



Azure Cosmos DB

Technical Overview

Chris Joakim, Specialist / Global Black Belt, Charlotte

Hailey Wu, Cloud Solution Architect, Toronto

Xavier Guérette, Cloud Solution Architect, Montreal



AGENDA

Overview

- CosmosDB: What is it, and why use it?
- It's different. Growth Mindset

Main Features

- Change Feed
- TTL
- HTAP with Synapse Link
- Integrated Cache
- Integration Examples

The Basics

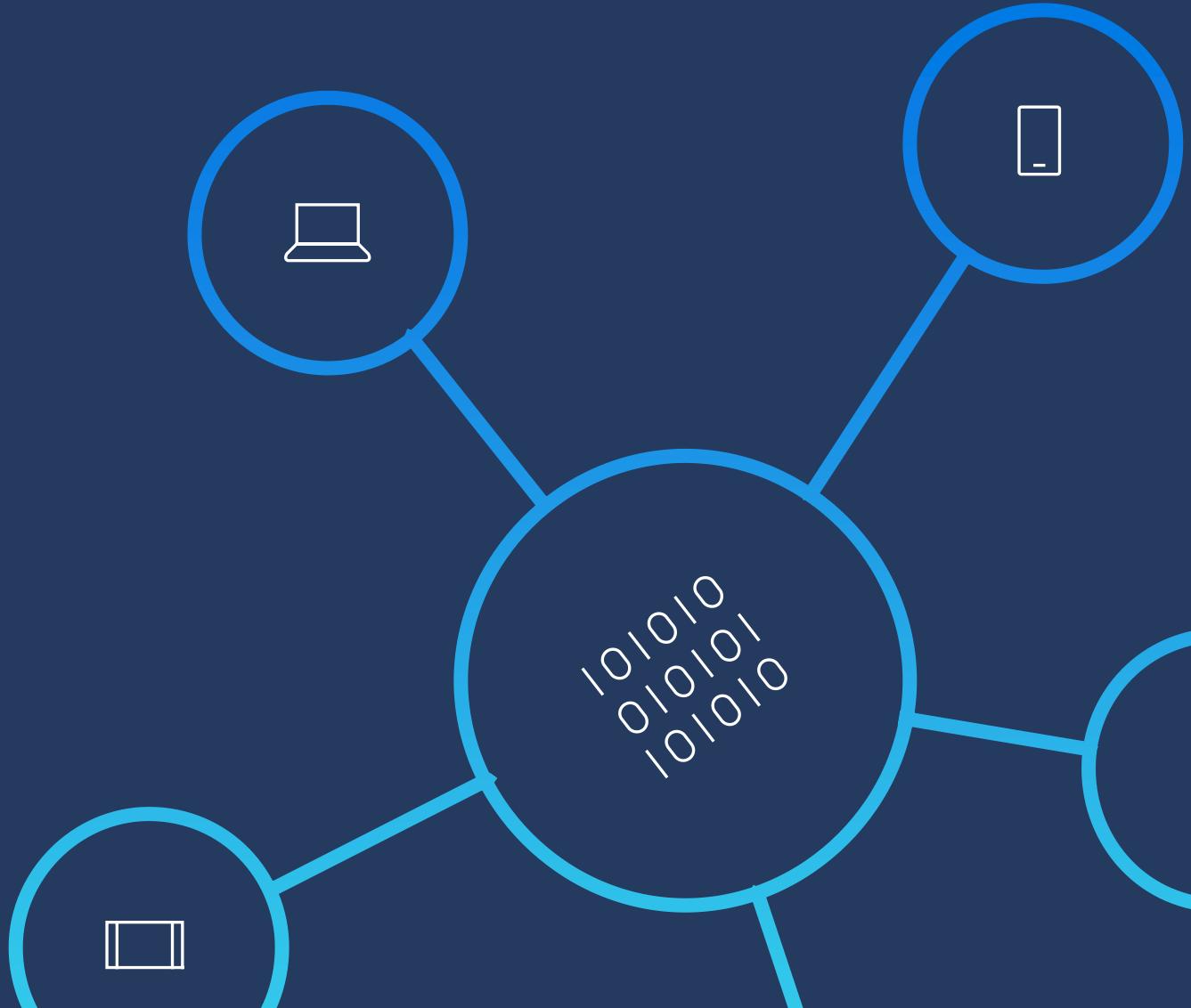
- Resource Model, JSON data
- Request Units (RU), Pricing, Scaling
- Physical and Logical Partitioning
- Global Distribution & Consistency Models
- HA/DR, Backups
- Indexing
- Logging / Azure Monitor / Kusto

Questions, Discussion, Next Steps

Next Session: Design and Development

- Design – patterns and anti-patterns
- Java – Spring, Spring Data, SDK

OVERVIEW



Cosmos DB

- A Family of NoSQL Databases. OLTP
- PaaS / Managed
- Scalable
- Distributed
- HA/DR, SLAs
- Secure
- Integrations

APIs

- Core (SQL) API (msft)
- Table (msft)
- MongoDB (oss)
- Cassandra (oss)
- Gremlin (oss)



SQL

SQL

{LEAF}

API for MongoDB



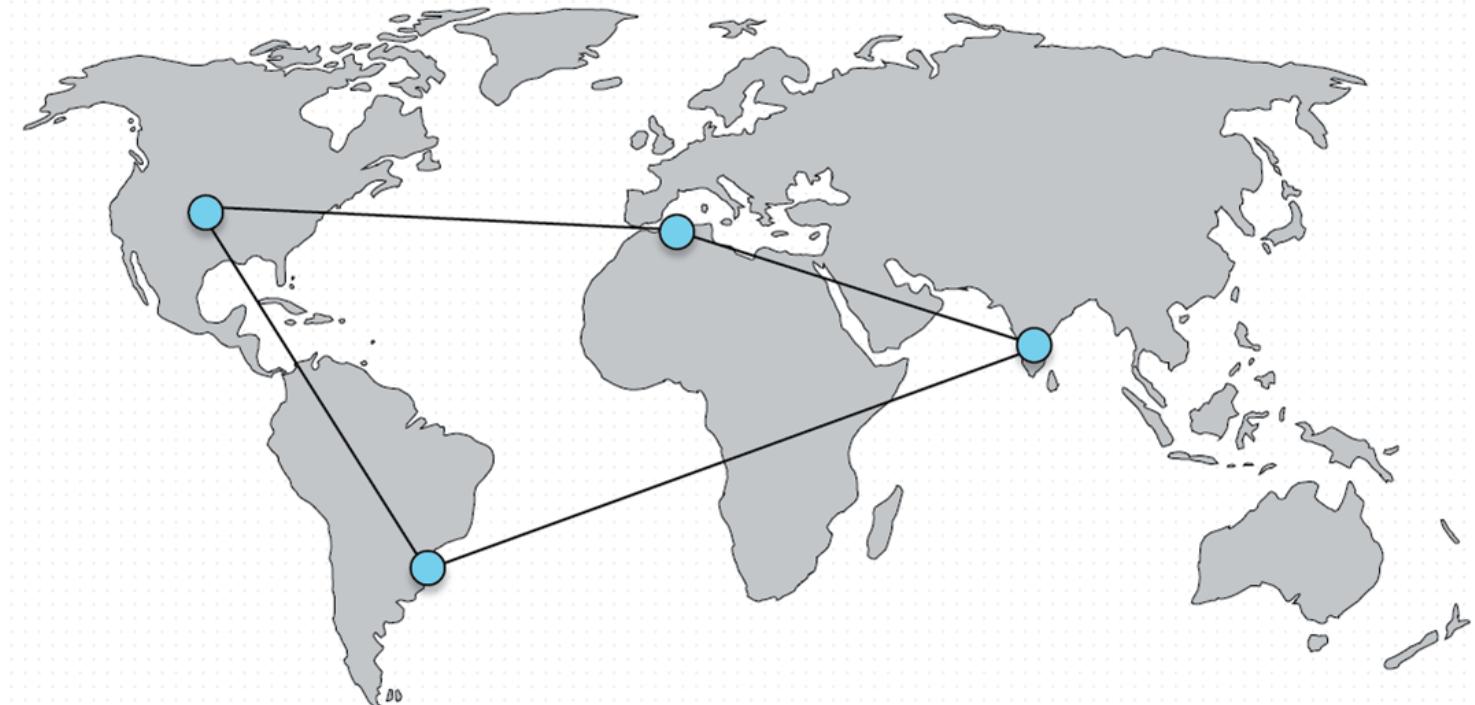
Gremlin



Cassandra



Table



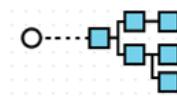
Key-Value



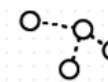
Column-Family



Documents



Graph



Guaranteed speed at any scale

Simplified application development

Mission-critical ready

Fully managed and cost effective

TOP 10 REASONS WHY CUSTOMERS USE AZURE COSMOS DB



The 1st and only database with **global distribution turnkey capability**



Very small to massively scalable in the same account



Guaranteed single digit millisecond latency at 99th percentile



Excellent integration with other Azure PaaS services



Boasts 5 well-defined consistency models to pick the right consistency/latency/throughput tradeoff



High SLA, suitable for mission critical applications



High flexibility to optimize for speed and cost. RU cost model.



Tackles **big data** workloads with **high availability and reliability**



Enterprise-grade security. In flight and on-disk encryption, BYOK, RBAC, Private Networking, etc.



Perfect for event-driven architectures as well as analytics. Ease of use.

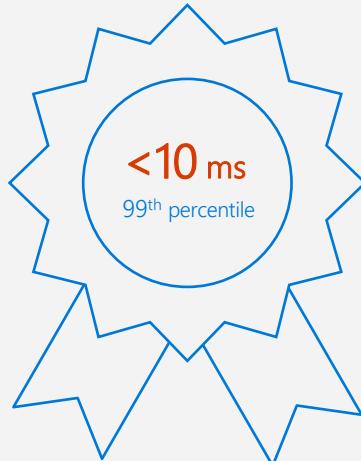
COMPREHENSIVE SLAs

COVERS AVAILABILITY AND LATENCY

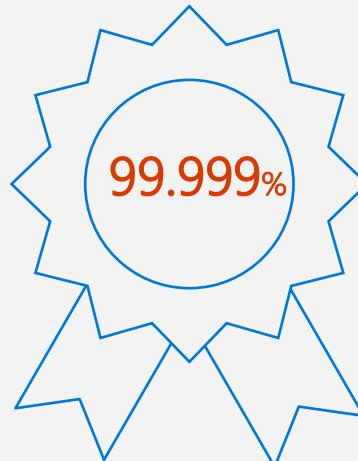
RUN YOUR APP ON WORLD-CLASS INFRASTRUCTURE

Azure Cosmos DB is the only service with **financially-backed SLAs for single-digit millisecond read and write latency** at the 99th percentile, **99.999%** high availability and guaranteed throughput and consistency

Latency



High Availability



Throughput



Consistency



GROWTH MINDSET; ALWAYS BE LEARNING PLEASE EMBRACE LEARNING COSMOSDB TO LEVERAGE IT BEST

Similar, but different:

- C# != Java
- CosmosDB != DB2
- CosmosDB != Atlas

Each programming language and DB has its' own Idioms/Patterns.

Please take the time to learn CosmosDB; it is different.

