Microsoft

Azure Cosmos DB

Technical Overview

Chris Joakim, Specialist / Global Black Belt, Charlotte
Hailey Wu, Cloud Soluution Architect, Toronto
Xavier Guérette, Cloud Soluution Architect, Montreal

AGENDA

Overview

- CosmosDB: What it is, and why use it?
- Resource Model, Provisioning, Automation

The Basics

- System Topology & Partitioning
- Request Units (RU) and Scaling
- Global Distribution & Consistency Models
- HA/DR, Backups
- Logging / Azure Monitor / Kusto

Main Features

- Change Feed
- TTL

Main Features (continued)

- HTAP with Synapse Link
- Integrated Cache
- Integration Examples:
  - Streaming – Stream Analytics, Functions
  - Search with Azure Cognitive Search

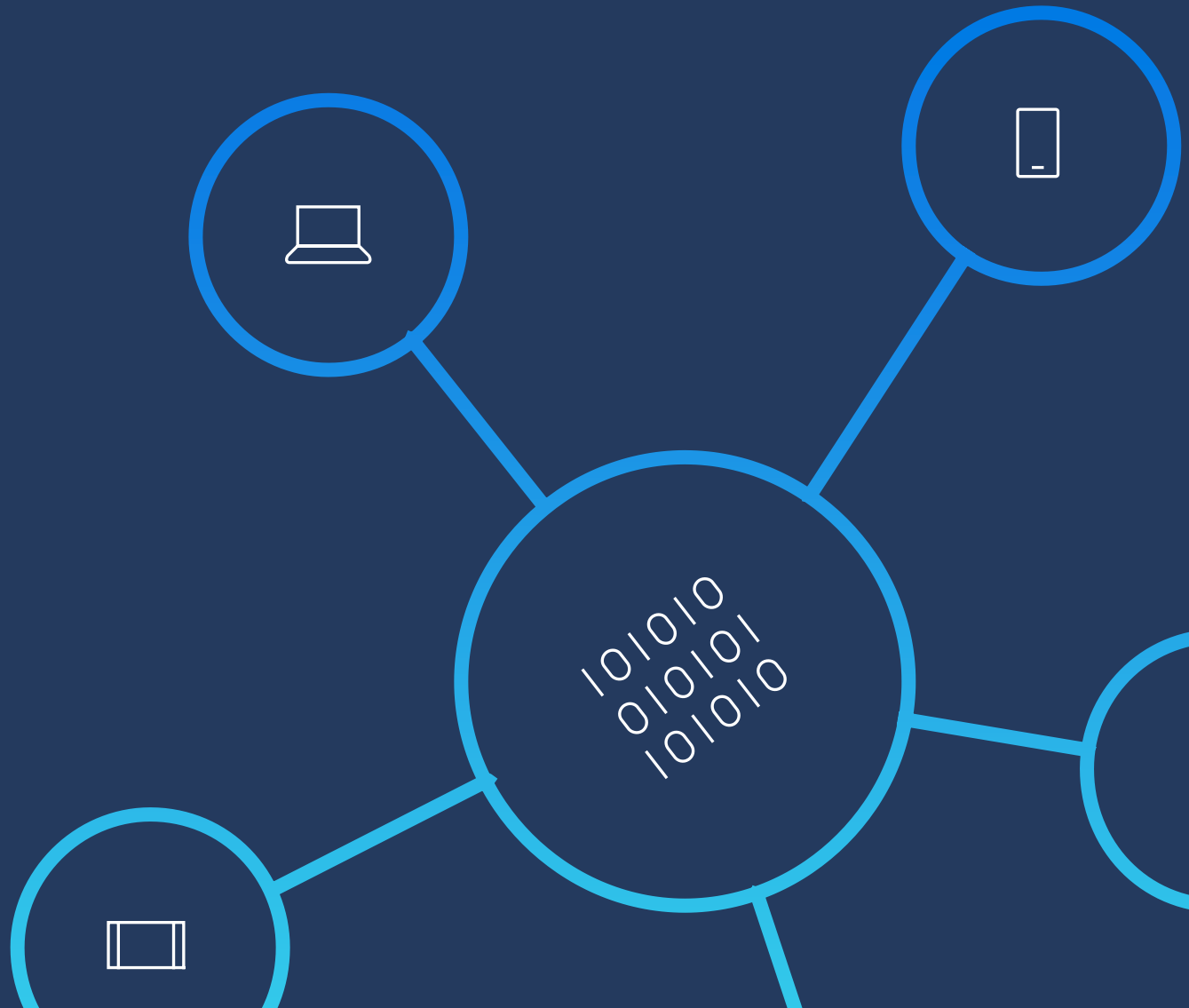
Questions, Discussion, Next Steps

Design and Development (Next Session)

- Design – patterns and anti-patterns
- Java – Spring, Spring Data, SDK
- Links to Documentation and Examples

Microsoft Azure DATA & AI

OVERVIEW

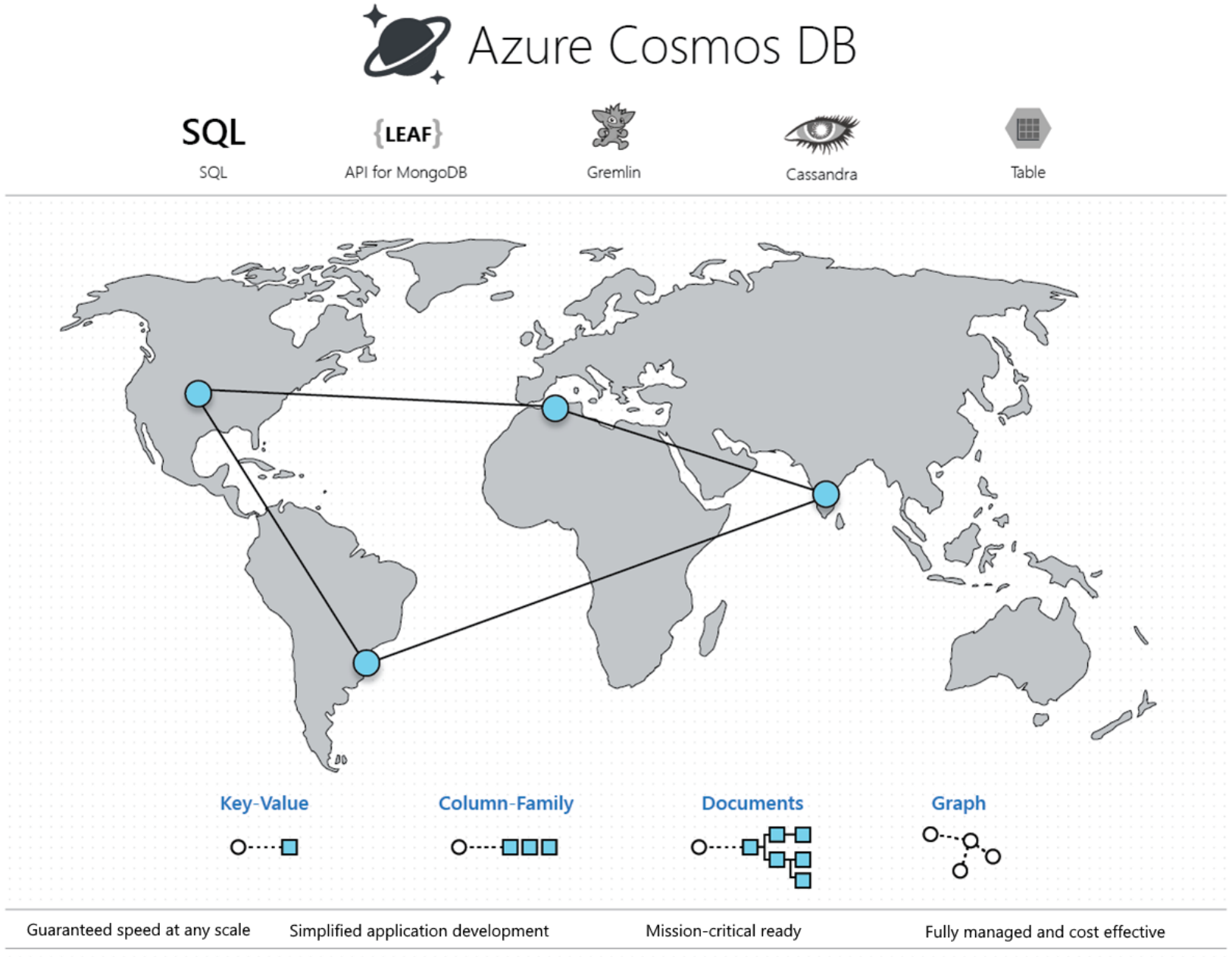


Cosmos DB

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

APIs

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



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

Guranteed 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**.

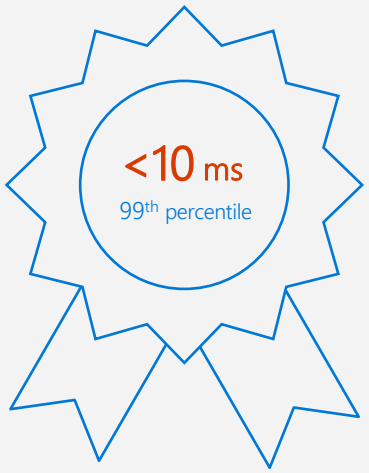
Ease of use.

