this capability out of the box. Our copy pipeline design can be used by any other Walmart team if they are facing a similar problem of higher partitions in Azure Cosmos DB. Our in-house autoscaling service concept can be leveraged to build similar service for other distributed databases to avoid dependency on cloud provided auto-scaling service.