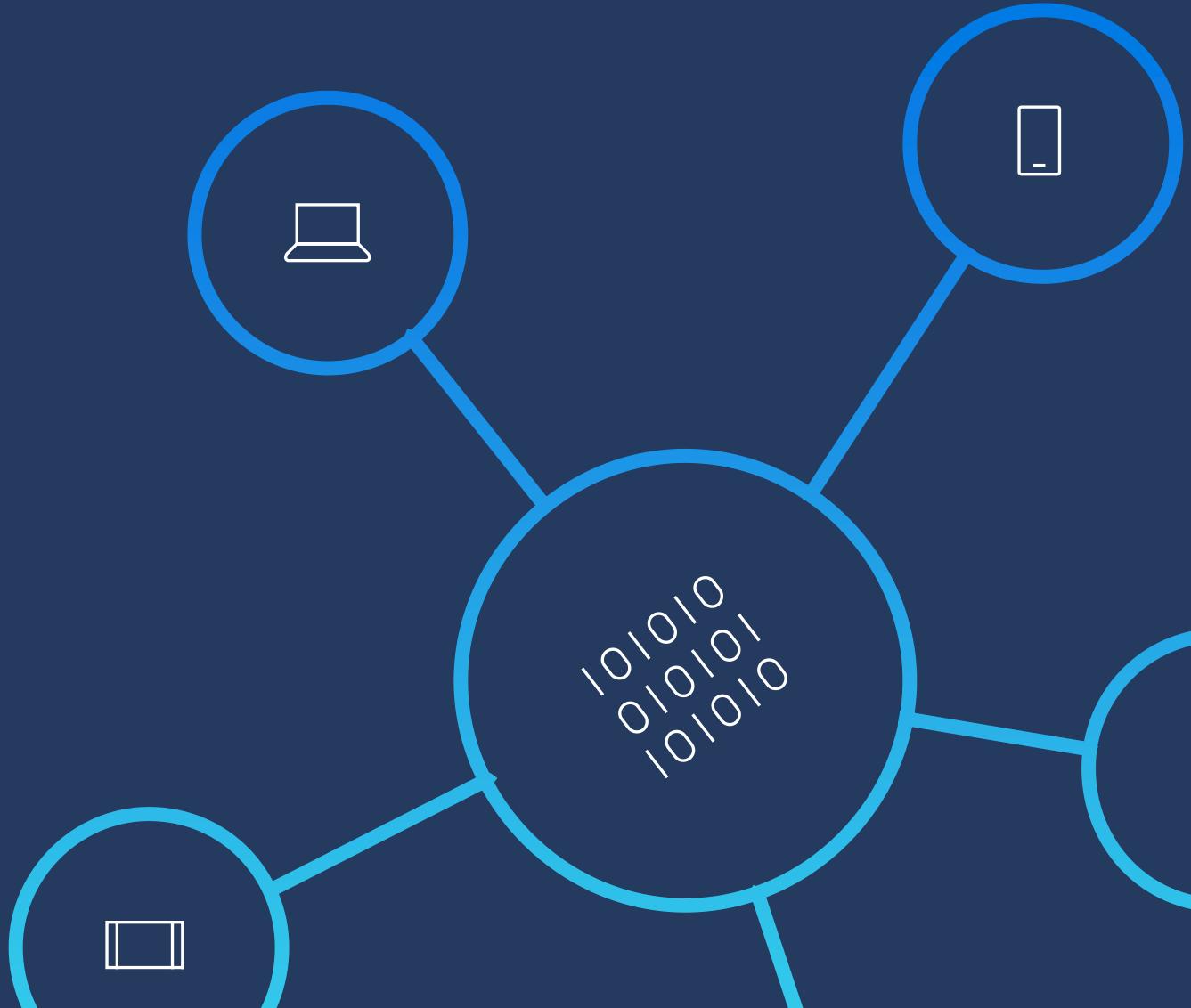
Growth Mindset

"We need to be always learning and insatiably curious. We need to be willing to lean in to uncertainty, take risks and move quickly when we make mistakes, recognizing failure happens along the way to mastery."

- Satya Nadella

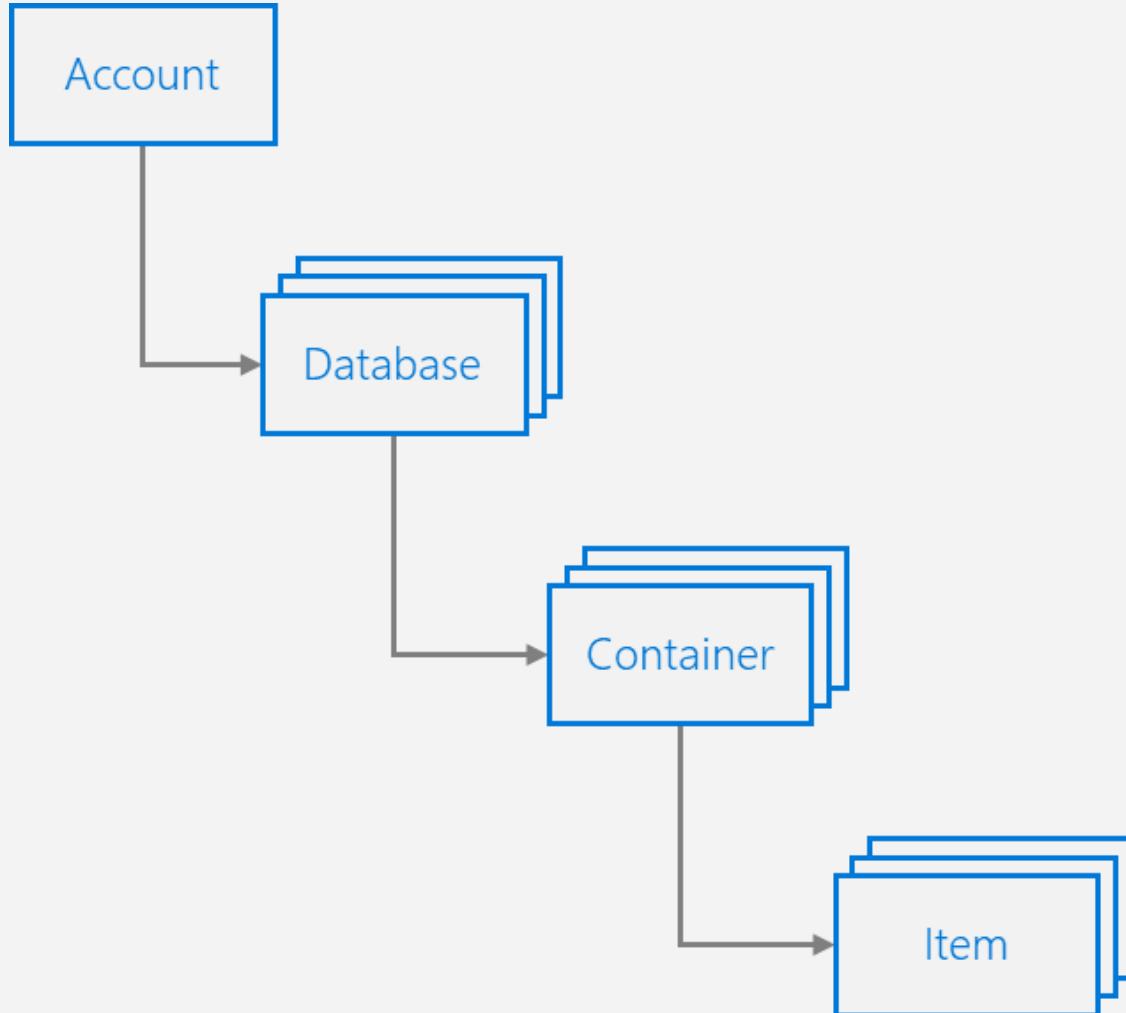


The Basics



THE STRUCTURE OF A COSMOSDB ACCOUNT

"CONTAINER" == "COLLECTION"



IS THE COSMOSDB SQL API RELATIONAL? NO!

- The data is **JSON**
- It's **Schemaless**
 - But requires a **partition key**
- Nesting
- 2MB max document size
- Indexing
- **Query it with SQL**

```
[  
 {  
   "id": "f38bd845-83da-4b31-8c1e-3668eb9ac26a",  
   "pk": "30.497478|-87.880258",  
   "stateCode": "01",  
   "countyCode": "003",  
   "siteNum": "0010",  
   "latitude": 30.497478,  
   "longitude": -87.880258,  
   "datum": "NAD83",  
   "metric": "Ozone",  
   "localDateTime": "2021-02-28 23:00",  
   "gmtDateTime": "2021-03-01 05:00",  
   "uom": "Parts per million",  
   "observationCount": 6,  
   "nullObservations": 1,  
   "meanObservation": 0.018,  
   "_rid": "gklzAKAAj3tX6wAAAAAACA==",  
   "_self": "dbs/gklzAA==/colls/gklzAKAAj3s=/docs/gklzAKAAj3tX6wAAAAAACA==/",  
   "_etag": "\"3c01959a-0000-0100-0000-6310d6120000\"",  
   "_attachments": "attachments/",  
   "_ts": 1662047762  
 }]  
 ]
```

```
SELECT * FROM c WHERE c.id = 'f38bd845-83da-4b31-8c1e-3668eb9ac26a' AND c.pk = '30.497478|-87.880258'
```

REQUEST UNITS – THE UNIT OF SCALE AND PRICE

- Serverless
- Provisioned
 - Manual Scale
 - **Auto Scale**
- RUs are also correlated to total data size
- Pay per Use - Uber ride vs buying a car

Data Explorer ...

New Container New Notebook Connect to GitHub Open Terminal

SQL API Home Scale

DATA

dev

- Scale
- altgraph
- telemetry
- testing
- users

graph

- Scale
- graph

NOTEBOOKS

Gallery

Throughput (autoscale)

Autoscale

Manual

Provision maximum RU/s required by this resource. Estimate your required RU/s with [capacity calculator](#)

Max RU/s *

5000

Your container throughput will automatically scale from **500 RU/s (10% of max RU/s) - 5000 RU/s** based on usage.

	Min Per Month	Max Per Month
Current Cost	\$ 43.80	\$ 438.00

(regions: 1, 500 RU/s - 5000 RU/s, \$0.00012/RU)

This cost is an estimate and may vary based on the regions where your account is deployed and potential discounts applied to your account

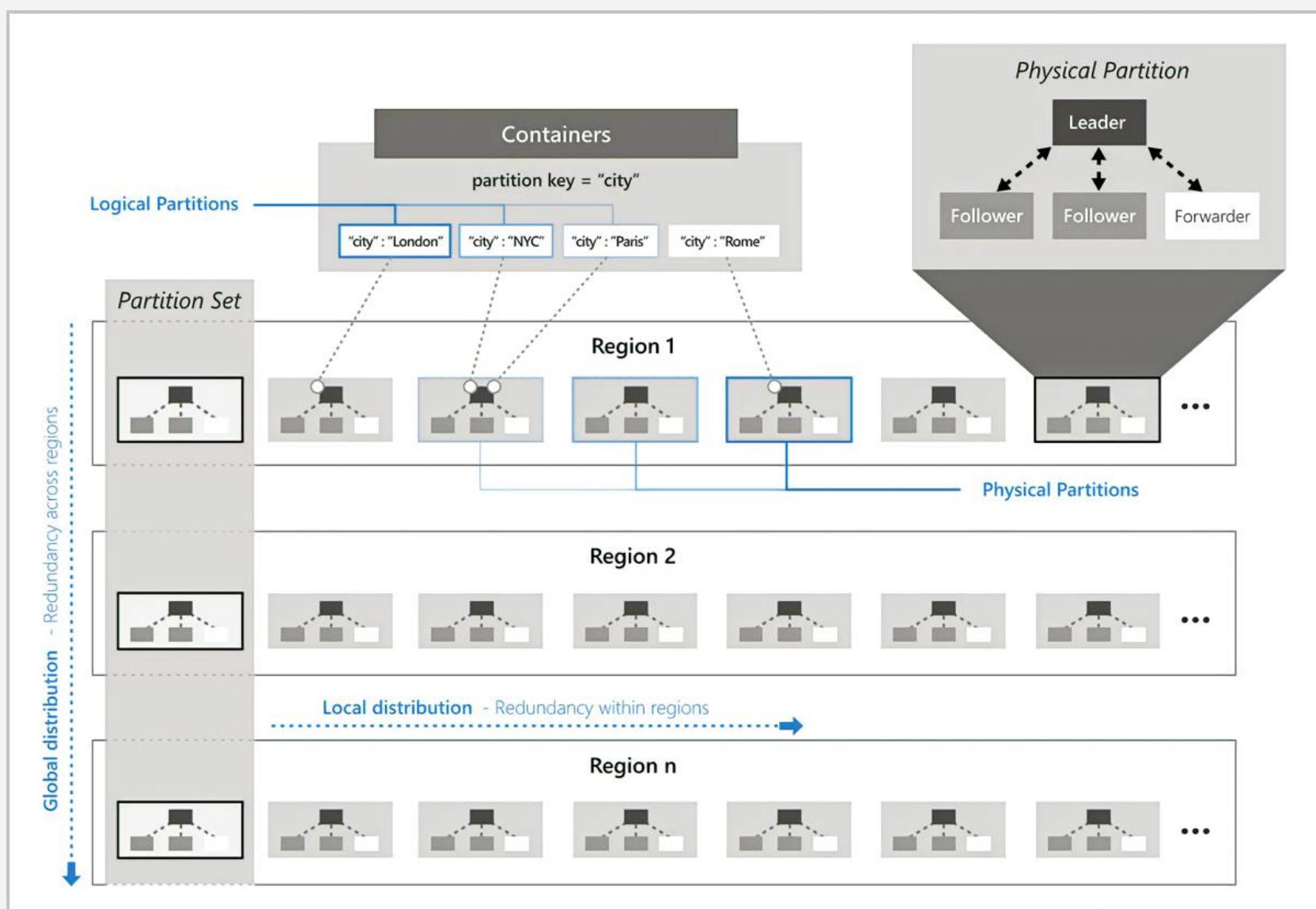
After the first 50 GB of data stored, the max RU/s will be automatically upgraded based on the new storage value. [Learn more](#).