COMPREHENSIVE SLAs
COVERS AVAILABILITY AND LATENCY

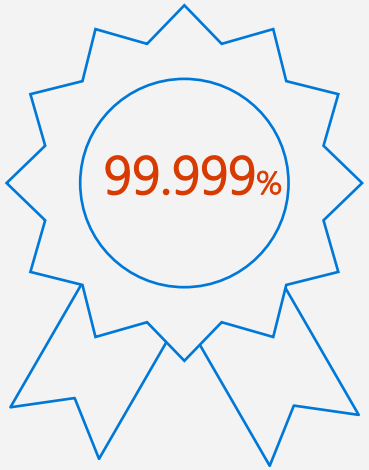
RUN YOUR APP ON WORLD-CLASS INFRASTRUCTURE

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

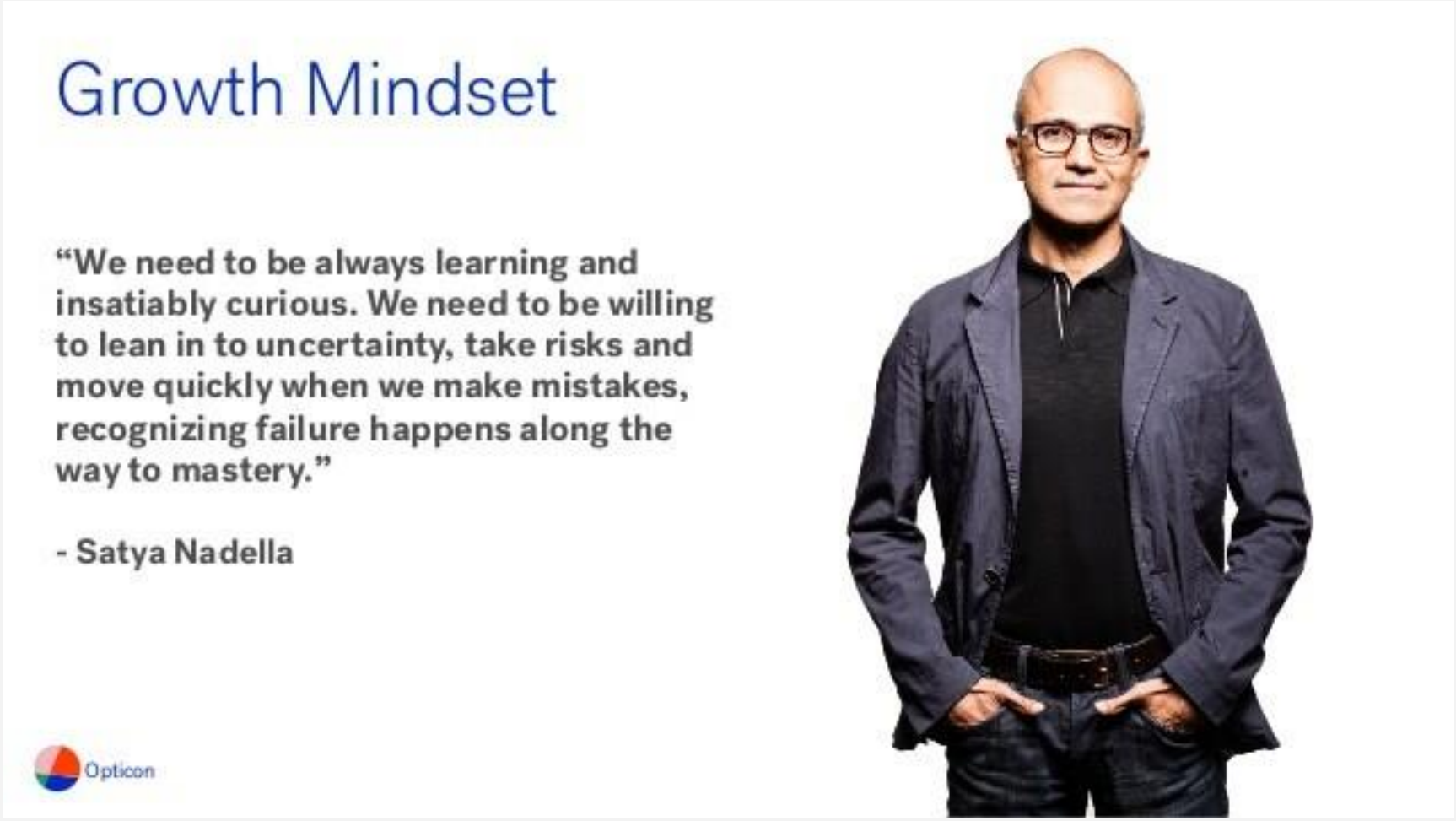
GROWTH MINDSET; ALWAYS BE LEARNING
PLEASE EMBRACE LEARNING COSMOSDB TO LEVERAGE IT BEST

Similar, but different:

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

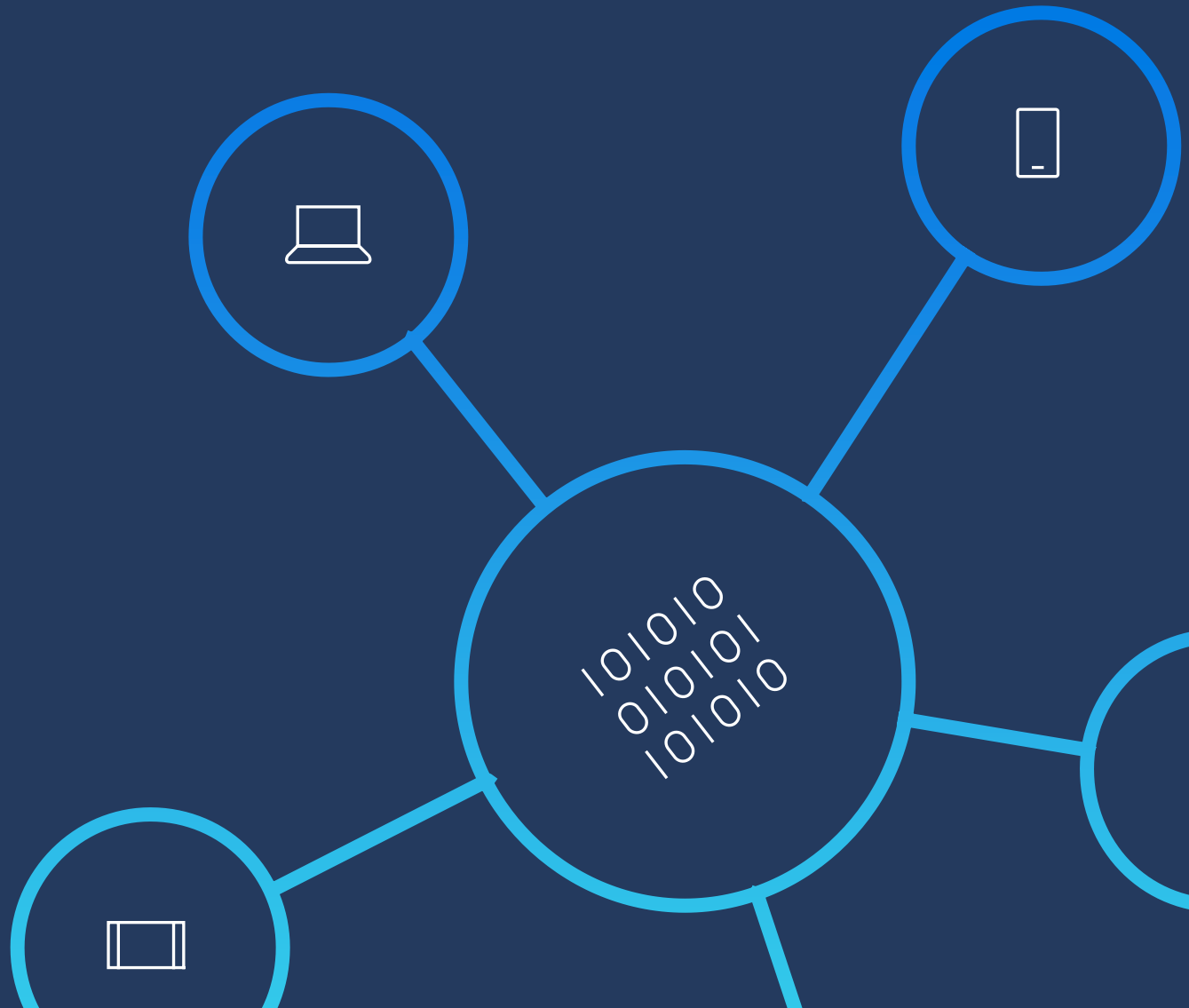
Each programming language and DB has its' own Idioms/Patterns.

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



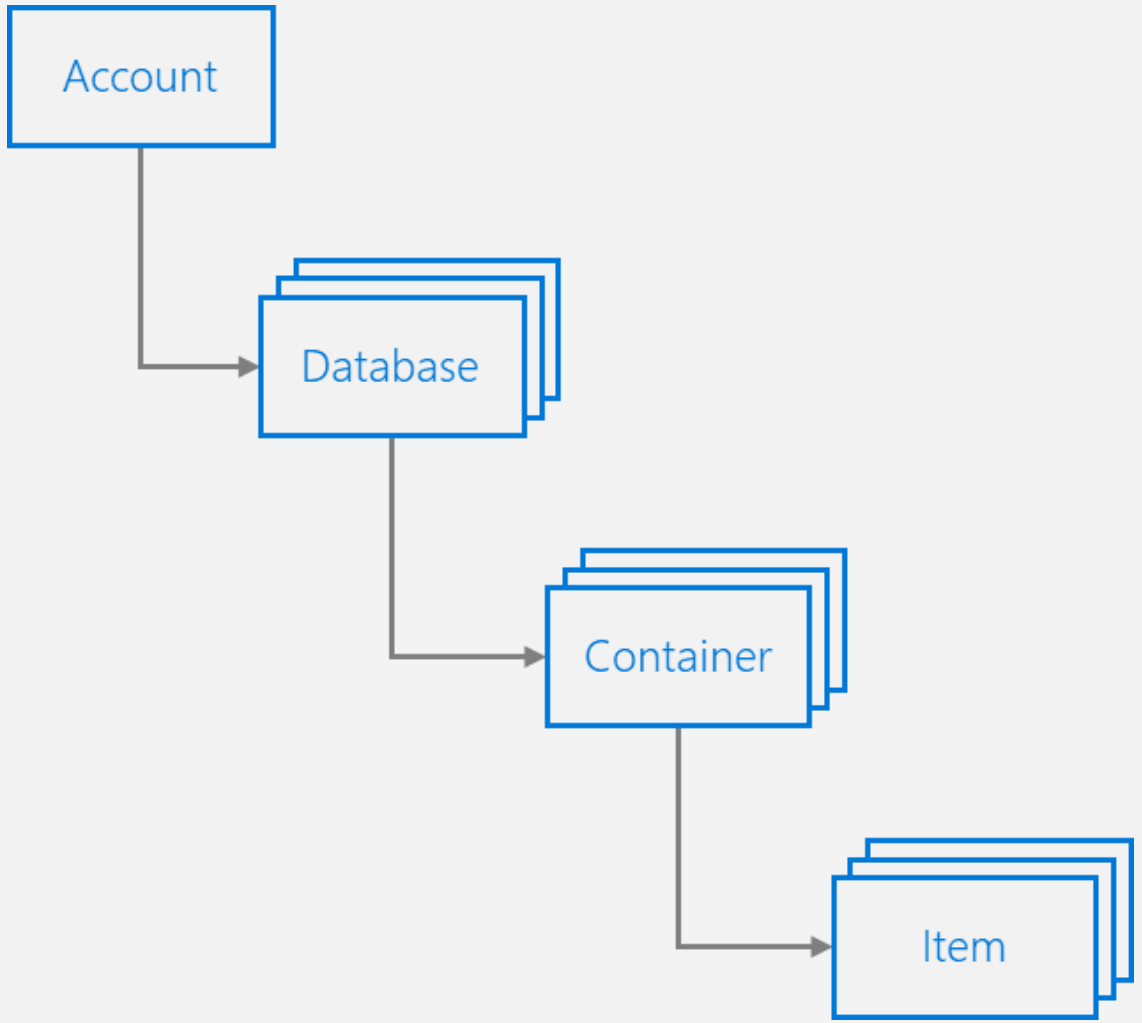
Microsoft Azure DATA & AI

The Basics



THE STRUCTURE OF A COSMOSDB ACCOUNT

"CONTAINER" == "COLLECTION"



IS THE COSMOSDB SQL API RELATIONAL? NO!

