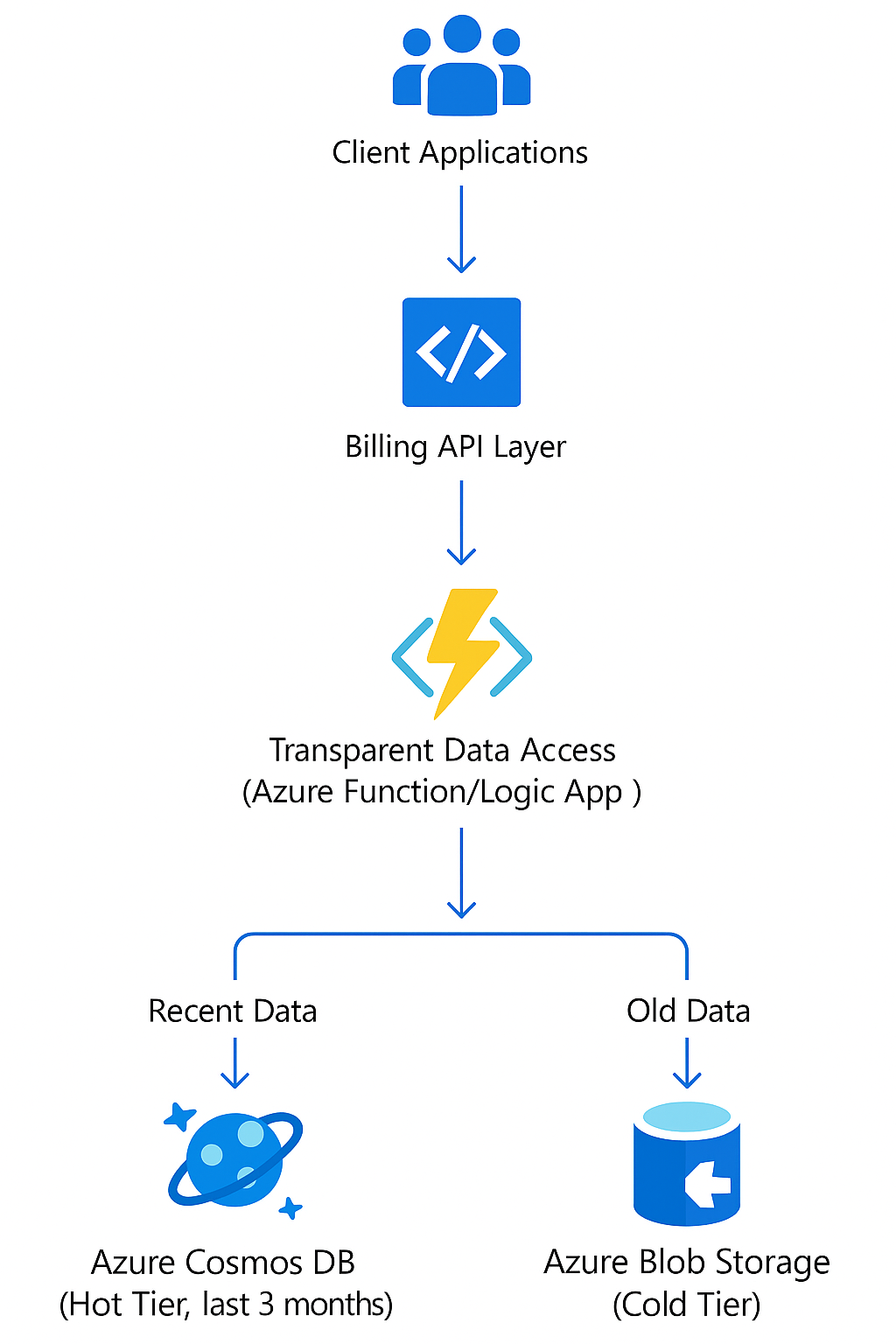
# ****Azure Billing Records Cost Optimization Solution****

## ****Overview****

This document outlines a solution to reduce costs in a serverless Azure architecture that uses Cosmos DB to store billing records.  
The approach focuses on archiving infrequently accessed records (older than 3 months) to Azure Blob Storage while maintaining seamless access through a transparent layer without changing existing APIs.