REQUEST UNITS – THE UNIT OF SCALE AND PRICE

- Request Units (RU) are a scalable unit of Performance - Memory, CPU, and IOPS
- RUs are the primary cost component of CosmosDB
 - storage, network egress, backups, etc. are other cost components
- The cost to read a **1.0 KB document** by Id and Partition Key is **1.0 RU**
- Approx 5x for writes
- RUs can either be Manually or Auto-Scaled, at the container or DB level
- Think of Request Units as a configured "**Per-Second Budget**" of Throughput
- **429** error code when exceeding RU limit. Retry logic.
- **Your operation costs are provided by our SDKs, Azure Portal, Azure Monitor**
- "Burst" Capacity functionality is currently in preview mode
- **The RU cost model has design implications for your application**
 - The size of your documents matters, as well as the updates
- <https://docs.microsoft.com/en-us/azure/cosmos-db/request-units>

HORIZONTAL SCALING, PHYSICAL AND LOGICAL PARTITIONING

EACH CONTAINER HAS A FIXED PARTITION KEY ATTRIBUTE



REPLICATE TO ANY AZURE REGION(S)

USE A SINGLE REGION, OR MULTIPLE WRITE REGIONS (W/ FAILOVER CONFIG)

- Single Region
 - Availability Zone
- Multi-Region
 - Single Write Region
 - Multiple-Write Regions
 - 5 Consistency Levels
- Manual or **Service Managed Failover**
- Manual fail back
- Failover testing

Replicate data globally ...

Save Discard Manual failover Service-Managed Failover

⚠️ When you add a region to your account, you will be billed for the additional RU/s and storage copied to the region. Click here to learn more. →

Click on a location to add or remove regions from your Azure Cosmos DB account.
* Each region is billable based on the throughput and storage for the account. [Learn more](#)

Configure regions
Multi-region writes ⓘ
Disable Enable

Configure the regions for reads, writes and availability zone
+ Add region

Write region
East US

Read regions
North Europe

FIVE CONSISTENCY LEVELS FOR MULTIPLE WRITE REGIONS

Default consistency ...

 Save  Discard

STRONG

BOUNDED STALENESS

SESSION

CONSISTENT PREFIX

EVENTUAL

HIGH AVAILABILITY, DISASTER RECOVERY, BACKUPS

- Periodic or Continuous Backups
- Restores to *another* account
- Periodic Backups
 - You specify interval and retention
 - Support Ticket to execute a restore
- Continuous Backups
 - Granular in time
 - You execute the restore (PITR)
- Also, Synapse Link (discussed later)

Backup & Restore ...

Your account is on periodic backup mode. You can now change to continuous mode for a better backup and restore experience. [Change to continuous mode](#)

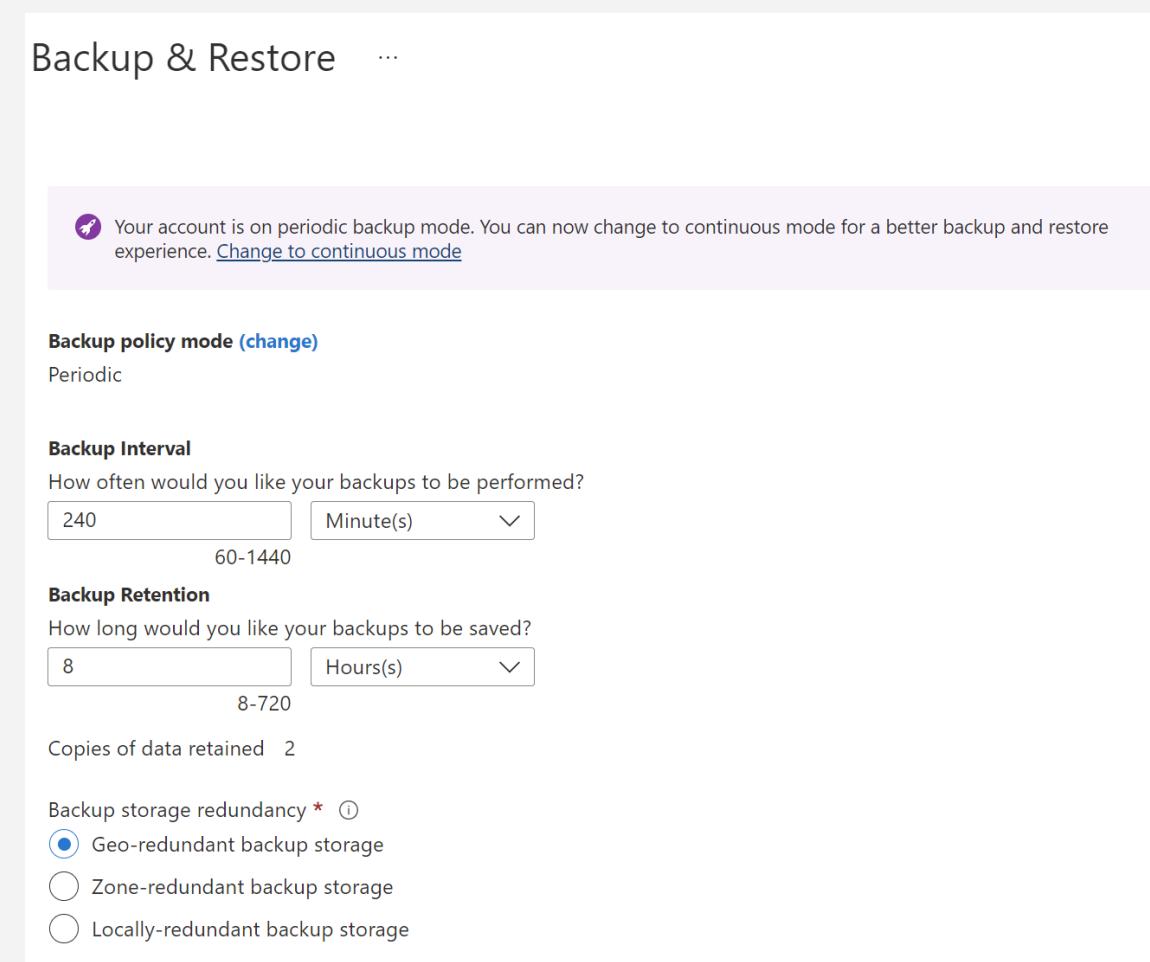
Backup policy mode (change)
Periodic

Backup Interval
How often would you like your backups to be performed?
240 Minute(s) 60-1440

Backup Retention
How long would you like your backups to be saved?
8 Hours(s) 8-720

Copies of data retained 2

Backup storage redundancy * ⓘ
 Geo-redundant backup storage
 Zone-redundant backup storage
 Locally-redundant backup storage



LOGGING, MONITORING, ALERTS

SLIDE 1 – EXAMPLE CLASSIC (ORIGINAL) INHERENT METRICS

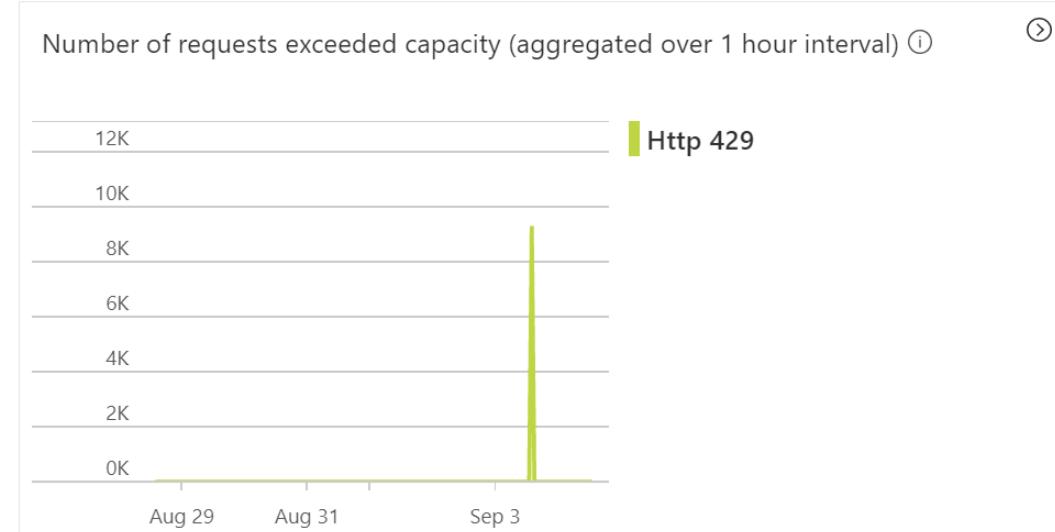
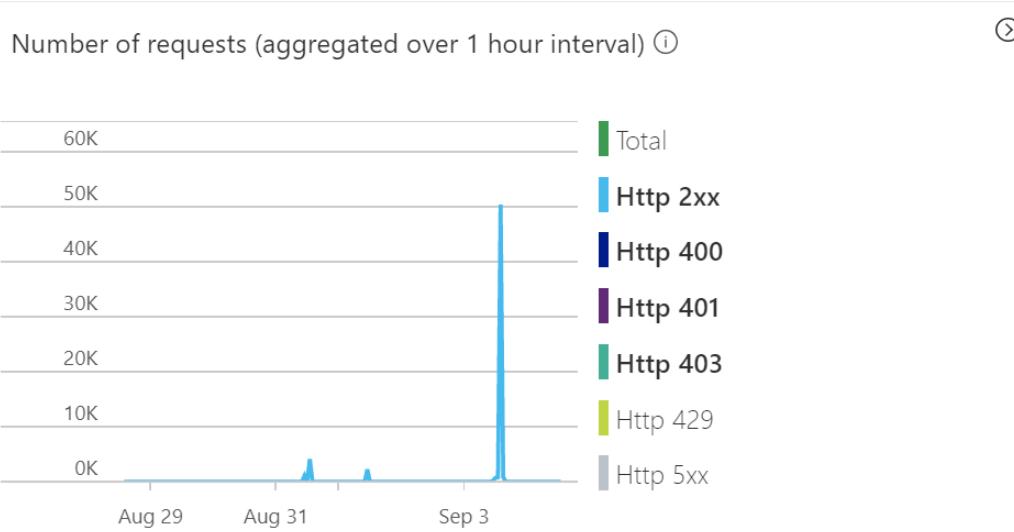
Metrics (Classic) ...

[Download as csv](#) [Refresh](#) [Feedback](#)

Overview **Throughput** Storage Availability Latency Consistency System

Database(s) Container(s) Region(s) Show data for last

Collection telemetry has provisioned throughput at the database level.



LOGGING, MONITORING, ALERTS

SLIDE 2 – AZURE MONITOR - WHAT GETS LOGGED, AND TO WHERE

- Use the newer **Resource Specific** Destination
- These correlate to the simpler “**CDBxxx**” tables in Azure Monitor

The screenshot shows the 'Diagnostic setting' configuration page in the Azure portal. The top navigation bar includes 'Dashboard', 'cjoakimcosmssql | Diagnostic settings', 'Diagnostic settings', and a save/discard/delete/feedback button. The main content area has two main sections: 'Logs' and 'Metrics'. Under 'Logs', there is a 'Categories' section with several checkboxes, some of which are checked (e.g., DataPlaneRequests, QueryRuntimeStatistics, PartitionKeyStatistics, PartitionKeyRUConsumption, ControlPlaneRequests). Under 'Metrics', there is a single checked checkbox for 'Requests'. To the right of these sections is a 'Destination details' panel. It contains a checked checkbox for 'Send to Log Analytics workspace', a dropdown for 'Subscription' set to 'Microsoft Azure Internal Consumption', a dropdown for 'Log Analytics workspace' set to 'cjoakimla (eastus)', and a radio button for 'Destination table' set to 'Resource specific'. There are also three unchecked checkboxes for 'Archive to a storage account', 'Stream to an event hub', and 'Send to partner solution'.