- The data is JSON
- It's Schemaless
  - other than the pk
- Nesting
- Query it with SQL
- Indexing – default, custom, paths, composite


```
[
    {
        "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
    }
]
```

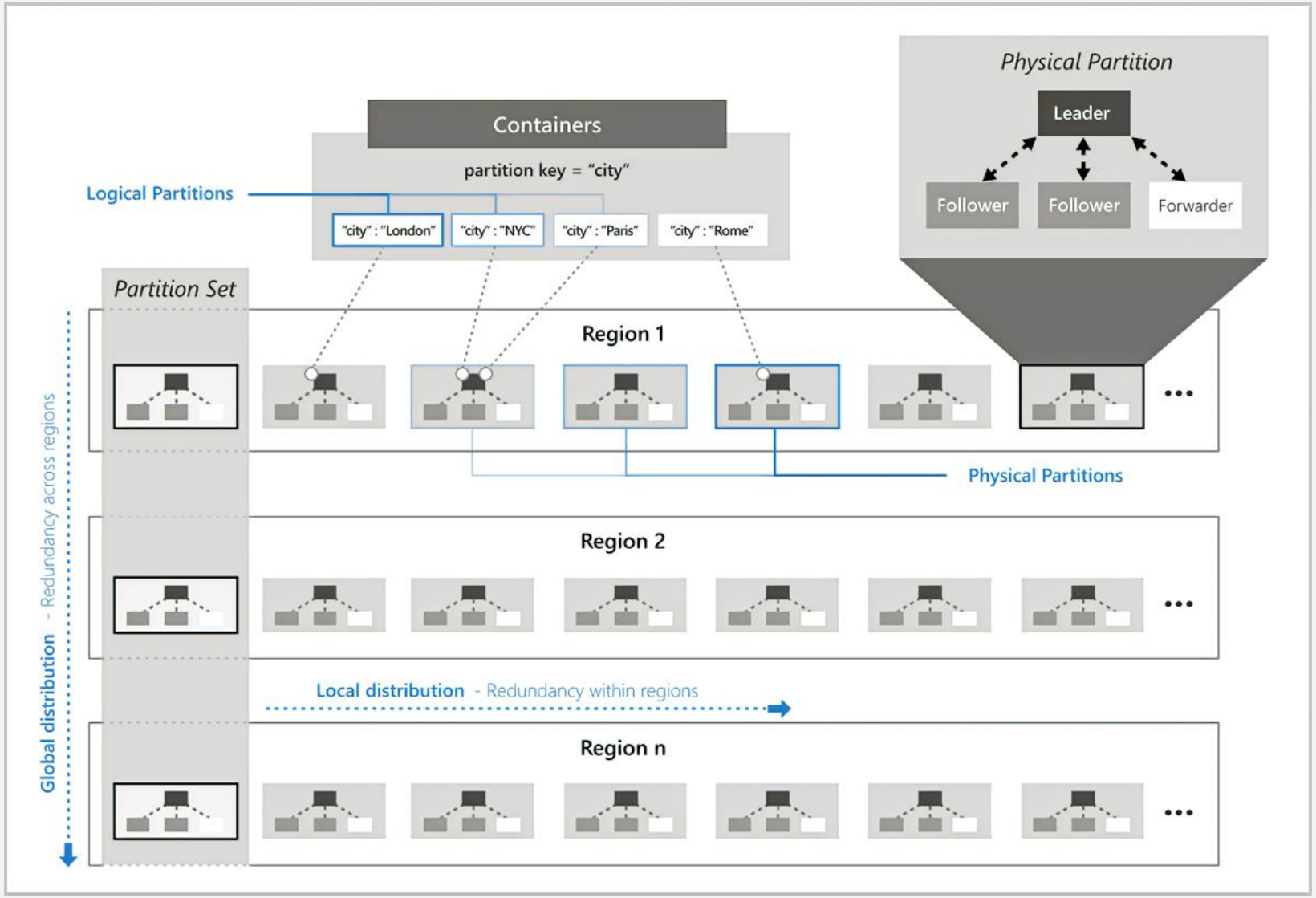

```
SELECT * FROM c where c.id = 'f38bd845-83da-4b31-8c1e-3668eb9ac26a' and c.pk = '30.497478|-87.880258'
```

REQUEST UNITS – THE UNIT OF SCALE AND PRICE

- Request Units (RU) are a scalable unit of 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
- Think of an Uber ride vs buying a car; you pay per use
- RU 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)

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

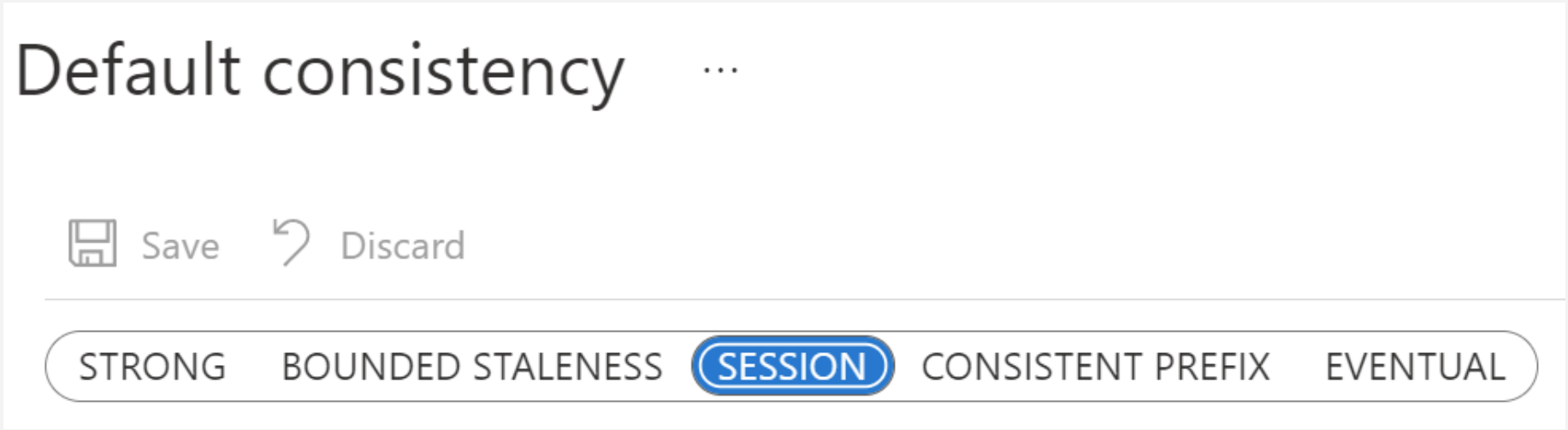
Disable | Enable

Configure the regions for reads, writes and availability zone

+ Add region

Write region
East US
Read regions
North Europe

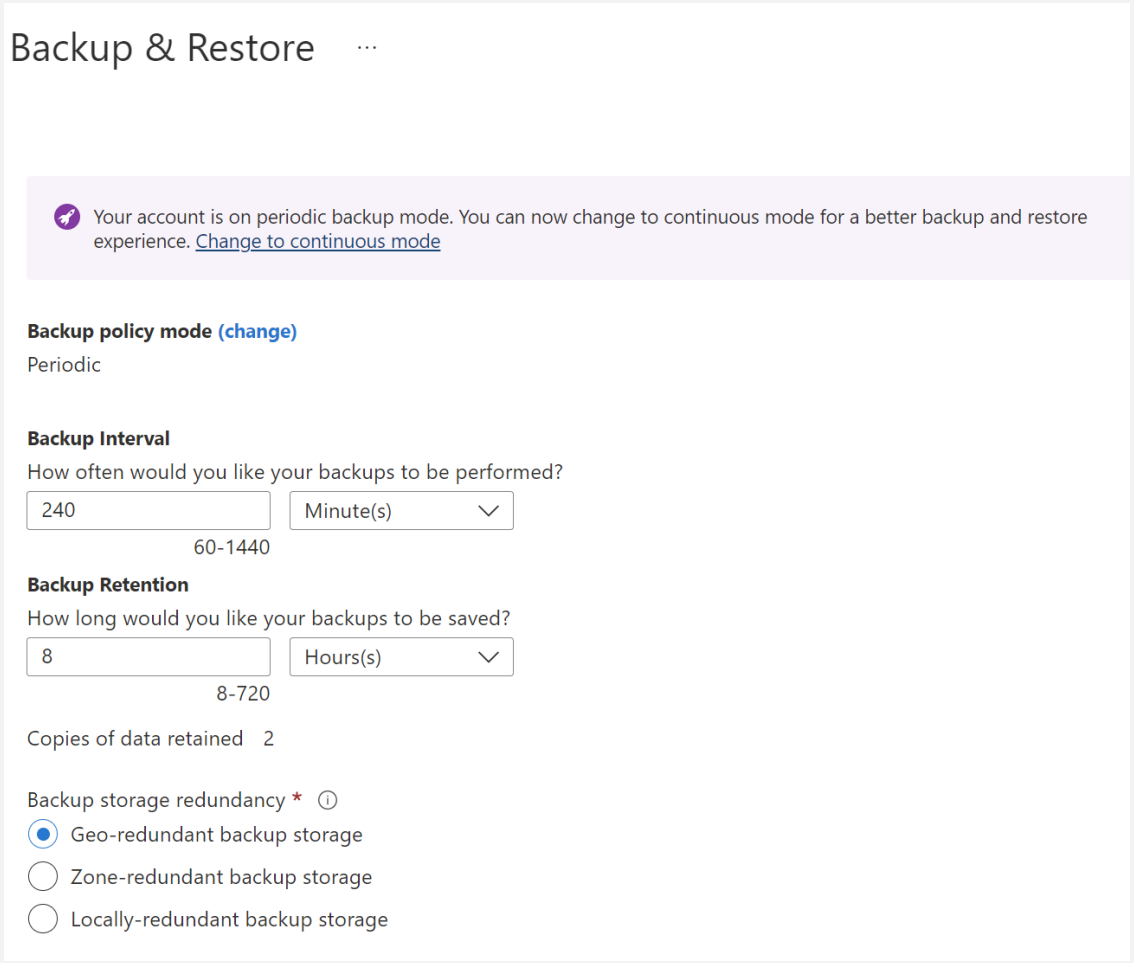
FIVE CONSISTENCY LEVELS FOR MULTIPLE WRITE REGIONS



https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels

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)



https://docs.microsoft.com/en-us/azure/cosmos-db/online-backup-and-restore

LOGGING, MONITORING, ALERTS
SLIDE 1 – EXAMPLE CLASSIC (ORIGINAL) INHERENT METRICS

Metrics (Classic) ...

