# Cost Optimization Challenge: Managing Billing Records in Azure Serverless Architecture

We have a serverless architecture in Azure, where one of our services stores billing records in Azure Cosmos DB. The system is read-heavy, but records older than three months are rarely accessed.

Over the past few years, the database size has significantly grown, leading to increased costs. We need an efficient way to reduce costs while maintaining data availability.

## Current System Constraints

1. Record Size: Each billing record can be as large as 300 KB.

2.Total Records: The database currently holds over 2 million records.

3. Access Latency: When an old record is requested, it should still be served, with a response time in the order of seconds.

## Solution Requirements

#### Please propose a detailed solution to optimize costs while ensuring the following

1. Simplicity & Ease of Implementation – The solution should be straightforward to deploy and maintain.
2. No Data Loss & No Downtime – The transition should be seamless, without losing any records or requiring2 service downtime.
3. No Changes to API Contracts – The existing read/write APIs for billing records must remain unchanged

**RESULT**

1. **Architecture Diagram**

Client App → API (unchanged)

↓

Data Access Service

Cosmos DB │ ← Recent (< 3 mo)

↓

Check TTL → Archive Index (e.g., Azure Table or Cosmos DB index)

↓

Azure Blob Storage │ ← Archived (> 3 mo)

1. **Implementation Strategy**

**I ) Data Archival**

Use an Azure Function with a Timer Trigger to run daily

· Identify records older than 3 months using Cosmos DB \_ts (timestamp).

- Move data to Blob Storage (compressed JSON or Parquet).

-Store metadata (Record ID - to- Blob URI) in a lightweight index (Cosmos DB or Table Storage).

-Delete the archived record from the primary Cosmos DB after successful archival.

**Sample Python code**

for record in query\_cosmos("SELECT \* FROM c WHERE c.\_ts < cutoff\_date"):

blob\_name = f"{record.id}.json"

upload\_blob(container="archive", name=blob\_name, content=record)

upsert\_index(record.id, blob\_uri)

delete\_from\_cosmos(record.id)

**II)Transparent Read Access**

Update your DAL logic internally—no external API changes required.

**Sample Python code**

def get\_billing\_record(record\_id):

try:

return cosmos\_db.read(record\_id)

except NotFound:

uri = archive\_index.get(record\_id)

return read\_blob(uri)

**III ) No Downtime, Seamless Transition**

Phase 1: Copy older data to archive and maintain dual-write temporarily.

Phase 2: Once integrity validated, delete only archived records from Cosmos DB.

Phase 3: Switch read path in DAL - transparent to clients.

**Conclision :-**

Cost Benefits :

· **Azure Blob Storage** is dramatically cheaper than Cosmos DB RU/s or storage.

· Removes ~90% of old, rarely accessed data from costly Cosmos DB queries.

· Cold data retrieval via Blob is cost-efficient with Hot or Cool tier configs.

### \*\* Maintenance & Extensibility

Azure Data Factory or Logic Apps can replace the Azure Function for low-code maintenance.

Index can be enhanced with retrieval logs, audit history, etc.

Archive format supports Parquet for future Power BI reporting integration.