LOGGING, MONITORING, ALERTS

SLIDE 3 – AZURE MONITOR QUERIES WITH THE KUSTO SYNTAX

- How to get partition key statistics to evaluate skew across top 3 partitions for a database account:

Kusto

 Copy

```
CDBPartitionKeyStatistics  
| project RegionName, DatabaseName, CollectionName, PartitionKey, SizeKb
```

- How to get the request charges for expensive queries?

Kusto

 Copy

```
CDBDataPlaneRequests  
| where todouble(RequestCharge) > 10.0  
| project ActivityId, RequestCharge  
| join kind= inner (  
CDBQueryRuntimeStatistics  
| project ActivityId, QueryText  
) on $left.ActivityId == $right.ActivityId  
| order by RequestCharge desc  
| limit 100
```

<https://docs.microsoft.com/en-us/azure/cosmos-db/cosmosdb-monitor-logs-basic-queries>

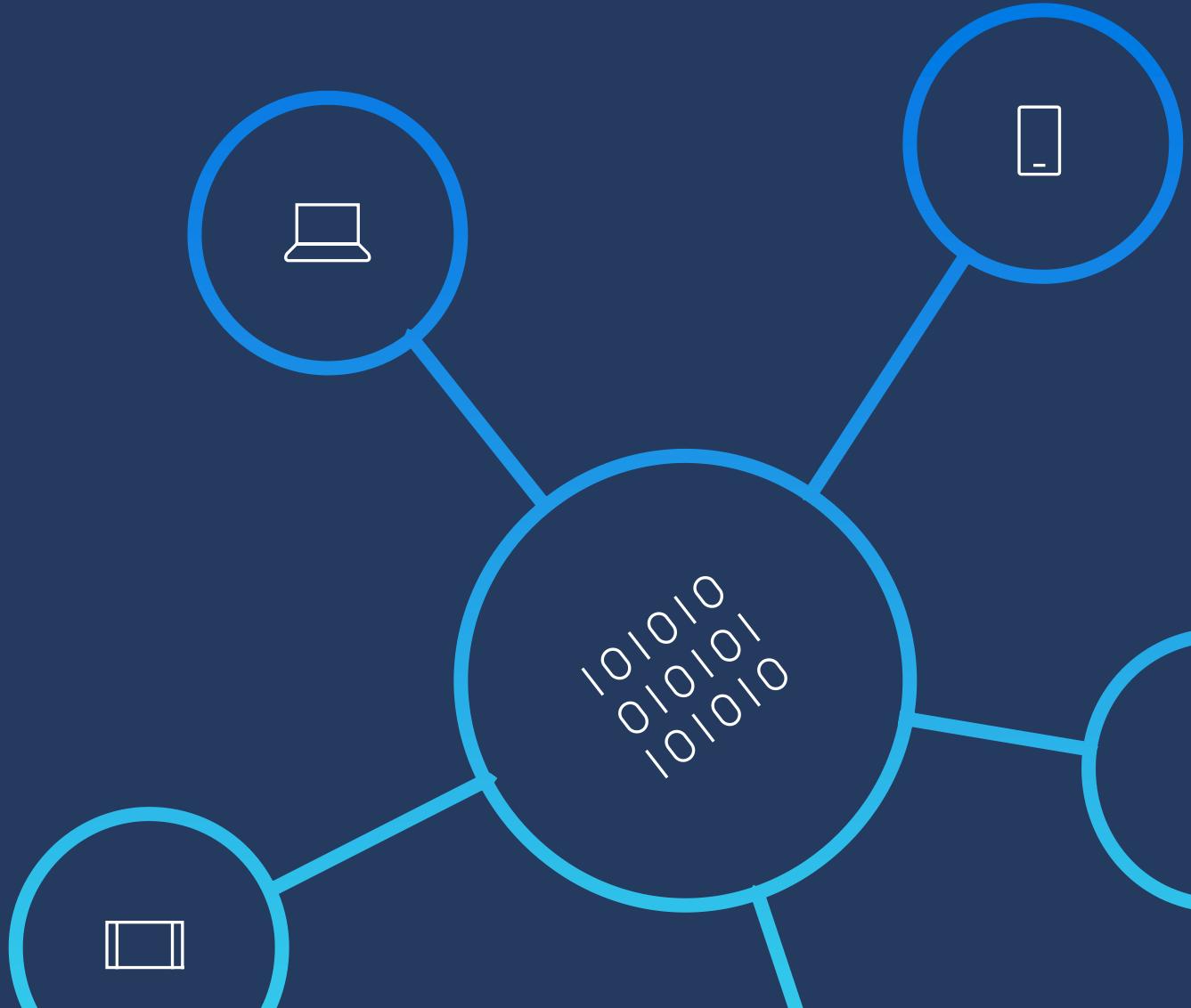
<https://docs.microsoft.com/en-us/azure/cosmos-db/cosmos-db-advanced-queries?tabs=resource-specific>

INDEXING

- Defined with JSON
- Improves query performance
- Types:
 - Range
 - Spatial (GPS)
 - Composite
- **Default Indexing**
 - All attributes, All Levels
- excludePaths
- **compositIndexes**
 - WHERE clause
 - ORDER BY clause
 - Field sequence matters

```
{  
  "indexingMode": "consistent",  
  "automatic": true,  
  "includedPaths": [  
    {  
      "path": "/"  
    }  
  ],  
  "excludedPaths": [  
    {  
      "path": "/\\"_etag\\"/?"  
    }  
  ],  
  "compositeIndexes": [  
    [  
      {  
        "path": "/pk",  
        "order": "ascending"  
      },  
      {  
        "path": "/id",  
        "order": "descending"  
      },  
      {  
        "path": "/doctype",  
        "order": "descending"  
      }  
    ]  
  ]  
}
```

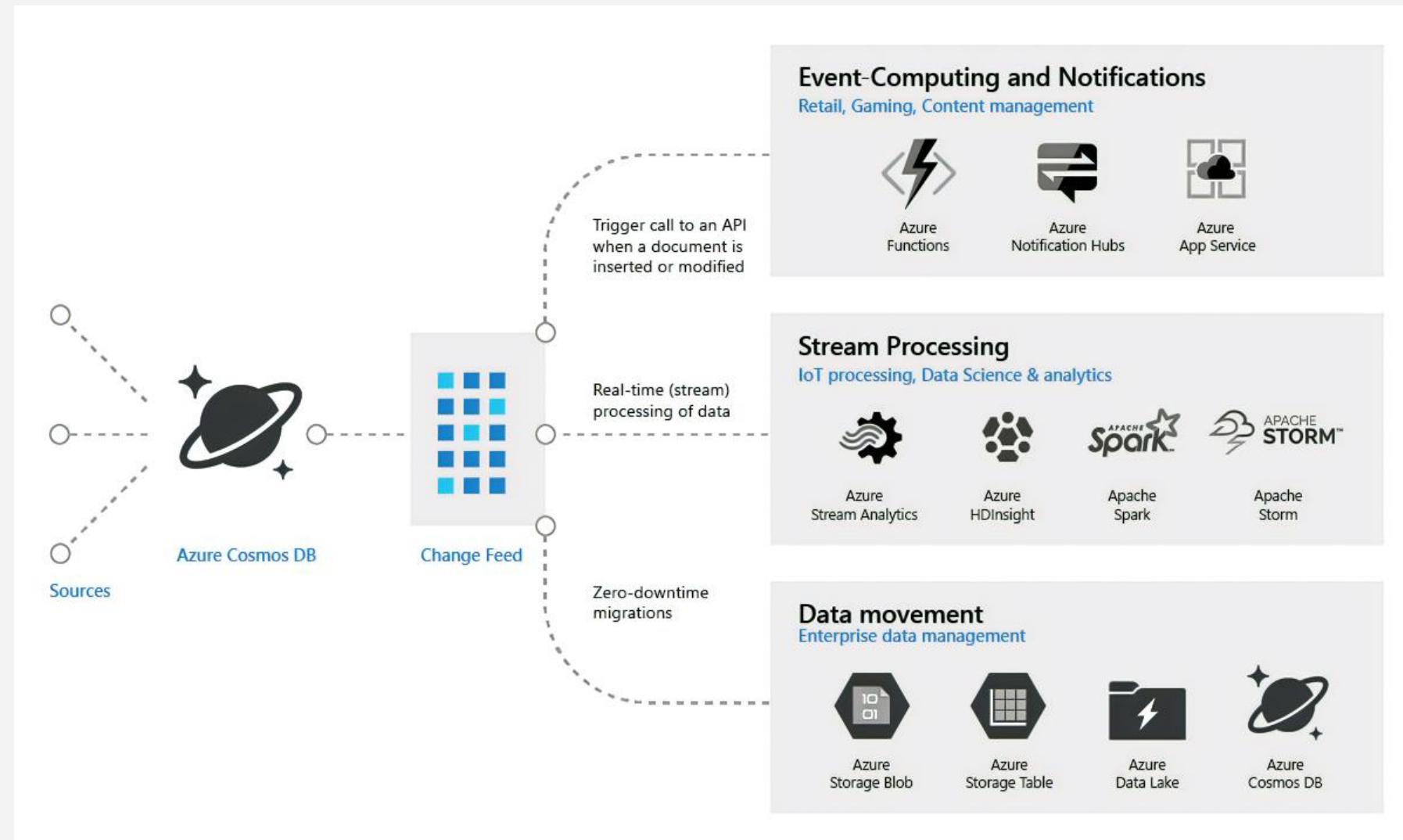
Main Features



COSMOSDB CHANGE FEED FOR REALTIME EVENT-DRIVEN APPLICATIONS

- Change Feed is a stream of ordered changed documents in a container that you can observe. No deletes, no diffs.

- **Event-Driven Apps**
 - Pipelines
- Consumers
 - **Azure Functions**
 - SDKs (Java, C#)
- **Leases Collection**
 - Consumer State



TIME-TO-LIVE (TTL) AUTOMATIC DOCUMENT DELETION WITH ZERO RU COSTS

- Enable TTL on your container
- Set a TTL duration in number of seconds
- Uses the system-generated `_ts` attribute
- Zero RU costs to delete the document

```
1 {  
2     "id": "726a44ab-856f-4027-b429-312c30250a79",  
3     "pk": "33.904039|-86.053867",  
4     "stateCode": "01",  
5     "countyCode": "055",  
6     "siteNum": "0011",  
7     "latitude": 33.904039,  
8     "longitude": -86.053867,  
9     "datum": "NAD83",  
10    "metric": "Ozone",  
11    "localDateTime": "2021-02-28 23:00",  
12    "gmtDateTime": "2021-03-01 05:00",  
13    "uom": "Parts per million",  
14    "observationCount": 6,  
15    "nullObservations": 1,  
16    "meanObservation": 0.029,  
17    "_rid": "gklzAKAAj3tvNwAAAAAAA==",  
18    "_self": "dbs/gklzAA==/colls/gklzAKAAj3s=/docs/gklzAKAAj3tvNwAAAAAAA==/",  
19    "_etag": "\"9103d226-0000-0100-0000-6313a0960000\"",  
20    "_attachments": "attachments/",  
21    "_ts": 1662230678  
22 }
```

Data Explorer

SQL API Home Settings

DATA

dev

telemetry

Settings

Indexing Policy

Time to Live

Off

On (no default)

On

2592000 second(s)