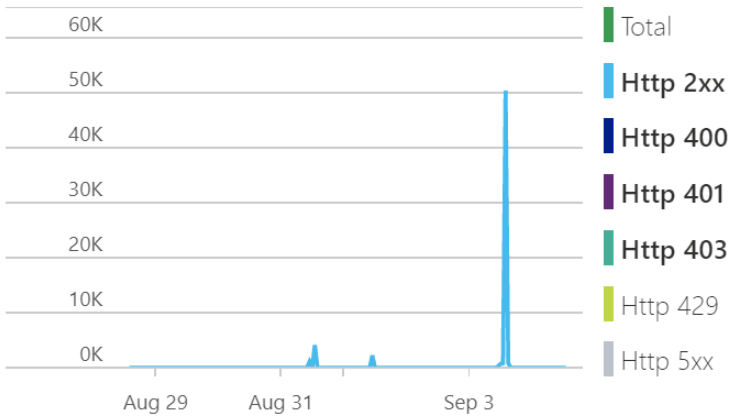
Download as csv Refresh Feedback

Overview | **Throughput** | Storage | Availability | Latency | Consistency | System

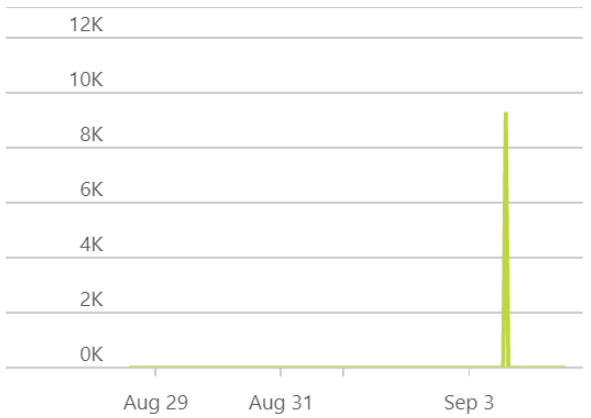
Database(s) dev Container(s) telemetry Region(s) All Show data for last 7 days

Collection telemetry has provisioned throughput at the database level.

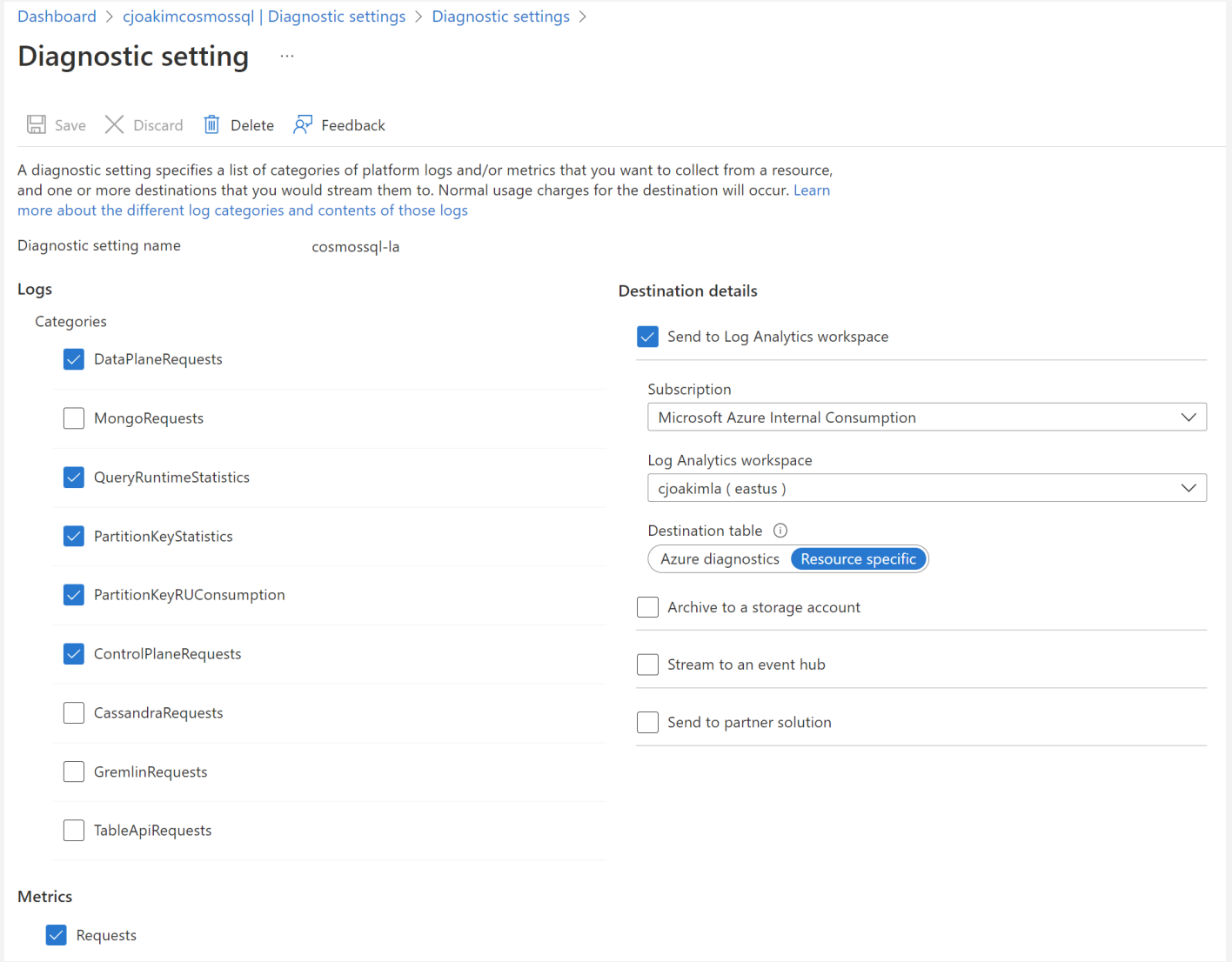
Number of requests (aggregated over 1 hour interval)



Number of requests exceeded capacity (aggregated over 1 hour interval)



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



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?

