1. Components (all serverless/Azure-native)

Cosmos DB - (hot tier for <90 days)

Azure Blob Storage - (archive container for ≥90 days)

Azure Table Storage - (index: `recordId → blobUri`)

Azure Function\*\* (timer-triggered archival + in-code read fallback)

2. High-Level Flow

Writes: unchanged → always go into Cosmos DB

Daily Archival Function

Query Cosmos for `/\_ts < now–90d/`

Serialize & compress each record to Blob (`.json.gz`)

mapping in Table Storage

Then delete from Cosmos

Reads (inside your DAL or Azure Function)

- Try `CosmosClient.Read(recordId)`

- On 404 → lookup Table Storage → `BlobClient.Download(recordId.json.gz)`

3. Why It’s Easy to Deploy & Operate

Zero new infr—all PaaS you already know

Tiny code footprint—azure-functions timer + a few SDK calls (<200 LOC)

No orchestration—single Function run on a schedule

No schema migrations—Table Storage is schemaless

No API changes—archival & fallback live purely in backend logic

Fully serverless—auto-scale, pay only for execution and storage

4. Step-by-Step Setup

1.Provision

- Create a Storage Account with:

- Blob container `billing-archive`

- Table `BillingIndex`

2. Archive Function (`TimerTrigger`)

- Query Cosmos:

3. Read Fallback - (in your existing service)

4.Test & Rollou

- Backfill 1 K records, verify reads from Blob

- Enable full schedule (e.g., every 6 hrs)

- Monitor Cosmos RUs and Blob accesses

5. Ongoing Maintenance

Logs & Alerts: Application Insights on Function failures

Blob Lifecycle: Add a 1-year automatic delete policy if desired

Cost Monitoring: Azure Cost Explorer + RU analytics on Cosmos