Geospatial Configuration

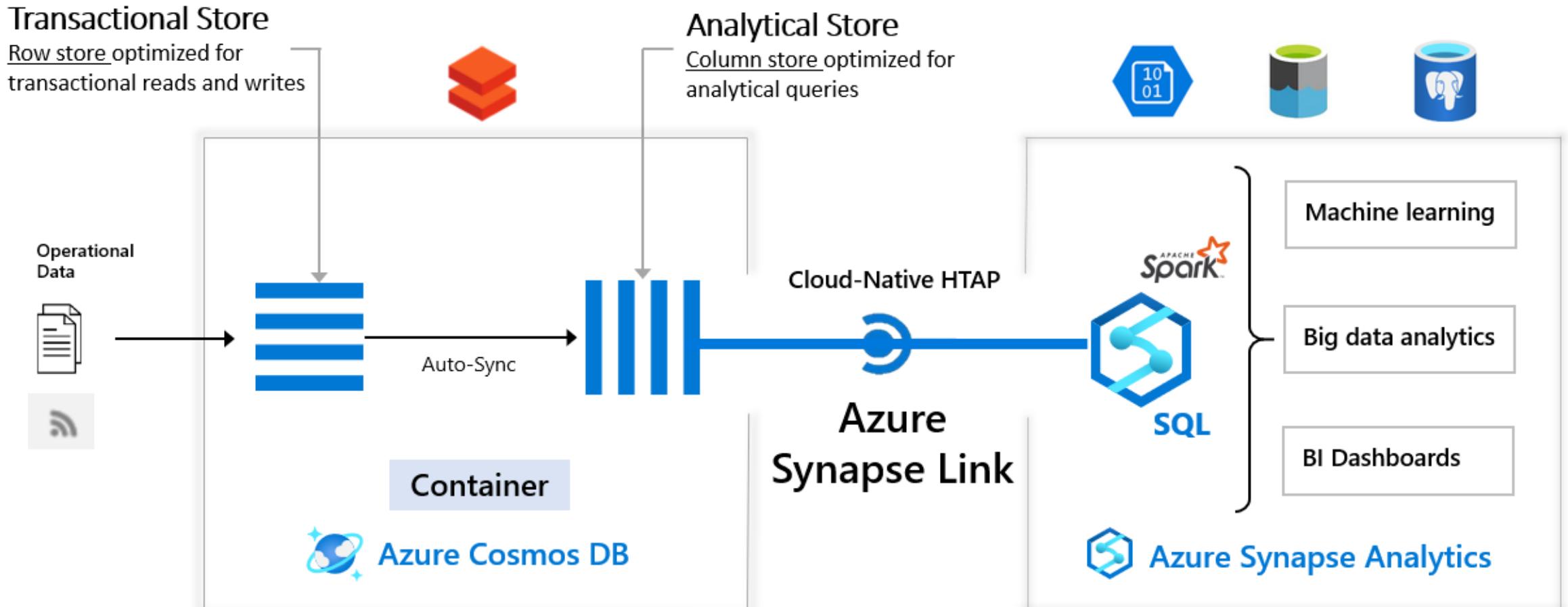
Graph

Scale

graph

HTAP WITH SYNAPSE LINK

HYBRID TRANSACTIONAL ANALYTICAL PROCESSING (OLTP & OLAP)



HTAP WITH SYNAPSE LINK

READING THE ANALYTIC DATASTORE WITH SPARK IN AZURE SYNAPSE

- SQL and Spark Compute Engines in Azure Synapse
- This example is Spark with PySpark (Python)

```
1 # Load the SynapseLink Sales Data into a Dataframe.
2 # Select just the "sale" document types from the sales container,
3 # which have a minimum _ts (timestamp) value
4
5 from pyspark.sql.functions import col
6
7 # initialize variables
8 begin_timestamp = 0
9 end_timestamp   = 1699999999
10
11 # read just the doctype "sales", not "line_item"
12 # "cosmos.oltp" = CosmosDB live database
13 # "cosmos.olap" = Synapse Link Analytic Datastore
14
15 df_sales = spark.read\
16     .format("cosmos.olap")\
17     .option("spark.synapse.linkedService", "CosmosSqlDemoDB")\
18     .option("spark.cosmos.container", "sales")\
19     .load().filter(col("doctype") == "sale")\
20     .filter(col("_ts") > begin_timestamp)\
21     .filter(col("_ts") < end_timestamp)
22
23 display(df_sales.limit(3))
```

INTEGRATED CACHE

OPTIONAL CACHE, AND GATEWAY VS DIRECT MODE

Connection Modes:

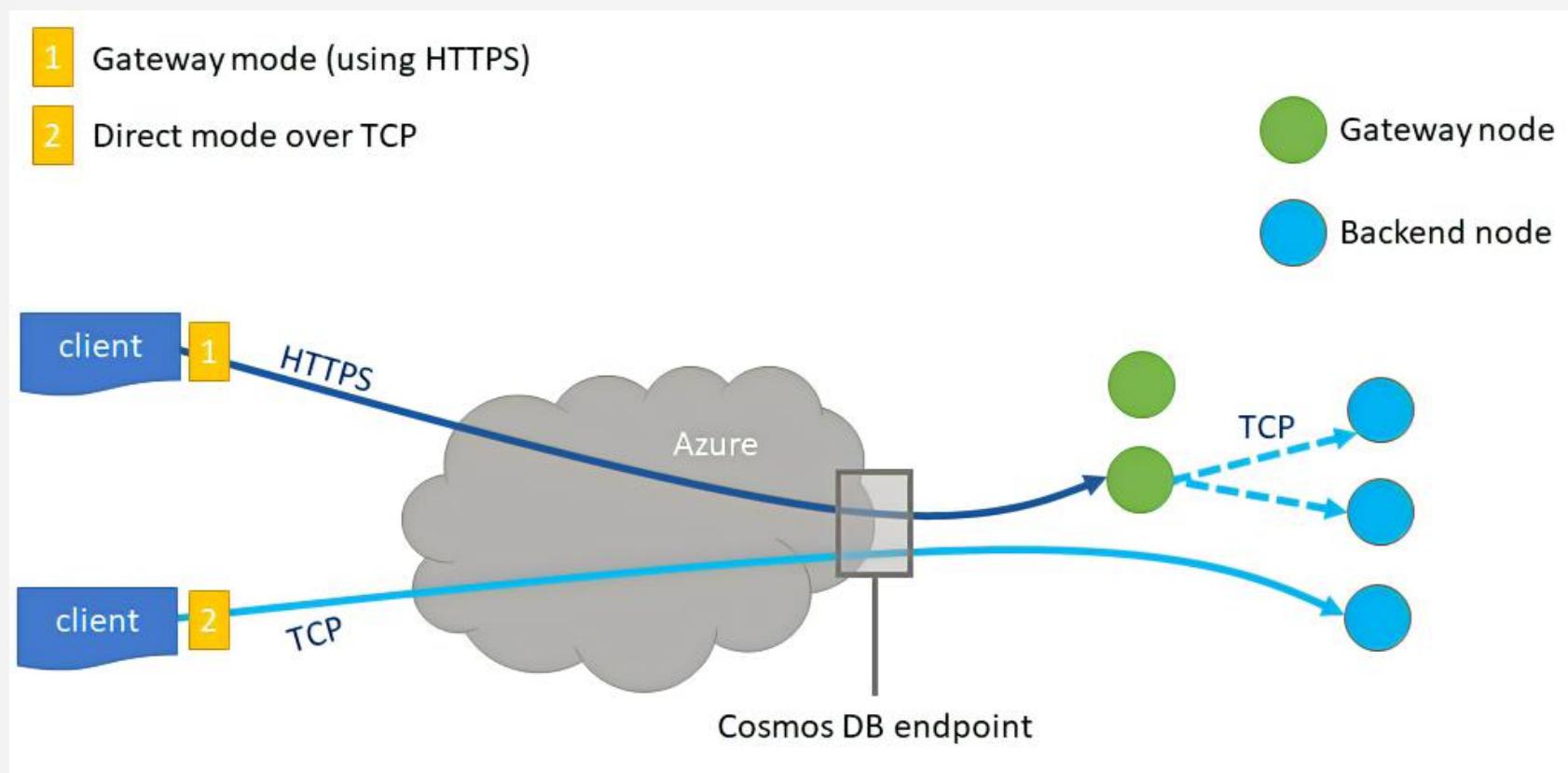
- HTTPS (Gateway)
- TCP (Direct)

Gateways:

- Standard (default)
- **Dedicated (cache)**

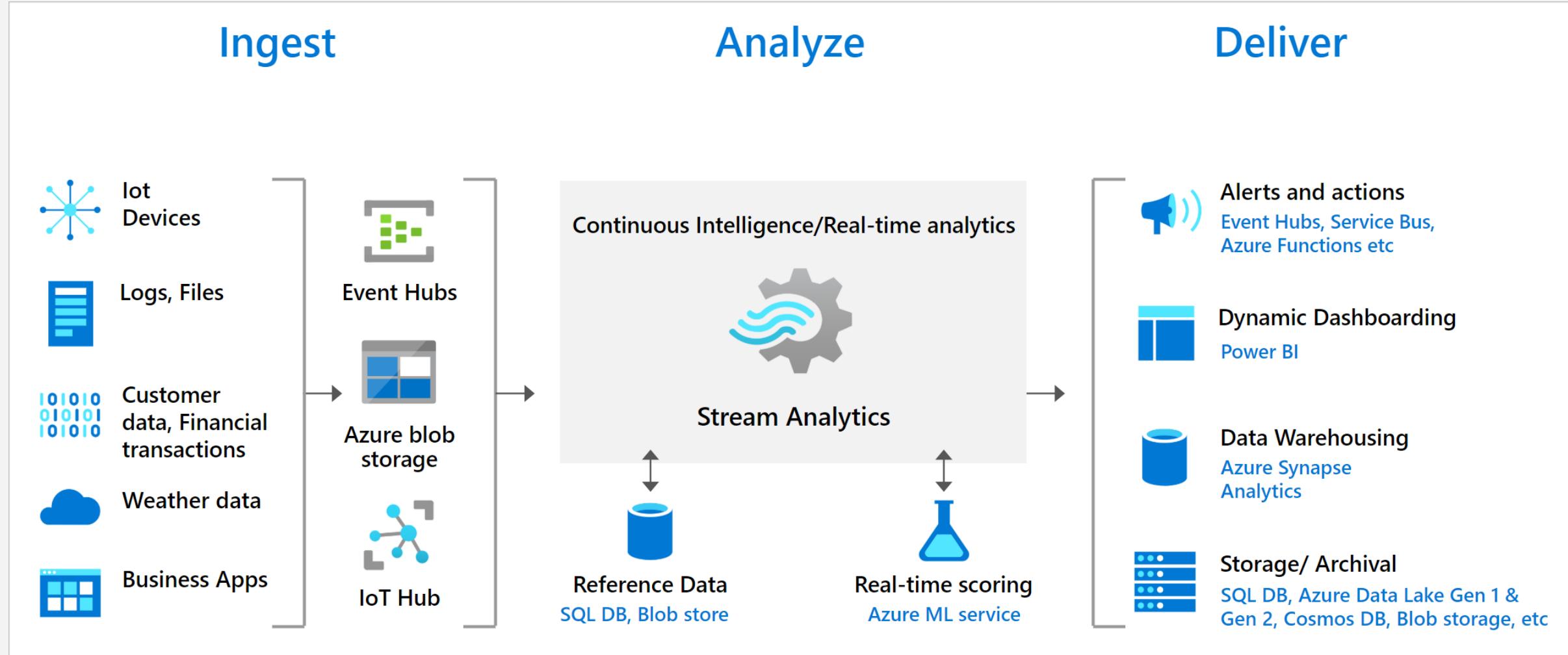
Caches:

- Sizes – S, M, L
- Types – Query, Item
- **LRU algorithm**
- **Auto-populated**



INTEGRATION EXAMPLE: STREAM ANALYTICS

SLIDE 1: STREAM FROM SOURCE TO SINK, WITH OPTIONAL AI



INTEGRATION EXAMPLE: STREAM ANALYTICS

SLIDE 2: SQL "CODE" – SELECT FROM SOURCE INTO SINK

 **cjoakim-sa - Query**
Stream Analytics job

Search (Ctrl+ /) Save Discard Test

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Inputs (1)
cjoakim-iot-hub

Outputs (1)
cosmosdb-iot-collection

Need help with your query? Check out some of the most common queries.

```
1 SELECT
2 *
3 INTO
4 [cosmosdb-iot-collection]
5 FROM
6 [cjoakim-iot-hub]
```

...