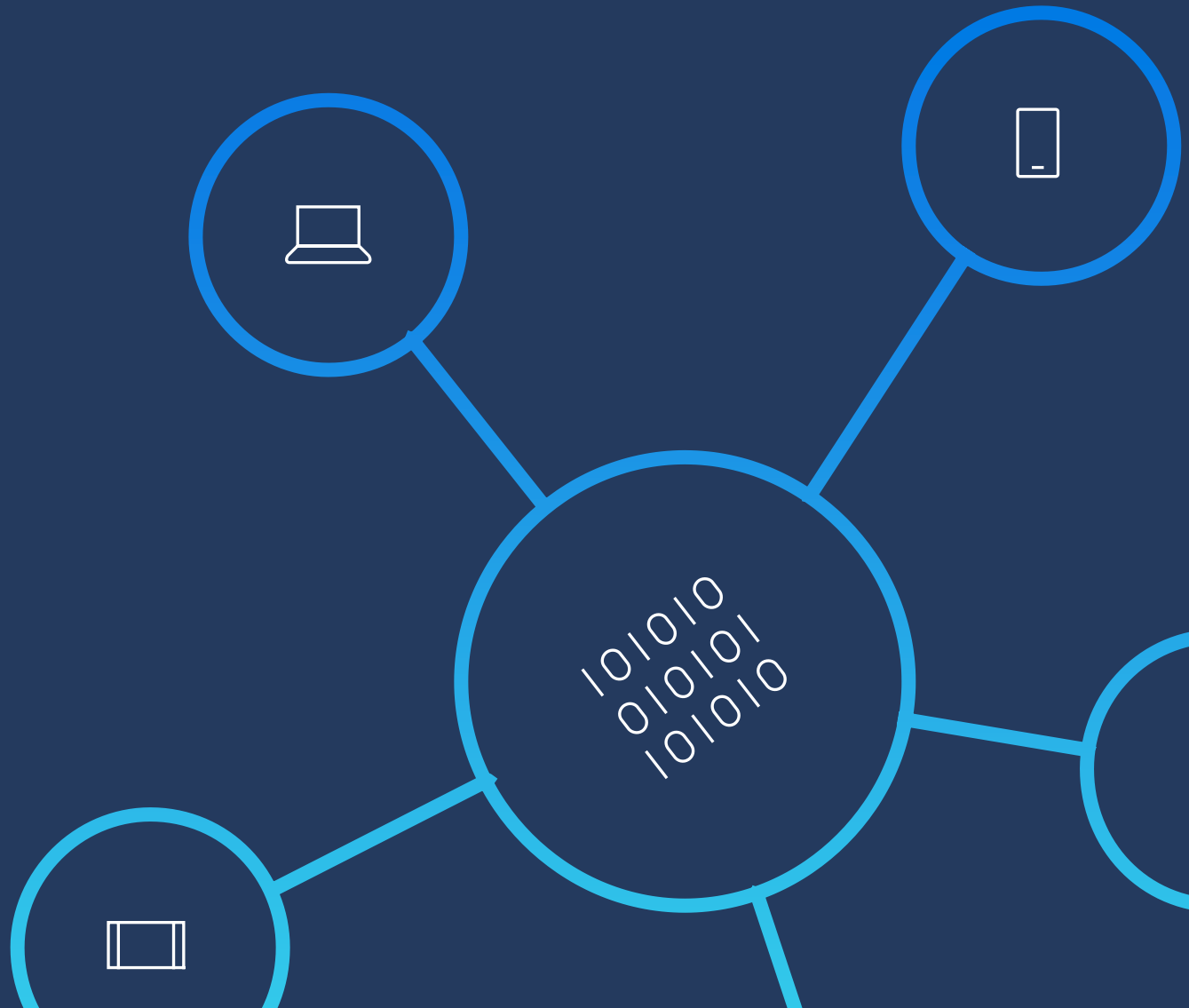
Kusto Copy

```
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

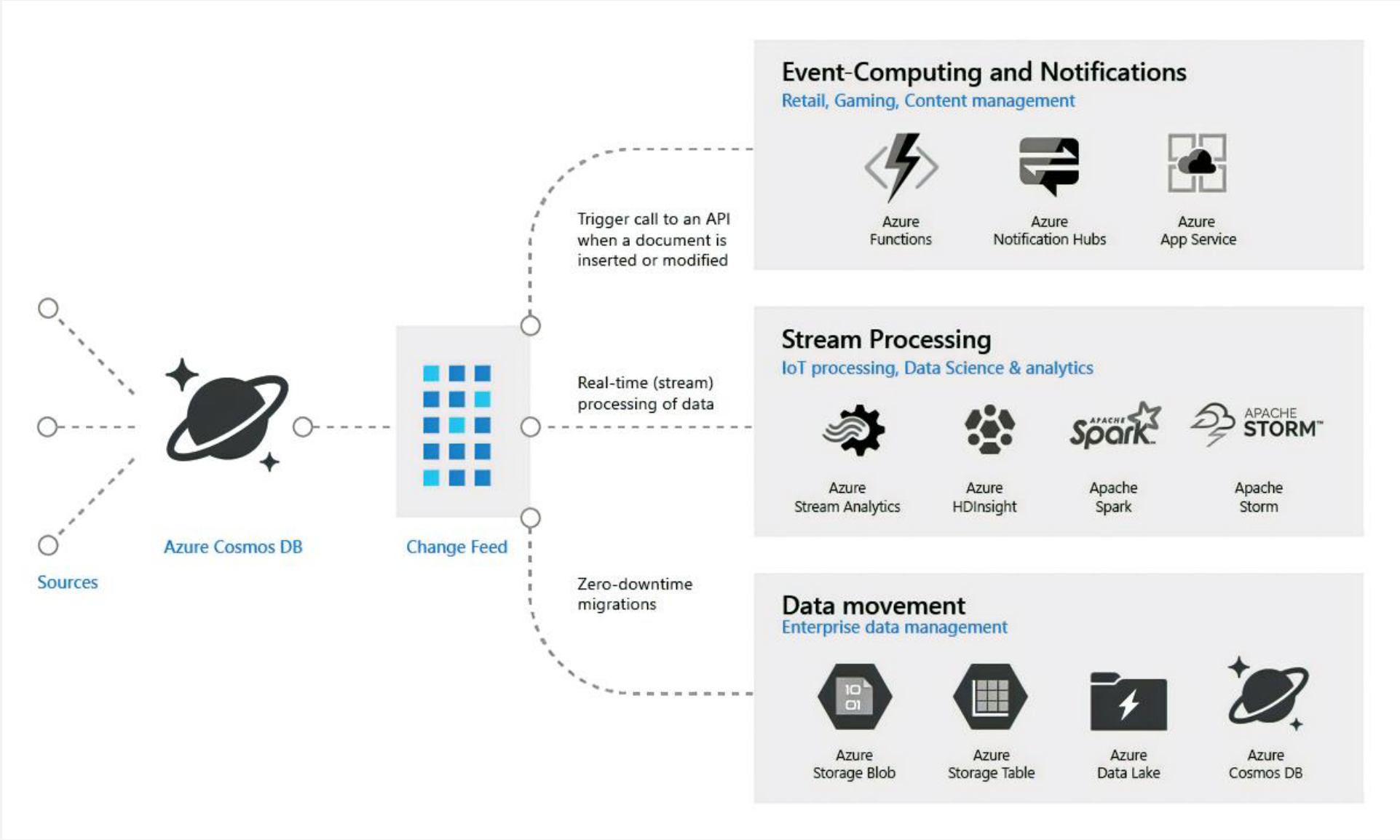
Microsoft Azure DATA & AI

Main Features



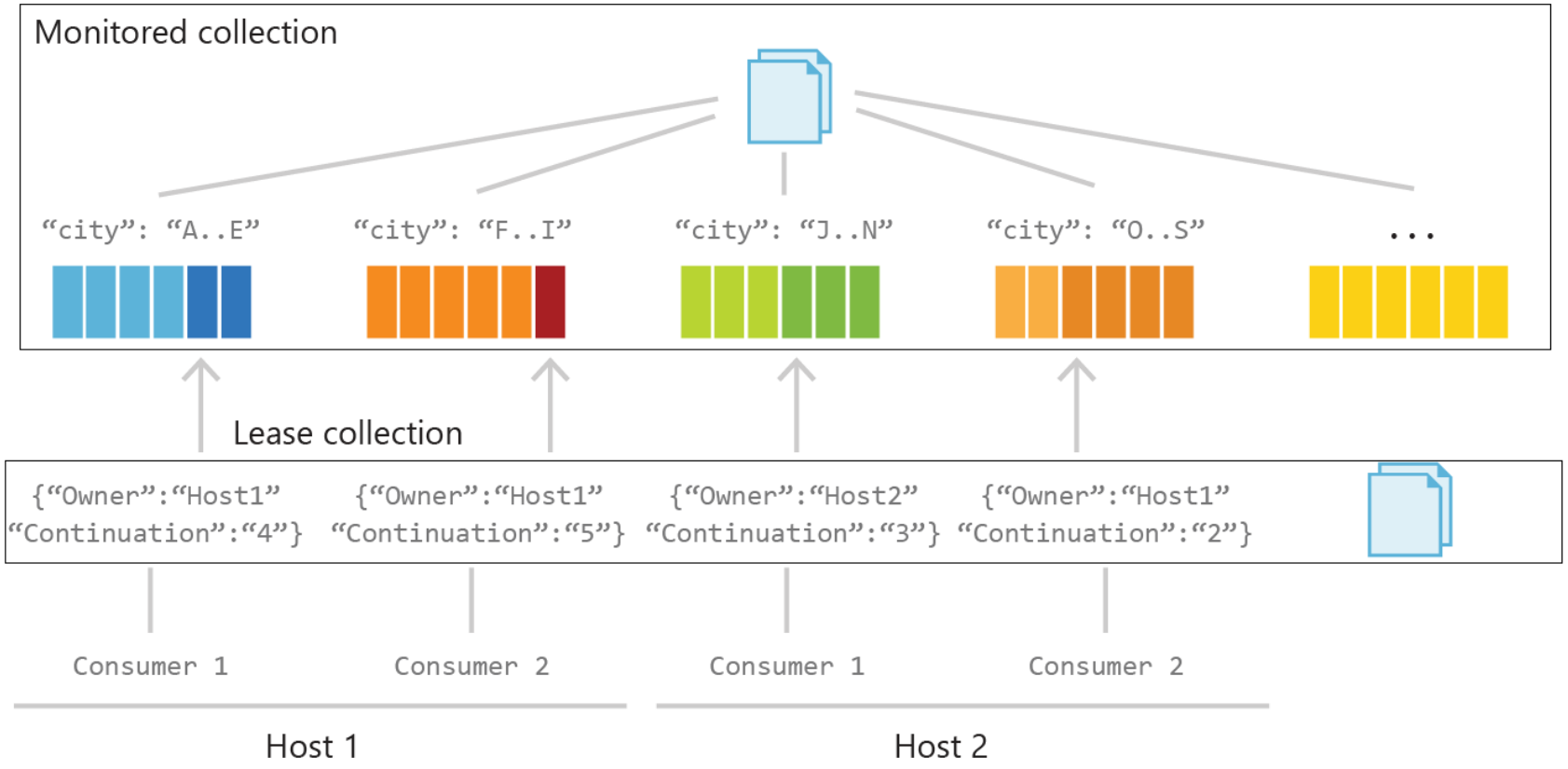
COSMOSDB CHANGE FEED

CONSUME WITH AZURE FUNCTIONS OR A SDK (JAVA OR C#)



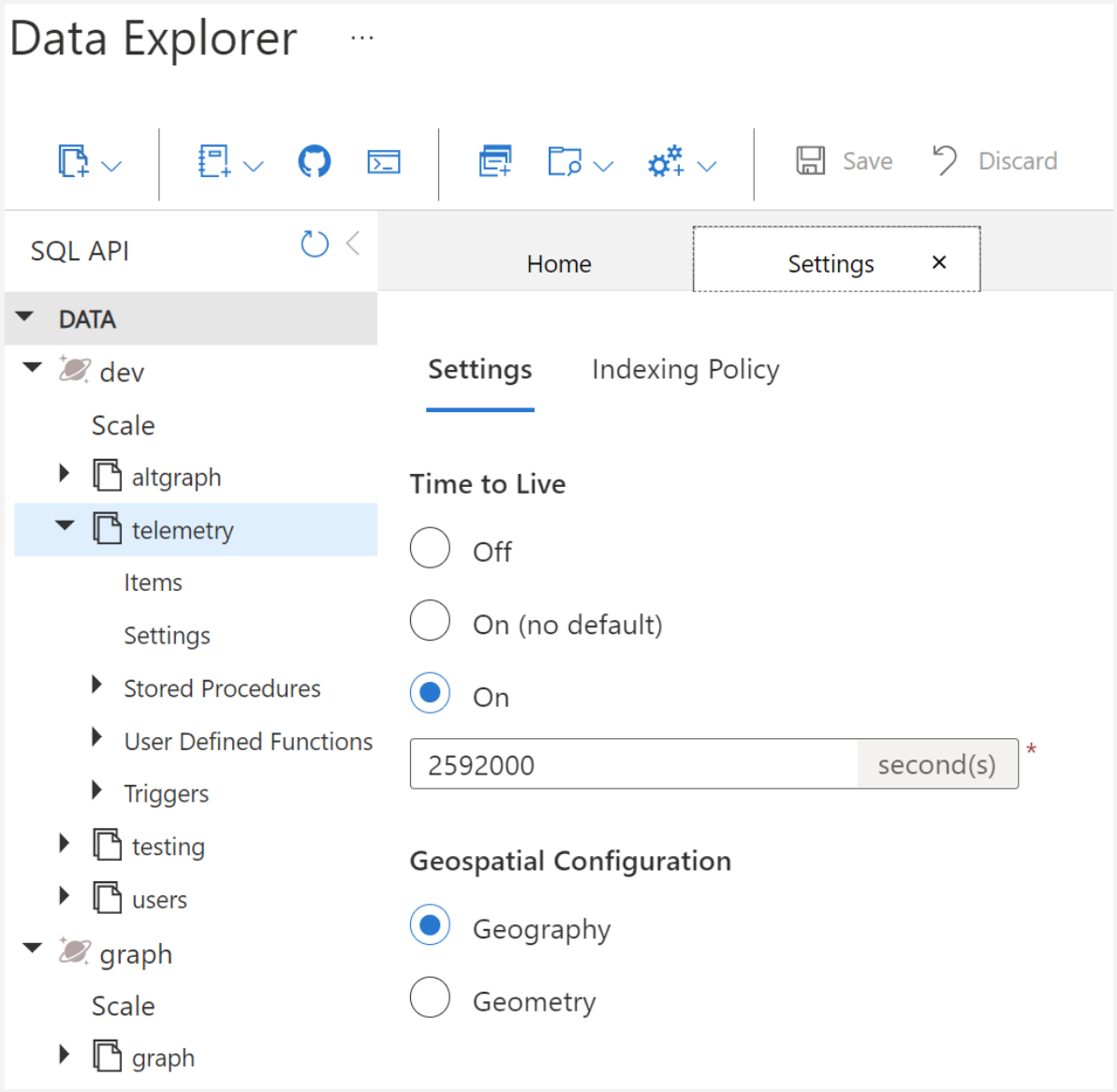
CHANGE FEED PROCESSOR
PARTITIONS, CONSUMERS, LEASES

A "Leases" container maintains the state of a change-feed consumer



TIME-TO-LIVE (TTL)

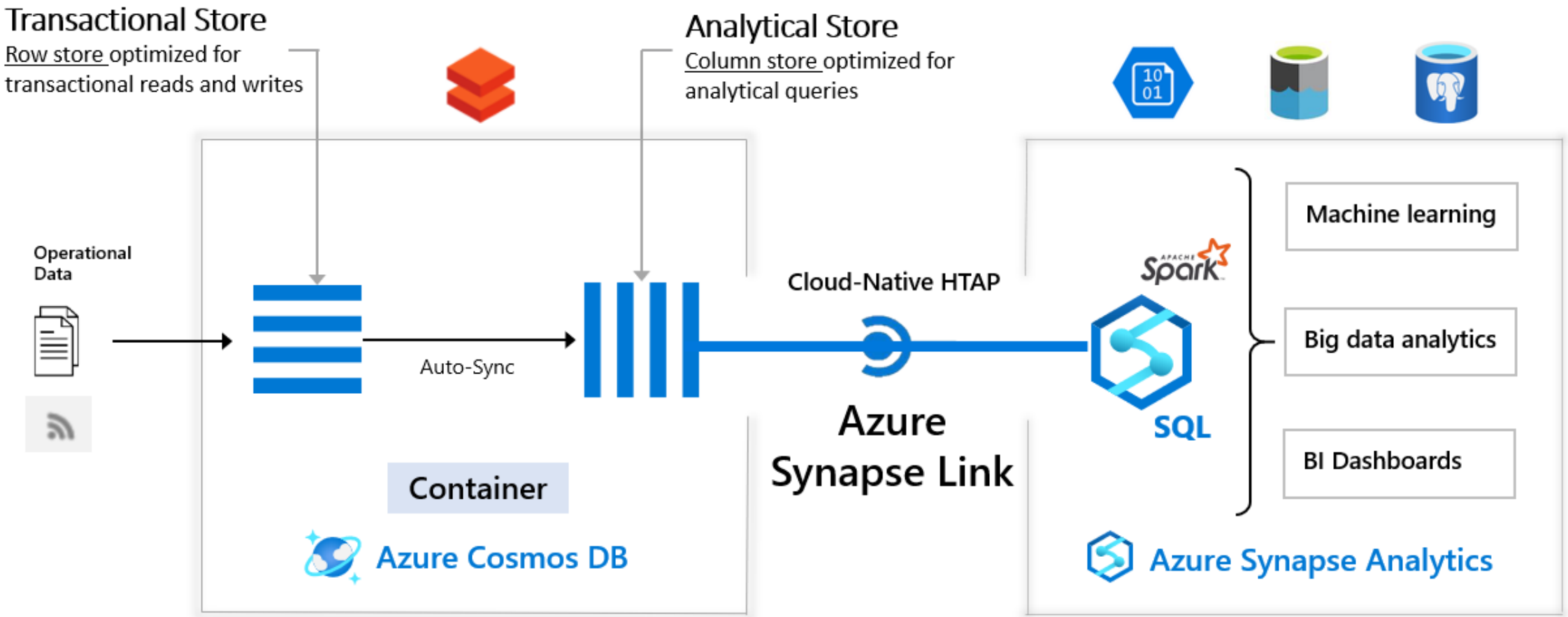
AUTOMATIC DELETION WITH ZERO RU COSTS, BASED ON _TS