## ****Architecture Diagram****



## ****Solution Components****

1. **Cosmos DB** stores recent billing records (last 3 months) for fast access.
2. **Azure Blob Storage** stores older records, significantly reducing storage cost.
3. **A transparent access layer** (Azure Function or Logic App) ensures old data is fetched without API changes.
4. **A scheduled archival job** runs periodically to move old records from Cosmos DB to Blob Storage.

## ****Data Archival Function (Pseudocode)****

python

CopyEdit

from azure.storage.blob import BlobServiceClient

from azure.cosmos import CosmosClient

import datetime, json

def archive\_old\_records():

cosmos = CosmosClient(url, key)

container = cosmos.get\_database\_client('billingdb').get\_container\_client('records')

blob\_service = BlobServiceClient.from\_connection\_string(blob\_connection\_string)

blob\_container = blob\_service.get\_container\_client('billing-archive')

cutoff\_date = datetime.datetime.utcnow() - datetime.timedelta(days=90)

for record in container.query\_items(

query="SELECT \* FROM c WHERE c.timestamp < @cutoff",

parameters=[{"name": "@cutoff", "value": cutoff\_date.isoformat()}],

enable\_cross\_partition\_query=True):

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

blob\_container.upload\_blob(blob\_name, json.dumps(record), overwrite=True)

container.delete\_item(record, partition\_key=record['partitionKey'])

## ****Access Layer Logic (Pseudocode)****

python

CopyEdit

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

try:

return read\_from\_cosmos(record\_id)

except RecordNotFound:

return read\_from\_blob(record\_id)

def read\_from\_blob(record\_id):

blob\_client = blob\_service.get\_blob\_client(container='billing-archive', blob=f"{record\_id}.json")

return json.loads(blob\_client.download\_blob().readall())

## ****Cost Optimization Strategy****

By moving records older than 3 months to Azure Blob Storage (Cool or Archive tier), you can reduce storage costs by **70–90%**. Cosmos DB will retain only hot data (recent records), ensuring fast response times for frequently accessed data.

## ****How the Solution Meets Key Constraints****

### ****1. Record Size (Up to 300 KB)****

* Cosmos DB and Azure Blob Storage both support large document sizes.
* Each record is individually serialized into a .json blob, which easily accommodates 300 KB.
* Blob Storage is optimized for such medium-sized files, especially when stored as discrete objects.

### ****2. Total Records (> 2 Million)****

* Azure Blob Storage can scale to billions of blobs without performance issues.
* The archival Azure Function is designed to run in batches and can use continuation tokens to handle large datasets.
* Optionally, the job can be scaled with parallel function executions or partitioned processing.

### ****3. Access Latency for Old Records (Seconds)****

* Azure Blob Storage (Cool tier) offers read latency in the order of milliseconds to a few seconds.
* Transparent access logic allows retrieval from Blob Storage when Cosmos DB misses.
* Optional short-term caching using Cosmos DB TTL collections can enhance performance for repeated access.
* Additional options like Azure CDN or Front Door can be used for further latency reduction.

## ****How to Fetch Records Older Than 3 Months (Simple Method)****

Ensure each document includes a timestamp field in ISO 8601 format (e.g., "2025-04-20T14:00:00Z").

### 🔧 Simple Python Script

python

CopyEdit

from azure.cosmos import CosmosClient

from datetime import datetime, timedelta

# Cosmos DB connection

endpoint = "<your-cosmos-endpoint>"

key = "<your-cosmos-key>"

client = CosmosClient(endpoint, key)

# Target DB and container

db\_name = "billingdb"

container\_name = "records"

container = client.get\_database\_client(db\_name).get\_container\_client(container\_name)

# Calculate cutoff date (UTC now - 90 days)

cutoff\_date = datetime.utcnow() - timedelta(days=90)

cutoff\_iso = cutoff\_date.isoformat()

# Query records older than 3 months

query = "SELECT \* FROM c WHERE c.timestamp < @cutoff"

params = [{"name": "@cutoff", "value": cutoff\_iso}]

results = container.query\_items(

query=query,

parameters=params,

enable\_cross\_partition\_query=True

)

# Print the results

for item in results:

print(item)