Settings

Locks

Job topology

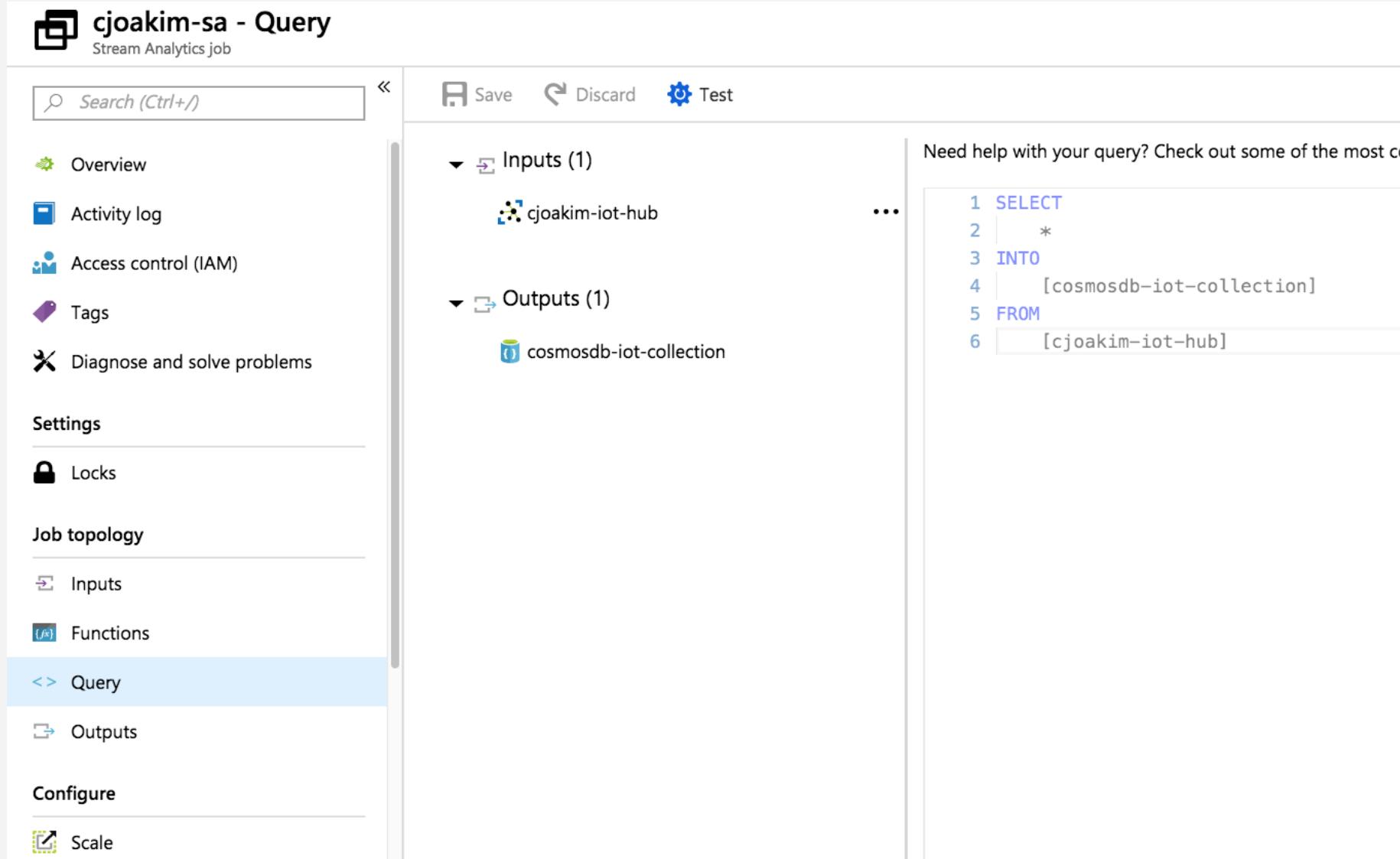
Inputs Functions

Query

Outputs

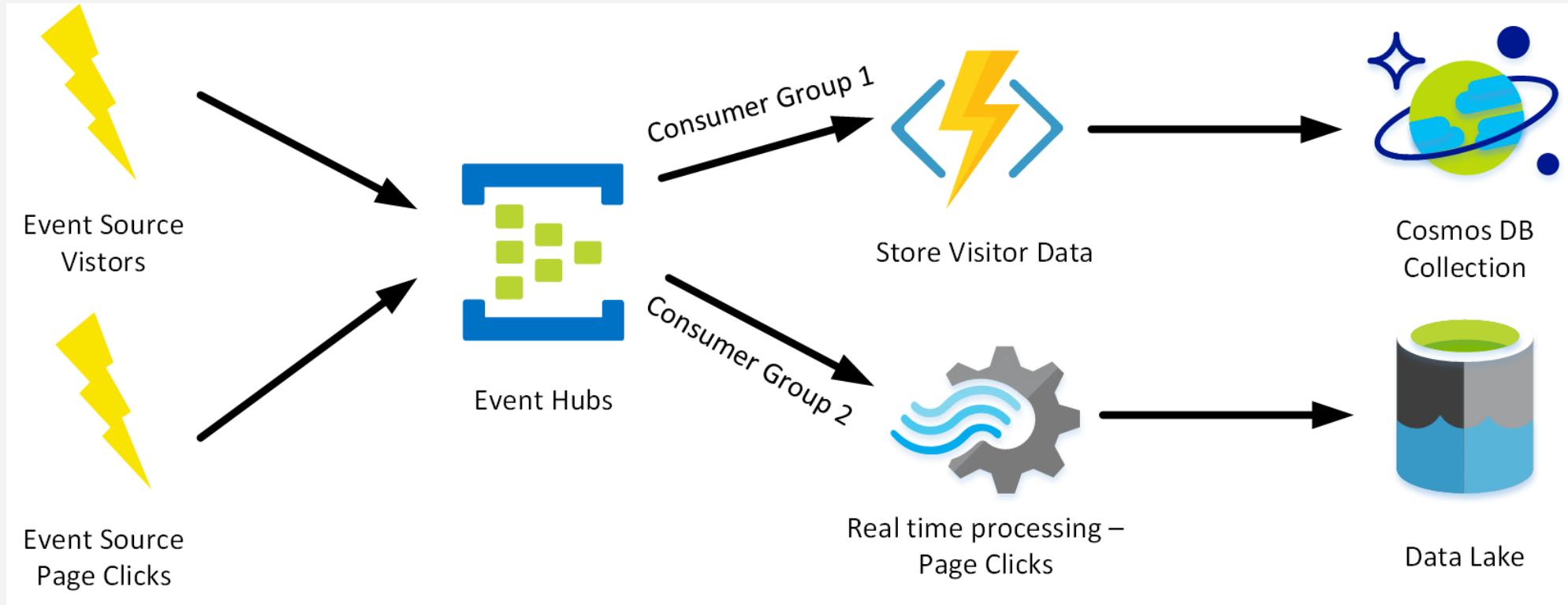
Configure

Scale



INTEGRATION EXAMPLE: STREAMING

Easily Integrate Azure PaaS Services. Low-code, No-Code

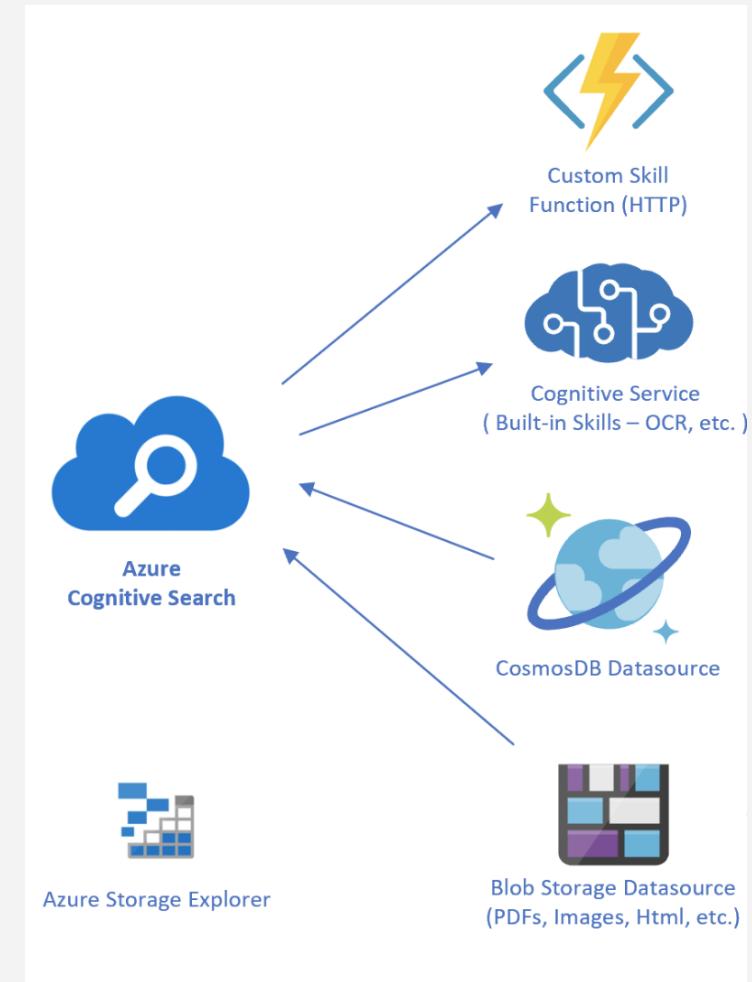


INTEGRATION EXAMPLE: AZURE COGNITIVE SEARCH

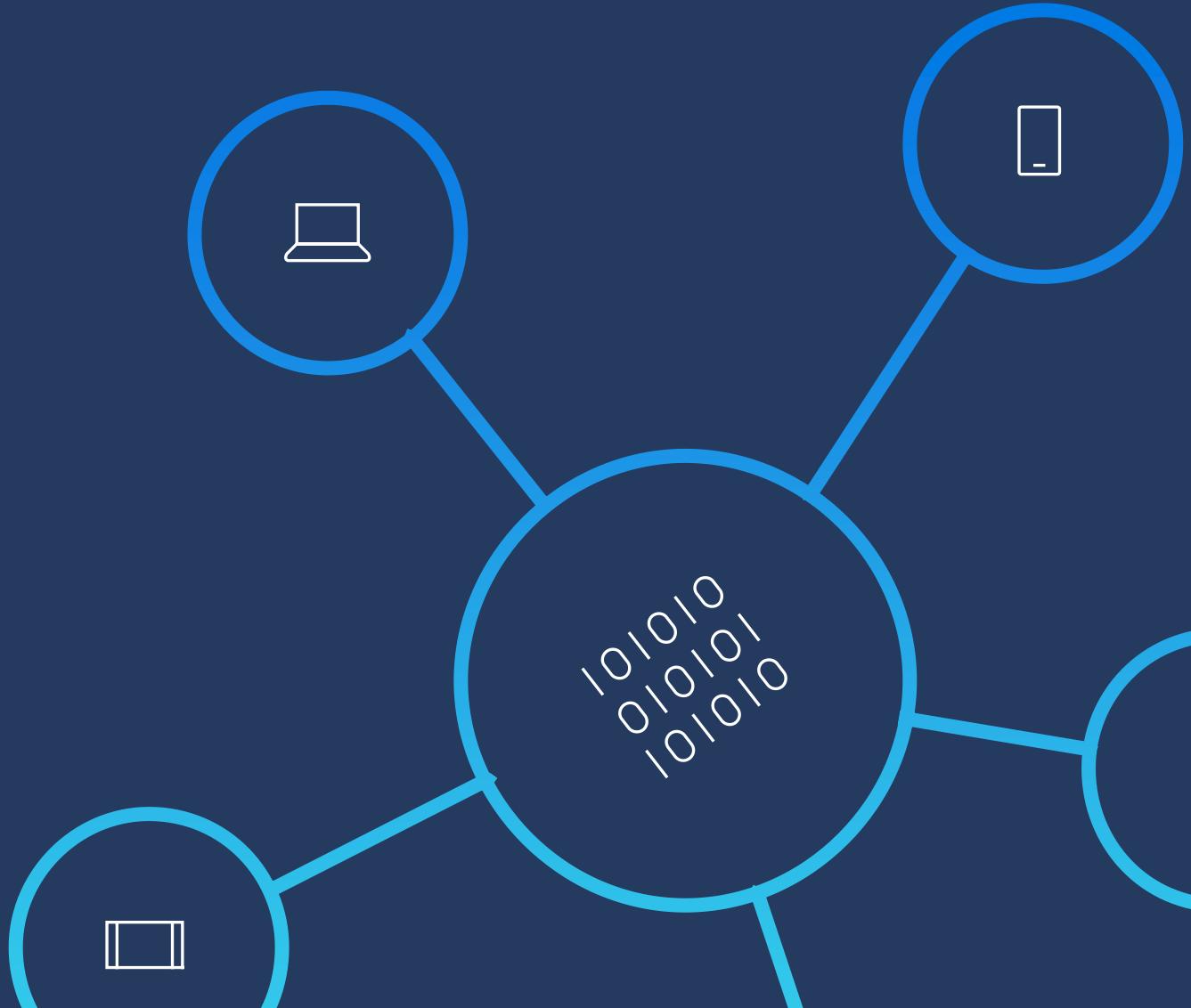
AI-DRIVEN INTELLIGENT SEARCHING



```
"imageDescription": [  
    {"tags": ["person", "road", "outdoor", "sport", "street", "woman", "people", "jumping",  
            "young", "standing", "riding", "city", "playing", "player", "group"], "captions": [{"text": "Shalane Flanagan et al. walking down the street", "confidence": 0.7455881694062344}]}]  
, "imageText": [  
    "B TATA CONSULTANCY SERVICES TATA TCS NEW FLANAGAN 2017 % WOR YORK CITY  
    airbnb"]]
```