```
{
    "id": "726a44ab-856f-4027-b429-312c30250a79",
    "pk": "33.904039|-86.053867",
    "stateCode": "01",
    "countyCode": "055",
    "siteNum": "0011",
    "latitude": 33.904039,
    "longitude": -86.053867,
    "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": 0.029,
    "_rid": "gklzAKAAj3tvNwAAAAAAAA==",
    "_self": "dbs/gklzAA==/colls/gklzAKAAj3s=/docs/gklzAKAAj3tvNwAAAAAAAA==/",
    "_etag": "\"9103d226-0000-0100-0000-6313a0960000\"",
    "_attachments": "attachments/",
    "_ts": 1662230678
}
```

HTAP WITH SYNAPSE LINK

HYBRID TRANSACTIONAL ANALYTICAL PROCESSING (OLTP & OLAP)



HTAP WITH SYNAPSE LINK

READING THE ANALYTIC DATASTORE WITH SPARK IN AZURE SYNAPSE

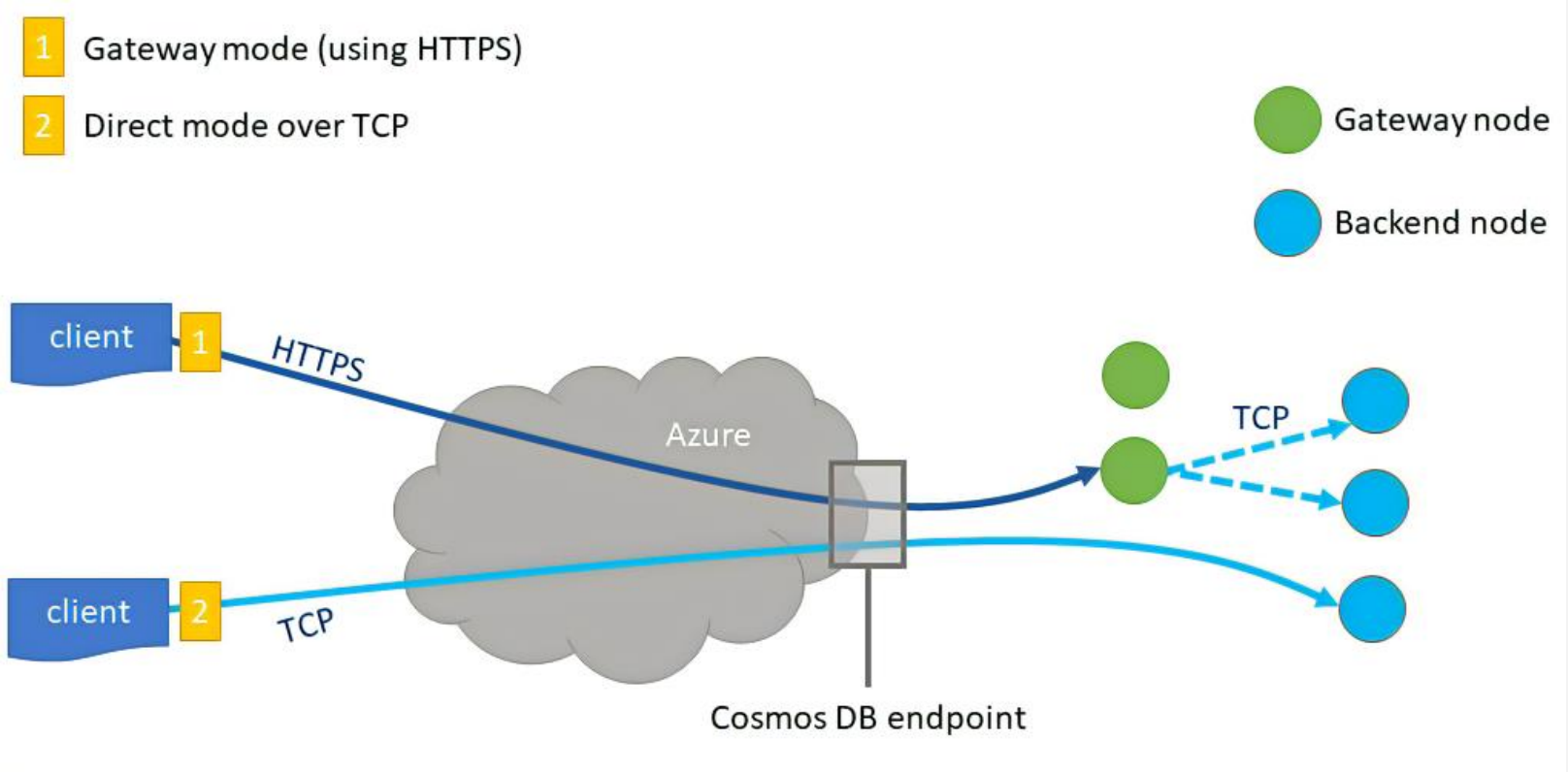
```
# Load the SynapseLink Sales Data into a Dataframe.
# Select just the "sale" document types from the sales container,
# which have a minimum _ts (timestamp) value

from pyspark.sql.functions import col

# initialize variables
begin_timestamp = 0
end_timestamp   = 1699999999

# read just the doctype "sales", not "line_item"
# "cosmos.oltp" = CosmosDB live database
# "cosmos.olap" = Synapse Link Analytic Datastore

df_sales = spark.read\
    .format("cosmos.olap")\
    .option("spark.synapse.linkedService", "CosmosSqlDemoDB")\
    .option("spark.cosmos.container", "sales")\
    .load().filter(col("doctype") == "sale")\
    .filter(col("_ts") > begin_timestamp)\
    .filter(col("_ts") < end_timestamp)

