



I have a scenario where my cosmos db has 2 million records which are size of 300kb each record, due to this my cost is spiked.

I structured below solution and need help if anything needs to be changed for better performance and cost optimisation

I built a azure solution to store recent records in cosmos db and records which are older than 3 months in cosmos db is moved to Blob storage using azure functions to save cost optimisation and will be retrieved when ever user requests the records.

Constraints

API call for existing read/write should not be changed

Need help to write azure function for data archival in python and retrieval layer to check first in cosmos db then fetch from blob if doesn't exist in db

Your solution of archiving older Cosmos DB records (>3 months) to Azure Blob Storage while keeping recent records in Cosmos DB is a common pattern for cost optimization at scale. Below, I'll address both the **Azure Function for data archival (Python)** and the **retrieval layer** with sample code and key recommendations.

1. Azure Function (Python) for Data Archival

Goal: Periodically move Cosmos DB records older than 3 months to Blob Storage and delete them from Cosmos DB.

Key Libraries

- azure-cosmos
- azure-storage-blob

Steps

1. Query Cosmos DB for records older than 3 months using a SQL query.
2. Upload each record to Blob Storage (recommended: use one blob per record, or batch if your access/fetch patterns allow).
3. Delete the record from Cosmos DB.

Sample Code Skeleton

```
import os
import datetime
import azure.functions as func
from azure.cosmos import CosmosClient
from azure.storage.blob import BlobServiceClient

# Config values (set as environment variables or Azure Function App settings)
COSMOS_URL = os.getenv('COSMOS_URL')
COSMOS_KEY = os.getenv('COSMOS_KEY')
COSMOS_DB = os.getenv('COSMOS_DB')
COSMOS_CONTAINER = os.getenv('COSMOS_CONTAINER')
BLOB_CONN_STR = os.getenv('BLOB_CONN_STR')
BLOB_CONTAINER = os.getenv('BLOB_CONTAINER')

def main(mytimer: func.TimerRequest) -> None:
    # Connect to Cosmos DB
    client = CosmosClient(COSMOS_URL, credential=COSMOS_KEY)
    db = client.get_database_client(COSMOS_DB)
    container = db.get_container_client(COSMOS_CONTAINER)

    # Connect to Blob Storage
    blob_service_client = BlobServiceClient.from_connection_string(BLOB_CONN_STR)
    blob_container_client = blob_service_client.get_container_client(BLOB_CONTAINER)

    # Calculate timestamp cutoff (3 months ago)
    cutoff_date = datetime.datetime.utcnow() - datetime.timedelta(days=90)

    # Query older records
    query = "SELECT * FROM c WHERE c.timestamp < @cutoff"
    params = [{"name": "@cutoff", "value": cutoff_date.isoformat()}]
    items = list(container.query_items(query=query, parameters=params, enable_cross_parti

    for item in items:
        item_id = item['id']
        # Write to blob as JSON
        blob_name = f"{item_id}.json"
        blob_container_client.upload_blob(blob_name, bytes(str(item), 'utf-8'), overwrite
        # Optionally, you can compress first (gzip, etc.) before upload

        # Delete item from Cosmos DB
        container.delete_item(item=item_id, partition_key=item['partitionKey'])

    print(f"Archived and deleted {len(items)} items.")
```

2. Retrieval Layer Pattern

Your API should **not change** for existing read/writes. For retrieval:

1. **Check Cosmos DB first** (current 0-3mo data).
2. **If not found, attempt fetch from Blob Storage.**

- If found, you may want to cache back to Cosmos DB (optional; control cost, respect TTL if needed).

Retrieval Pseudocode (Python)

```
def get_data(item_id, partition_key):
    # Attempt to fetch from Cosmos DB
    try:
        item = cosmos_container.read_item(item_id, partition_key=partition_key)
        return item
    except Exception:
        pass # Not found in Cosmos DB

    # Fetch from Blob Storage
    blob_name = f"{item_id}.json"
    blob_client = blob_container_client.get_blob_client(blob_name)
    if blob_client.exists():
        blob_data = blob_client.download_blob().readall()
        return json.loads(blob_data)
    # Item does not exist
    return None
```

Recommendations & Optimizations

- **Batch Operations:** For very large data moves, batch reads/deletes for Cosmos and batch uploads to Blob to improve throughput.
- **Blob Naming:** Use logical foldering (e.g., year/month/day/item_id.json) for efficient Blob Storage lookups and potential lifecycle policies.
- **Access Control:** Implement RBAC for blobs for security.
- **Compression:** Consider compressing large records before writing to Blob Storage.
- **Azure Function Triggers:** Use Timer Trigger for regular archival, or Logic Apps/Data Factory if you want more control.
- **Monitoring:** Log all moves and errors for audit and support.
- **Data Retention:** Configure Blob lifecycle management for further cost savings (cold/archive tier).
- **Schema Evolution:** Store a version number as metadata in blobs for possible schema upgrades in future.

Summary

- **Your architecture is well structured for cost optimization.**
- No need to change the API layer; handle Cosmos → Blob retrieval fallback internally.
- Use the provided Python sample code to quickly get started.

Let me know if you'd like more detailed code samples, deployment scripts, or help with additional Azure Function configurations!

What would be cost reduced due to this solution?

With your archival solution, here's a breakdown of the estimated **cost reduction**:

Calculation Details

- **Total records:** 2,000,000
- **Record size:** 300KB
- **Total data volume:** ~572GB (2,000,000 × 300KB = 572GB)
- **Cosmos DB storage cost:** ~\$0.25/GB/month^{[1] [2] [3]}
- **Azure Blob Storage (Cool tier) cost:** ~\$0.01/GB/month^{[4] [5] [6]}
- **Data retention:** 3 months in Cosmos DB (~143GB), 9 months in Blob (~429GB)

Monthly Storage Cost Comparison

Scenario	Monthly Cost
All data in Cosmos DB (~572GB)	~\$143
Optimized (3mo in Cosmos, 9mo in Blob)	~\$40
Monthly Cost Reduction	~\$103 (72%)

- **Old solution:** All data in Cosmos DB = ~\$143/month
- **New solution:** Only latest 3 months (~143GB) in Cosmos DB = ~\$36/month & Archived 9 months (~429GB) in Blob = ~\$4/month
= **Total ~\$40/month**
- **Savings per month:** ~\$103, or about **72% reduction** in your Cosmos DB storage costs just by archival, not counting additional savings on RU/s for read/write throughput, which may also decrease as older data access drops.

Notes

- Further savings are possible if you use Blob Storage archive tier.
- Actual costs may be affected by data redundancy, region, and access patterns.

This approach offers considerable monthly cost savings while keeping recent data performant and older records affordably available.



1. <https://azure.microsoft.com/en-us/pricing/details/cosmos-db/autoscale-provisioned/>
2. <https://intercept.cloud/en-gb/blogs/azure-cosmos-db-pricing>
3. <https://spot.io/resources/azure-pricing/azure-database-pricing-examples-and-5-ways-to-reduce-your-costs/>
4. <https://umbrellacost.com/blog/azure-storage-pricing-guide/>

5. <https://www.techtarget.com/searchstorage/tip/A-guide-to-Microsoft-Azure-storage-pricing>
6. <https://azure.microsoft.com/en-in/pricing/details/storage/blobs/>