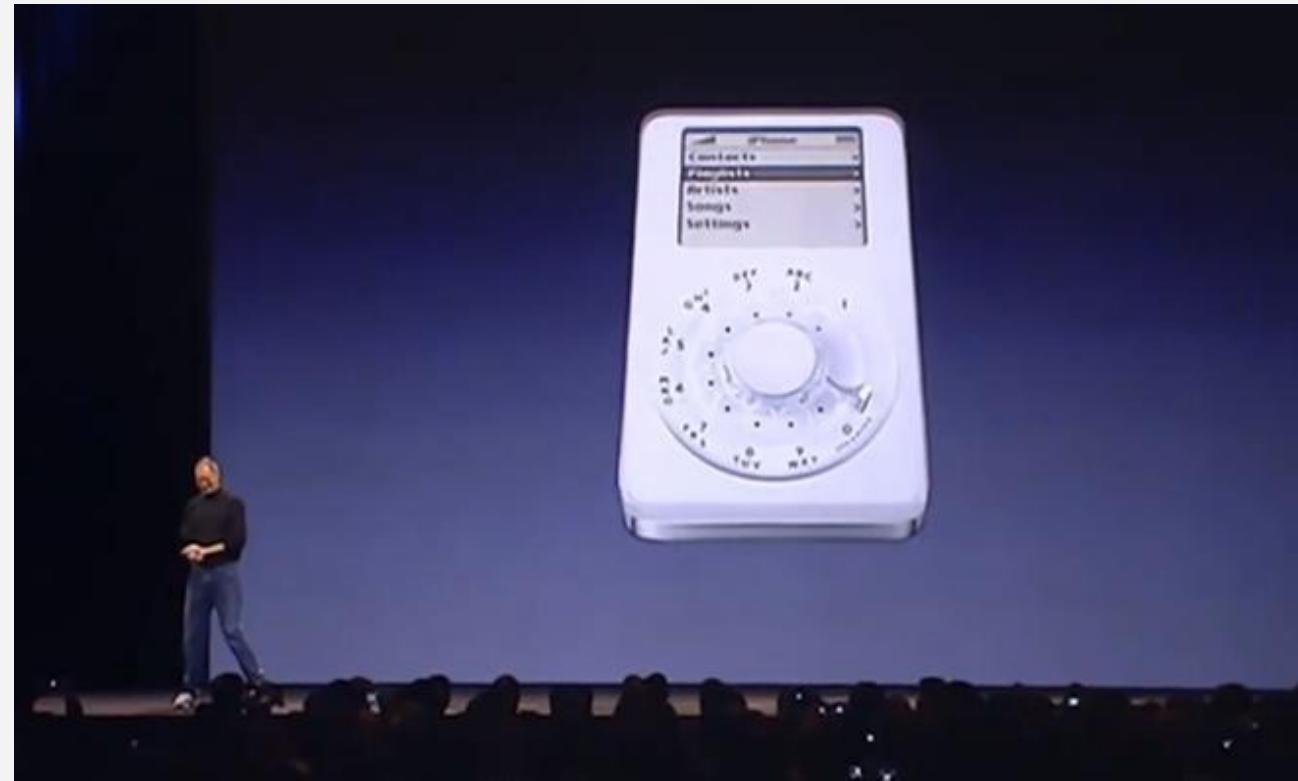


Designing and Developing with CosmosDB



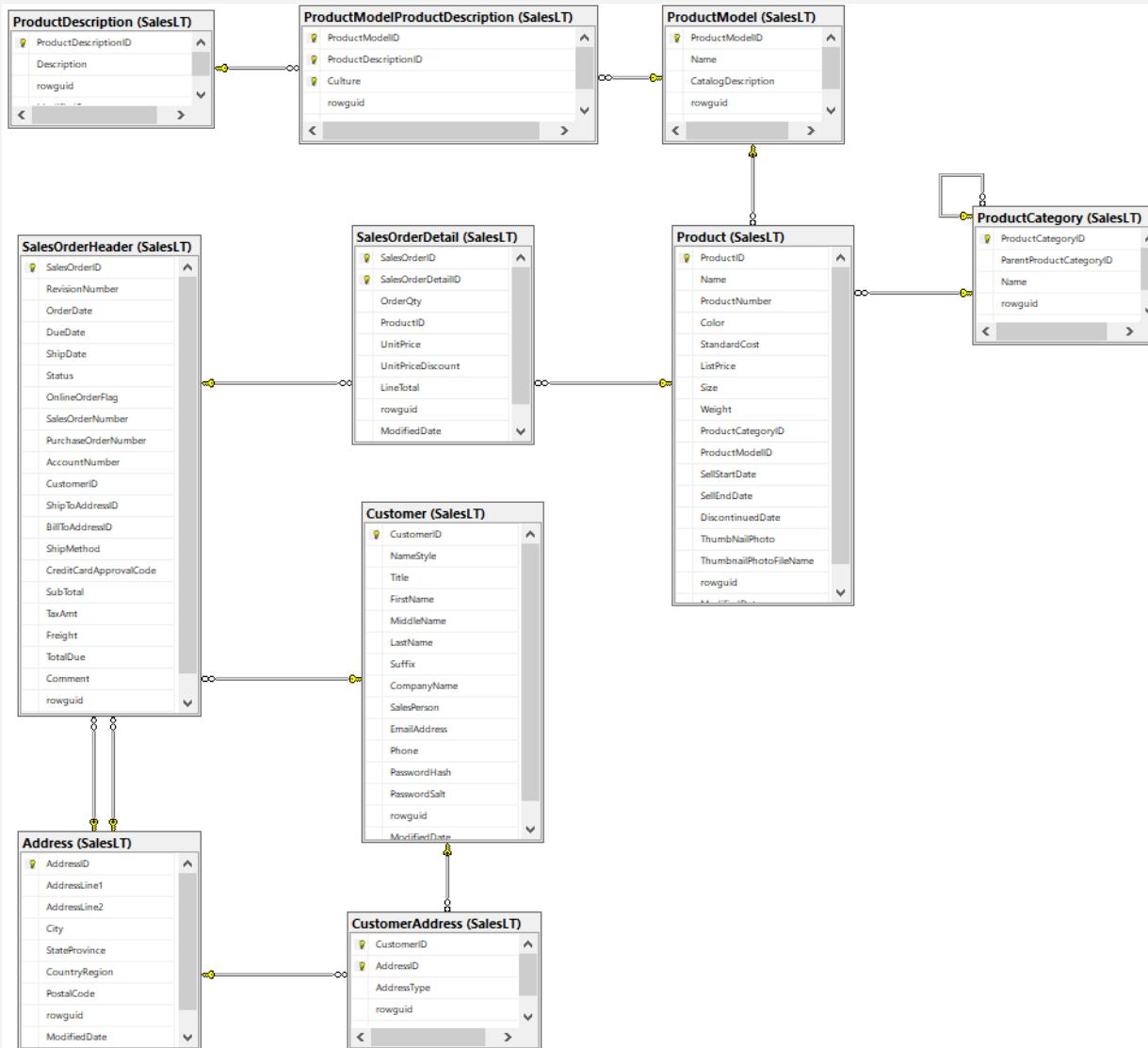
HOW DO I DESIGN THIS NEW THING?

BEWARE OF "DESIGN INERTIA" FROM YOUR PREVIOUS TECHNOLOGY



DESIGN: ANTI-PATTERNS

COSMOSDB IS NOT RELATIONAL - NO CROSS-CONTAINER JOINS OR RI



DESIGN: ANTI-PATTERNS

COSMOSDB IS NOT A DATA LAKE; IT'S OLTP NOT OLAP



!=



ONE COMMON AND SUCCESSFUL DESIGN

SINGLE CONTAINER, SMALL DOCUMENTS, PK JOINS

```
{  
  "pk": "XK1123",  
  "doctype": "order",  
  "orderNumber": "XK1123",  
  "customer": "cjoakim",  
  ... other order attributes ...  
}
```

```
{  
  "pk": "XK1123",  
  "doctype": "lineitem",  
  "orderNumber": "XK1123",  
  "number": 1,  
  "item": {"sku": 123, "cost": 12.44, "quantity": 6, "shipping_method": "usps"}  
}
```

```
{  
  "pk": "XK1123",  
  "doctype": "lineitem",  
  "orderNumber": "XK1123",  
  "number": 2,  
  "item": {"sku": 456, "cost": 349.99, "quantity": 1, "shipping_method": "fedex"}  
}
```

SQL Queries for all or parts of the Order

```
select * from c where c.pk = "XK1123"  
select * from c where c.pk = "XK1123" and cdoctype = "order"  
select * from c where c.pk = "XK1123" and cdoctype in ("order", "lineitem")
```

DESIGN BEST PRACTICES, PAGE 1 OF 3

- Use a **Query-Driven or Operation-driven methodology**, not Structure-Driven
- **CQRS** is helpful to first identify your operations and their relative counts
- **Partition Key design is critical** for perf and costs, especially for large containers
 - High-Cardinality and Well-Distributed are desirable
 - But, also a value that you typically query on (i.e. – customerID)
 - Be aware of 20GB limit per Logical Partition
 - Be aware of the 10K RU performance limit per physical partition
 - Be aware of “Hot Partitions”, at time of ingestion, with same Partition Key
 - Consider “Partition Key Joins” (previous slide)
 - Hierarchical (3 part) Partition Keys are currently in preview mode
- **Prefer Small Documents.** The 2MB max is still huge. Consider update costs.
- Beware of Unbounded Embedded data
 - it can lead to large documents and expensive reads and updates

DESIGN BEST PRACTICES, PAGE 2 OF 3

- **It's schemaless.** There is no need to have many "Entity Containers"
 - Use a "**doctype**" or similar attribute to differentiate the documents
- Iterate your design, and **be aware of your RU costs for each operation**
 - Azure Monitor is invaluable for this RU cost analysis
- Leverage Other PaaS services – Synapse Link, Cognitive Search, etc.
- Leverage the **Change Feed for Event-Driven processing**
- Leverage **TTL** to reduce storage and physical partitions
 - Take advantage of "free deletes"
 - **Your minimum RU level can be driven by the size of your container**
 - Each 50GB Physical Partition ~ = 10,000 RU
- Enable Synapse Link to offload data and batch processing to Synapse

DESIGN BEST PRACTICES, PAGE 3 OF 3

- Consider creating “**Materialized Views**”
 - Optimized data structures for fast reads and serving to web layer
 - Create these with Change Feed and Azure Functions, or Synapse Link
- Optimize performance with **Indexing and Composite Indexes**
- **Let Microsoft Help You** with Design and Architecture
 - We want you to be successful, and happy with Azure and CosmosDB

COSMOSDB PROGRAMMING OVERVIEW

- CosmosDB open-source APIs
 - use the open-source SDKs (Mongo/pymongo, Cassandra/datastax, etc.)
- **CosmosDB SQL API**
 - Microsoft SDKs for the major programming languages
 - Java, C#, Python, Node.js, Go, REST
 - Spring Data
 - <https://docs.microsoft.com/en-us/azure/developer/java/spring-framework/how-to-guides-spring-data-cosmosdb>
- Spark
 - <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/create-sql-api-spark?tabs=python>
- Kafka
 - <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/kafka-connector-sink>
- Emulator – For Windows, or Docker Container
 - <https://docs.microsoft.com/en-us/azure/cosmos-db/local-emulator>
- Azure Data Factory
 - <https://docs.microsoft.com/en-us/azure/data-factory/connector-azure-cosmos-db?tabs=data-factory>

COSMOSDB PROGRAMMING – JAVA SDK V4

- <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/sql-api-sdk-java-v4>
- JDK 8+
- Synchronous Client
- Asynchronous Client - Project Reactor, Mono, Flux
- Bulk Executor
 - <https://docs.microsoft.com/en-us/azure/cosmos-db/sql/bulk-executor-java>
- Excellent Insights into Performance
 - Request Unit Costs, Index Utilization, Server time, etc.
- Interesting Features
 - Preferred Regions List
 - Automatic Configurable Retries
 - Partial Document Updates - container.patchItem
- <https://github.com/cjoakim/azure-cosmosdb-java> My samples, Ask for help!