display(df_sales.limit(3))
```

INTEGRATED CACHE
GATEWAY (HTTPS) OR DIRECT (TCP)

Connection Modes:
- HTTPS (Gateway)
- TCP (Direct)

Gateways:
- Standard (default)
- Dedicated (cache)

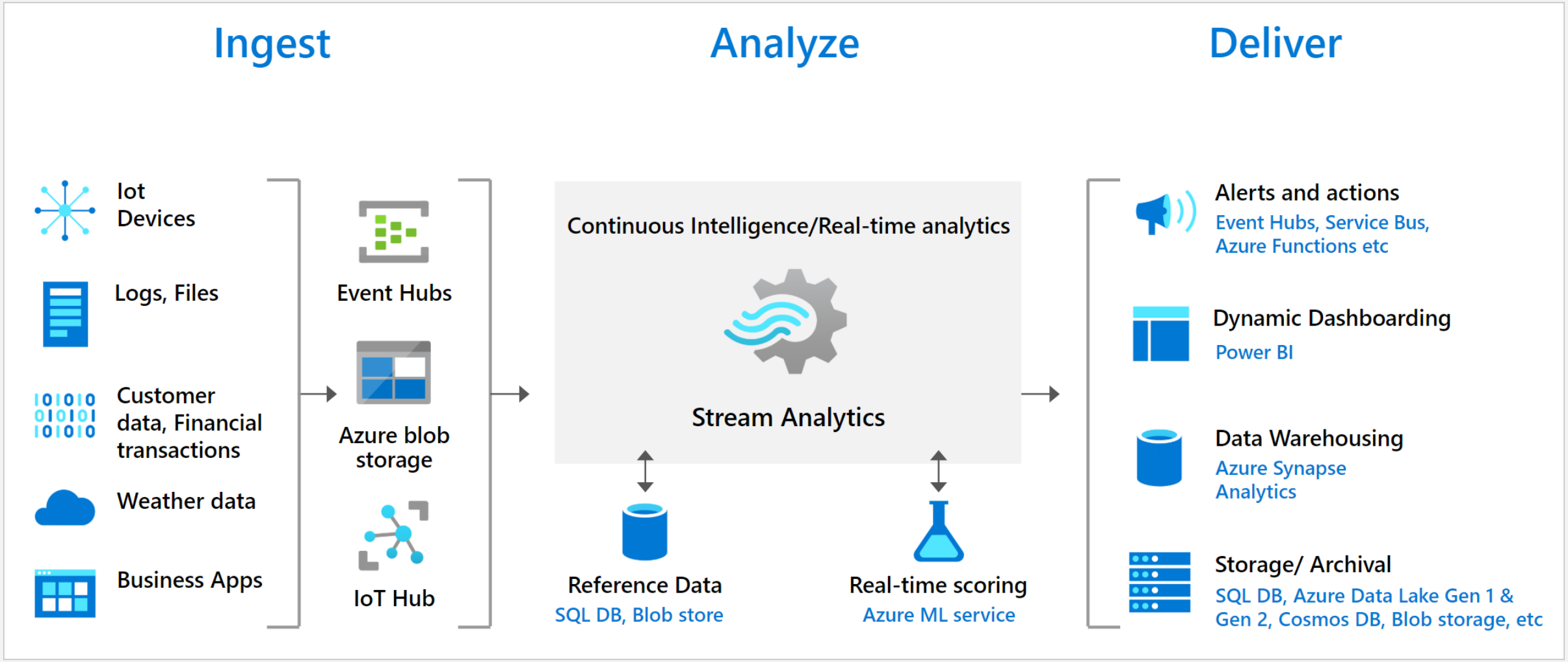
Caches:
- Sizes – S, M, L
- Types – Query, Item
- LRU algorithm
- Auto-populated



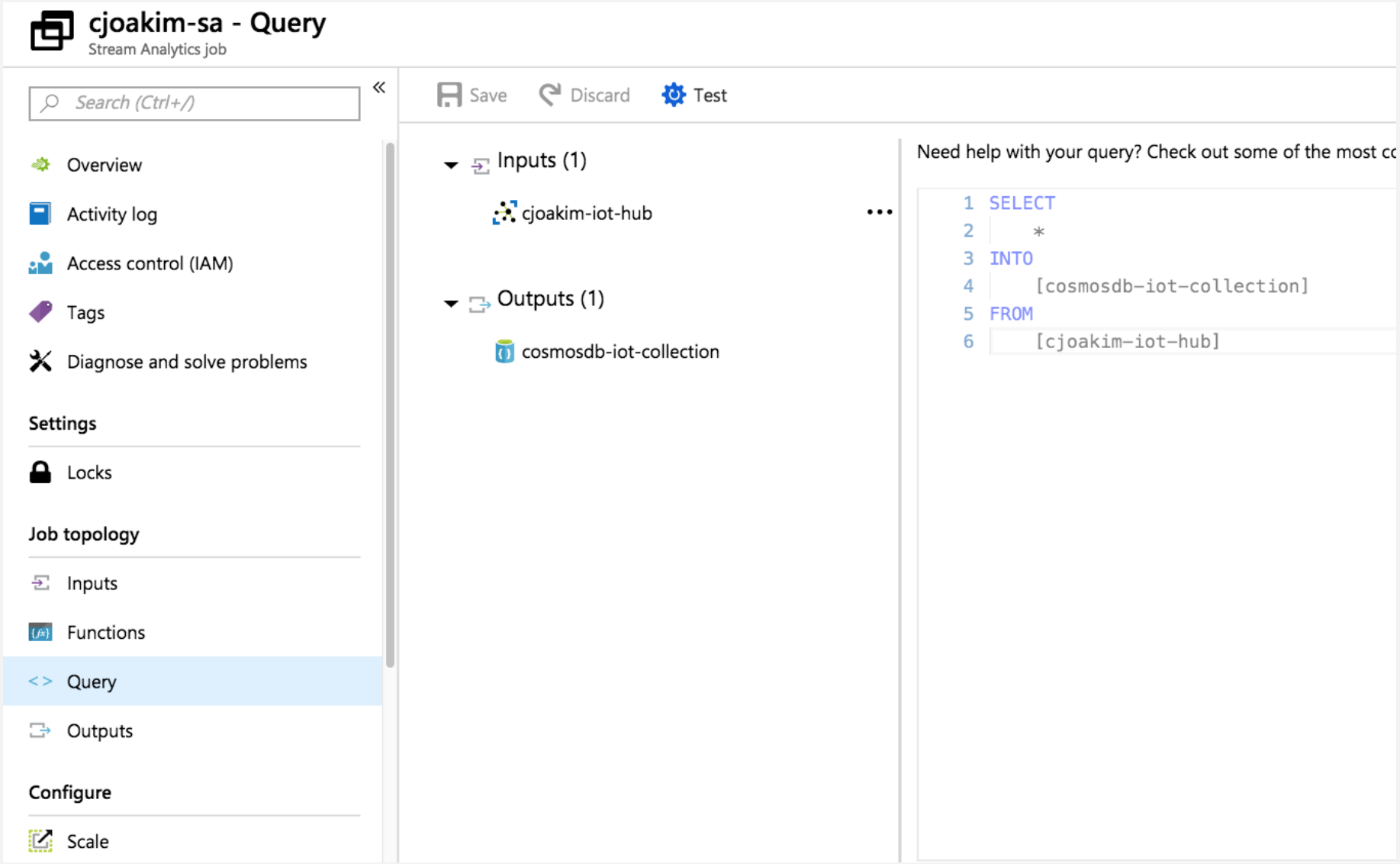
https://docs.microsoft.com/en-us/azure/cosmos-db/dedicated-gateway

INTEGRATION EXAMPLE: STREAM ANALYTICS

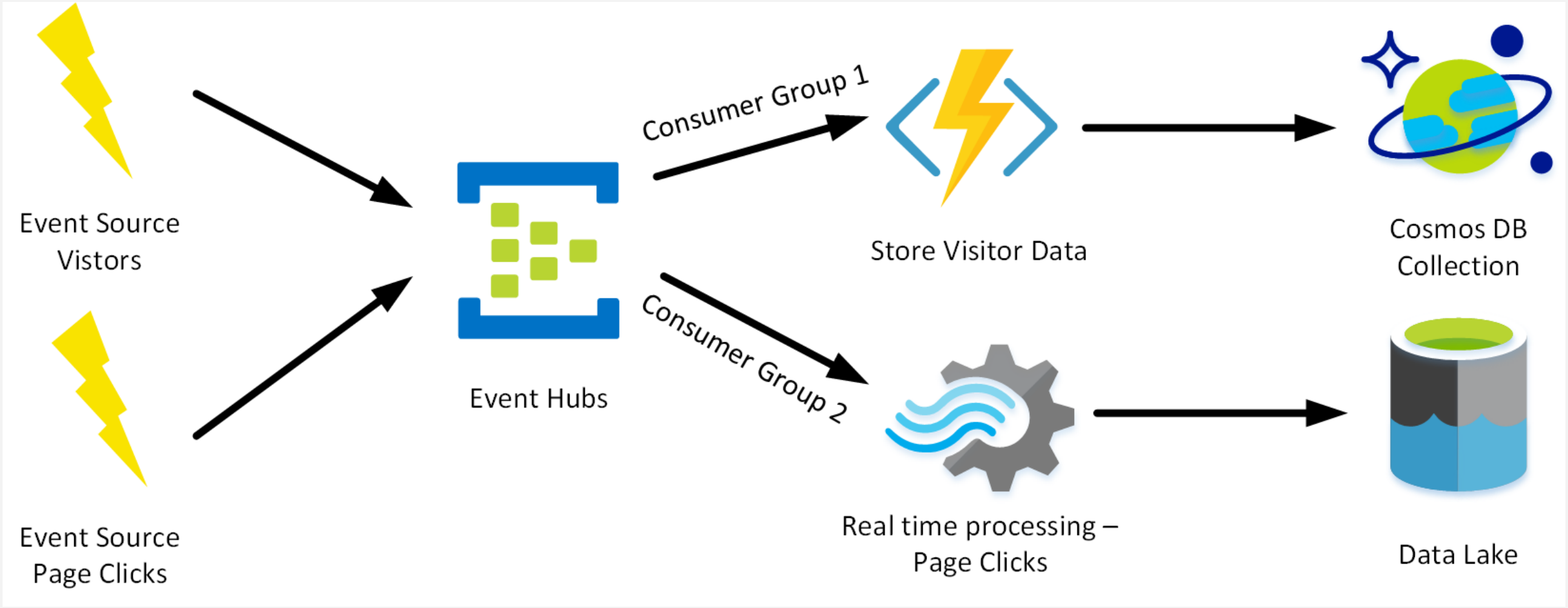
SLIDE 1: STREAM FROM SOURCE TO SINK, WITH OPTIONAL AI



INTEGRATION EXAMPLE: STREAM ANALYTICS
SLIDE 2: SQL "CODE" – SELECT FROM SOURCE INTO SINK

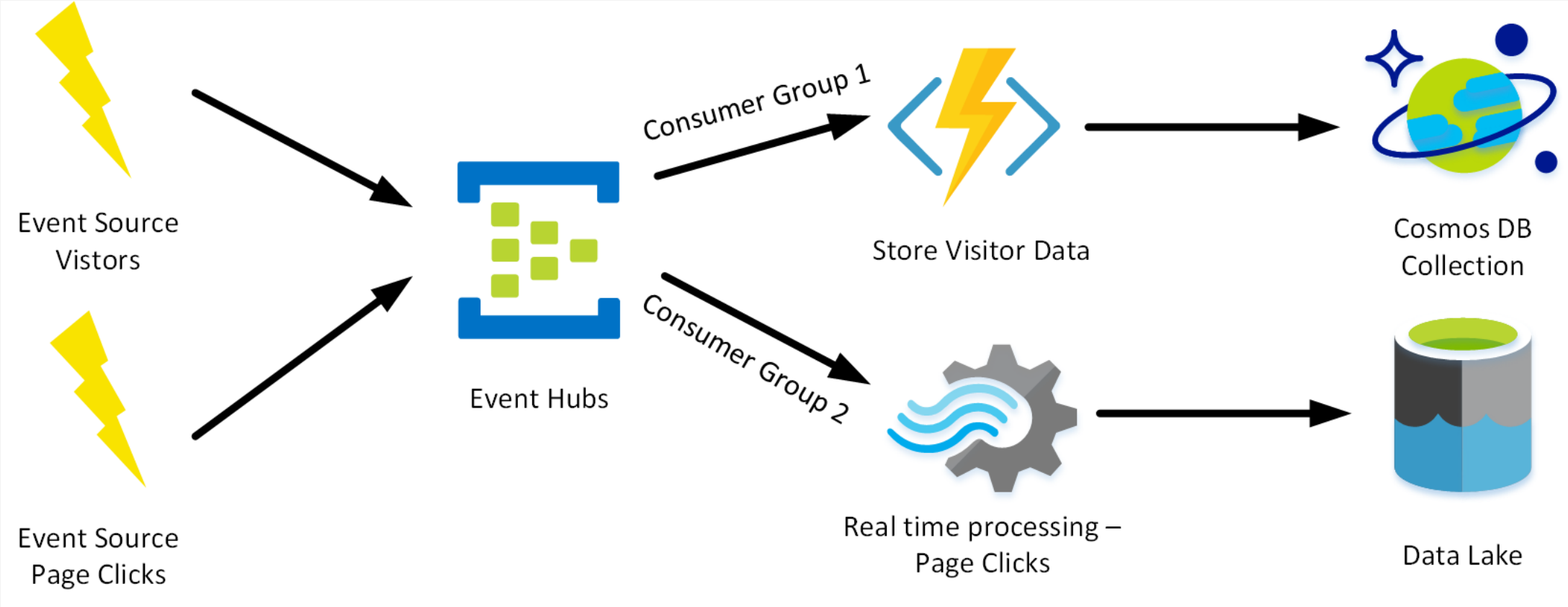


INTEGRATION EXAMPLE: STREAMING



INTEGRATION EXAMPLE: COGNITIVE SEARCH

MULTIPLE POSSIBILITIES FOR SOURCES AND SINKS



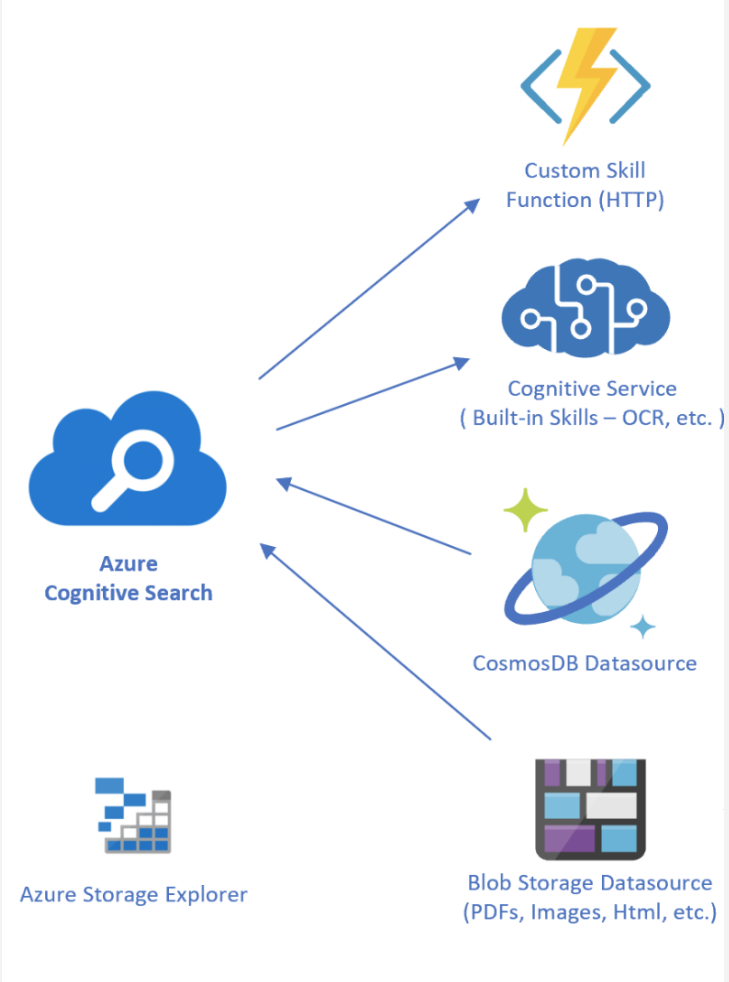
INTEGRATION EXAMPLE: AZURE COGNITIVE SEARCH

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“],

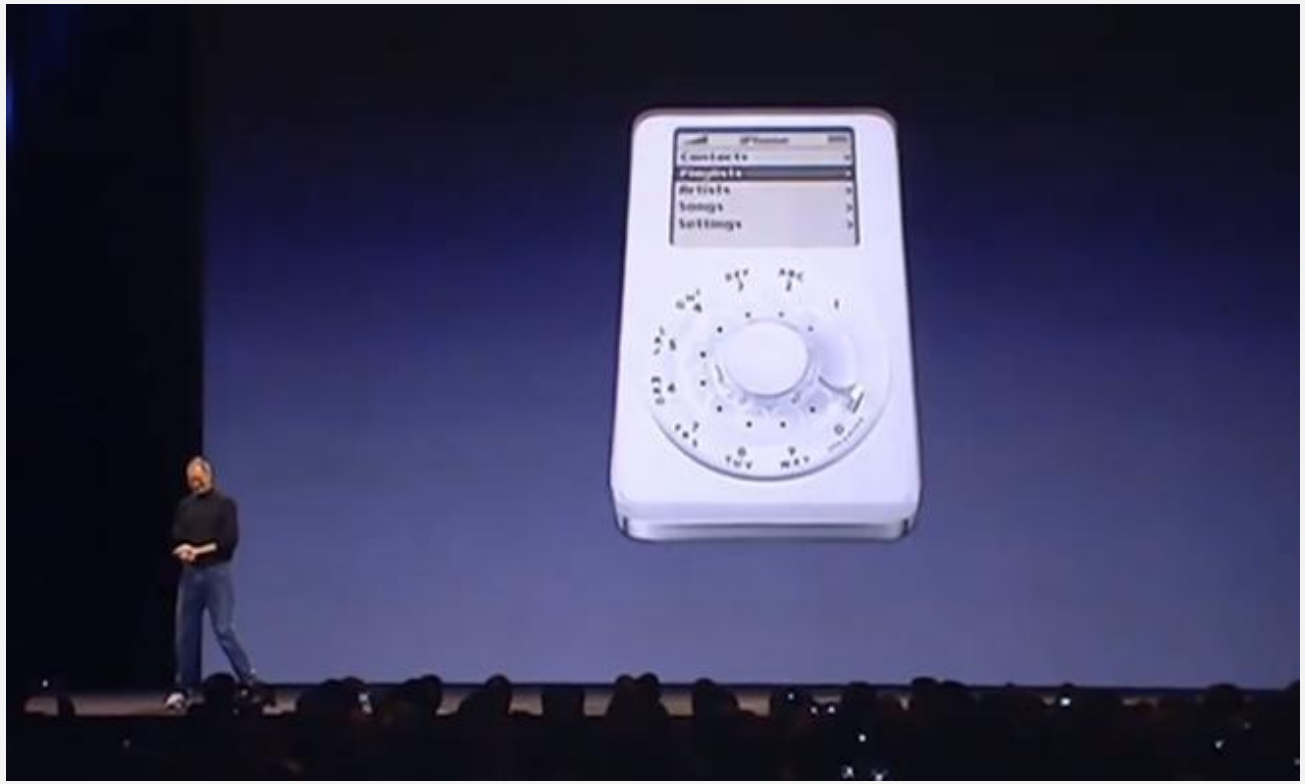


Microsoft Azure DATA & AI

Designing and Developing with CosmosDB

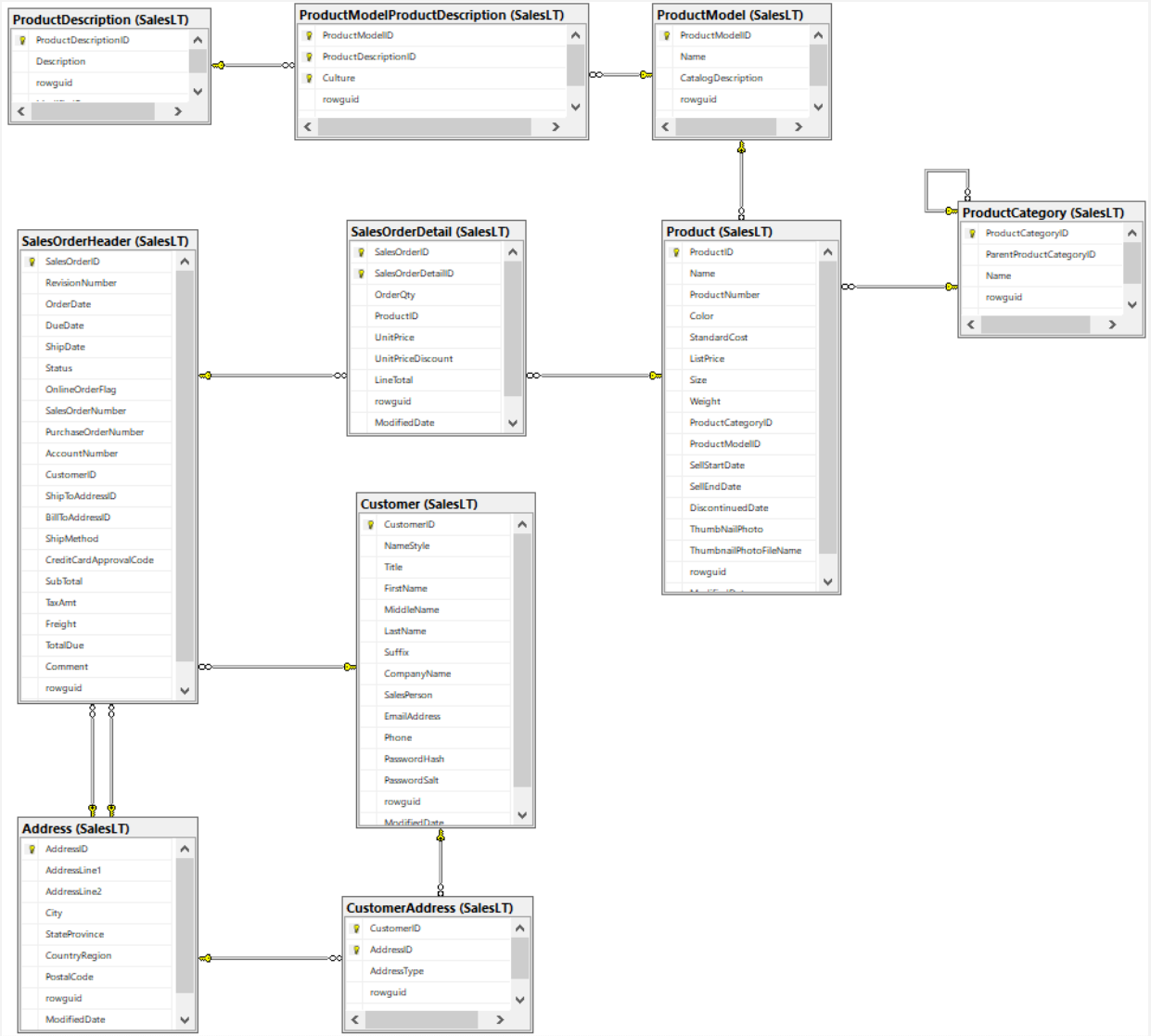
HOW DO I DESIGN THIS NEW THING?

BEWARE OF "DESIGN INERTIA" FROM YOUR PREVIOUS DBMS



DESIGN: ANTI-PATTERNS

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



DESIGN: ANTI-PATTERNS

COSMOSDB IS NOT A DATALAKE; 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 c.doctype = "order"
select * from c where c.pk = "XK1123" and c.doctype 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)
  - Heirarchical (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 a CosmosAsyncClient