COSMOSDB PROGRAMMING – JAVA SDK V4

Obtain instances of classes `CosmosAsyncClient`,
`CosmosAsyncDatabase`, `CosmosAsyncContainer`

```
if (client == null) {
    client = new CosmosClientBuilder()
        .endpoint(uri)
        .key(key)
        .preferredRegions(AppConfiguration.getInstance().getPreferredRegions())
        .consistencyLevel(ConsistencyLevel.SESSION)
        .contentResponseOnWriteEnabled(true)
        .buildAsyncClient();
    log.warn("client: " + client);

    database = client.getDatabase(this.currentDbName);
    log.warn("client connected to database Id: " + database.getId());
}
return client;
```

COSMOSDB PROGRAMMING – JAVA SDK V4

- Asynch Query Example
- Obtain a FeedResponse Iterable
 - Flux tolterable()
- Iterate the Pages
- Iterate the Page Items
- Aggregate the RU charges

```
public TelemetryQueryResults getAllTelemetry() {  
  
    String sql = "select * from c offset 0 limit 10000";  
    int    pageSize = 100;  
    String continuationToken = null;  
    CosmosQueryRequestOptions queryOptions = new CosmosQueryRequestOptions();  
    TelemetryQueryResults resultsStruct = new TelemetryQueryResults(sql);  
    resultsStruct.start();  
  
    // Execute the SQL query and iterate the paginated result set,  
    // collecting the documents and total RU charge.  
    do {  
        Iterable<FeedResponse<TelemetryEvent>> feedResponseIterator =  
            container.queryItems(sql, queryOptions, TelemetryEvent.class).byPage(  
                continuationToken, pageSize).toIterable(); // Asynch Flux to Iterable  
  
        for (FeedResponse<TelemetryEvent> page : feedResponseIterator) {  
            for (TelemetryEvent doc : page.getResults()) {  
                resultsStruct.addDocument(doc);  
            }  
            resultsStruct.addRequestCharge(page.getRequestCharge());  
            resultsStruct.incrementPageCount();  
            continuationToken = page.getContinuationToken();  
        }  
    }  
    while (continuationToken != null);  
    resultsStruct.stop();  
    return resultsStruct;  
}
```

JAVA: SPRING AND SPRING DATA

A SAMPLE REPOSITORY CLASS, EXTENDS COSMOSREPOSITORY

```
16 @Component
17 @Repository
18 public interface TelemetryRepository extends
19     CosmosRepository<TelemetryEvent, String>,
20     TelemetryRepositoryExtensions {
21
22     1 usage
23     Iterable<TelemetryEvent> findByStateCode(String stateCode);
24
25     1 usage
26     Iterable<TelemetryEvent> findByStateCodeAndCountyCode(String stateCode, String countyCode);
27
28     3 usages
29     @Query("select value count(1) from c")
30     long countAllDocuments();
31
32     1 usage
33     @Query( "select c.id, c.pk, c.stateCode, c.countyCode, c.siteNum, c.observationCount, c.nullObservations " +
34         "from c " +
35         "where c.observationCount >= @observationCount " +
36         "and c.nullObservations >= @nullObservations")
37     Iterable<TelemetryEvent> findByObservationCount(
38         @Param("observationCount") long observationCount,
39         @Param("nullObservations") long nullObservations);
40
41 }
```

JAVA: SPRING AND SPRING DATA

A SAMPLE DOMAIN OR ENTITY POJO CLASS, WITH ANNOTATIONS

```
21  @Data
22  @NoArgsConstructor
23  @JsonIgnoreProperties(ignoreUnknown=true)
24  @JsonInclude(JsonInclude.Include.NON_NULL)
25  @Container(containerName=AppConstants.TELEMETRY_CONTAINER_NAME)
26  public class TelemetryEvent implements AppConstants {
27
28      @Id
29      @GeneratedValue
30      private String id; // CosmosDB document ID
31
32      @PartitionKey
33      private String pk; // CosmosDB partition key
34
35      private String stateCode;
36      private String countyCode;
37      private String siteNum;
38      private double latitude;
39      private double longitude;|
40      private String datum;
41      private String metric;
42      private String localDateTime;
43      private String gmtDateTime;
44      private String uom;
45      private int observationCount;
46      private int nullObservations;
47      private double meanObservation;
48
49      private String _etag; // CosmosDB document version hash for OCC
50      private long _ts; // CosmosDB document timestamp
51
```

JAVA: SPRING AND SPRING DATA

A REPOSITORY EXTENSION INTERFACE TO EXTEND THE SPRING DATA API

```
7  /**
8   * This interface was created to extend the TelemetryRepository, which in turn extends
9   * CosmosRepository<TelemetryEvent, String> from the CosmosDB Spring Data SDK.
10  *
11  * This demonstrates how to leverage more of the power of the CosmosDB SQL syntax, by using
12  * "Criteria" objects and an Autowired "CosmosTemplate" object.
13  *
14  * See class TelemetryRepositoryExtensionsImpl in this package, which implements this interface.
15  *
16  * Chris Joakim, Microsoft, September 2022
17  */
18 public interface TelemetryRepositoryExtensions {
19
20     public Iterable<TelemetryEvent> findByStateCodeAndSiteNumbers(String stateCode, ArrayList<String> siteNumbers);
21 }
```

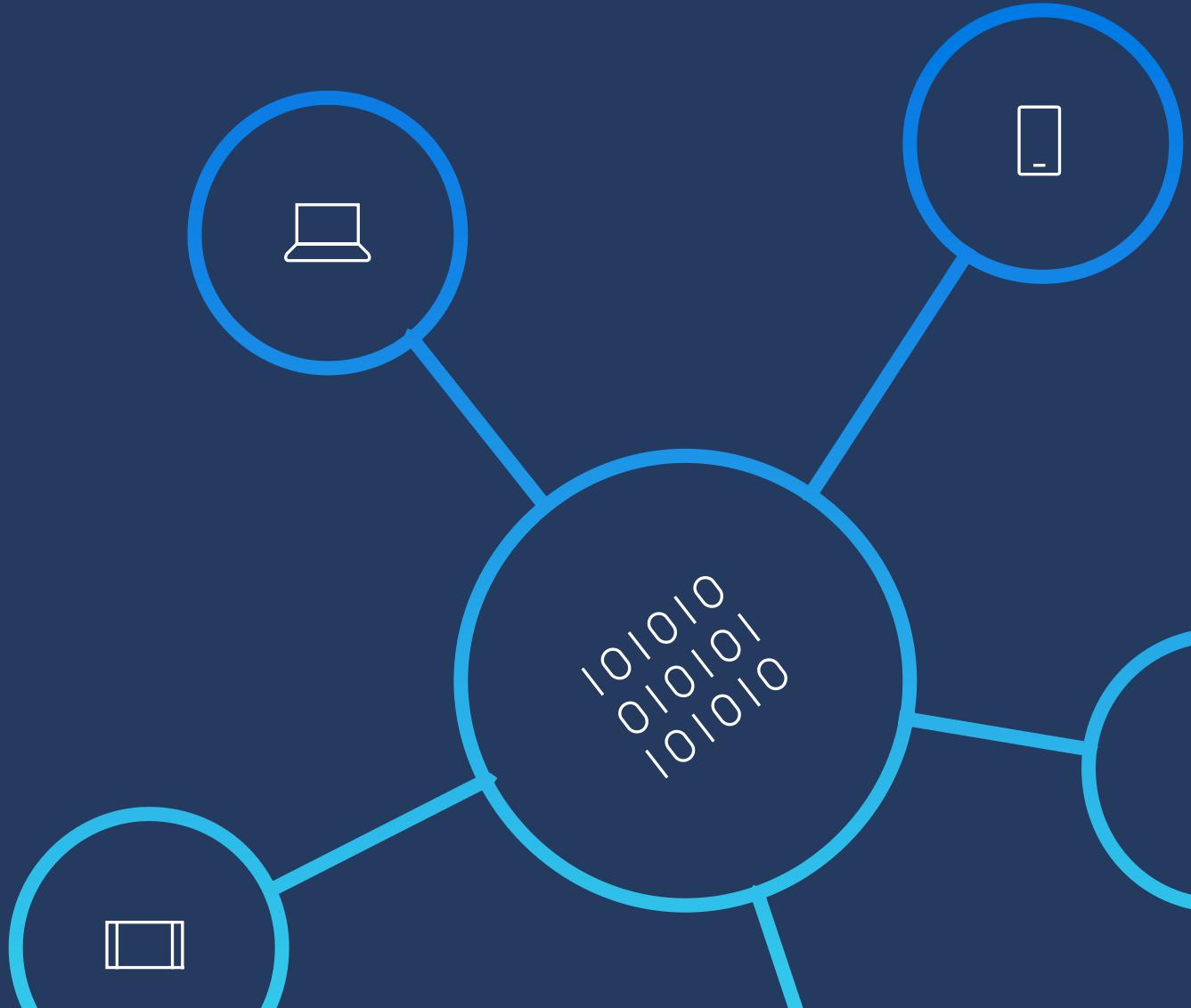
JAVA: SPRING AND SPRING DATA

A REPOSITORY EXTENSION INTERFACE IMPLEMENTATION

- **@Autowired CosmosTemplate**
- **Criteria Objects**
- Criteria.IS_EQUAL
- Criteria.IN
- Criteria.AND
- CosmosQuery
- template.find()

```
24 @Slf4j
25 public class TelemetryRepositoryExtensionsImpl implements TelemetryRepositoryExtensions {
26     3 usages
27     private CosmosTemplate template;
28
29     @Autowired
30     public TelemetryRepositoryExtensionsImpl(CosmosTemplate t) {
31         super();
32         this.template = t;
33         log.warn("TelemetryRepositoryExtensionsImpl constructor, template: " + this.template);
34     }
35     1 usage
36     public Iterable<TelemetryEvent> findByStateCodeAndSiteNumbers(String stateCode, ArrayList<String> siteNumbers) {
37
38         String containerName = AppConstants.TELEMETRY_CONTAINER_NAME;
39
40         Criteria criteria1 = Criteria.getInstance(
41             CriteriaType.IS_EQUAL, subject: "stateCode", Collections.singletonList(stateCode),
42             Part.IgnoreCaseType.NEVER);
43         Criteria criteria2 = Criteria.getInstance(
44             CriteriaType.IN, subject: "siteNum", Collections.singletonList(siteNumbers),
45             Part.IgnoreCaseType.NEVER);
46         Criteria allCriteria = Criteria.getInstance(CriteriaType.AND, criteria1, criteria2);
47
48         CosmosQuery query = new CosmosQuery(allCriteria);
49         return template.find(query, TelemetryEvent.class, containerName);
50     }
51 }
```

Questions & Discussion





Thank You!