```
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;
```

https://github.com/cjoakim/azure-cosmosdb-java/blob/main/console_app/src/main/java/org/cjoakim/cosmos/spring/dao/CosmosAsynchDao.java

COSMOSDB PROGRAMMING – JAVA SDK V4

- Asynch Query Example
- Obtain a FeedResponse Iterable
  - Flux toIterable()
- 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 REPOSITORY CLASS

```
@Component
@Repository
public interface TelemetryRepository extends
        CosmosRepository<TelemetryEvent, String>,
        TelemetryRepositoryExtensions {

    1 usage
    Iterable<TelemetryEvent> findByStateCode(String stateCode);

    1 usage
    Iterable<TelemetryEvent> findByStateCodeAndCountyCode(String stateCode, String countyCode);

    3 usages
    @Query("select value count(1) from c")
    long countAllDocuments();

    1 usage
    @Query( "select c.id, c.pk, c.stateCode, c.countyCode, c.siteNum, c.observationCount, c.nullObservations " +
            "from  c " +
            "where c.observationCount >= @observationCount " +
            "and   c.nullObservations >= @nullObservations")
    Iterable<TelemetryEvent>  findByObservationCount(
            @Param("observationCount") long observationCount,
            @Param("nullObservations") long nullObservations);
}
```

JAVA: SPRING AND SPRING DATA

A REPOSITORY EXTENSION INTERFACE TO EXTEND THE SPRING DATA API

```java
/**
 * This interface was created to extend the TelemetryRepository, which in turn extends
 * CosmosRepository<TelemetryEvent, String> from the CosmosDB Spring Data SDK.
 *
 * This demonstrates how to leverage more of the power of the CosmosDB SQL syntax, by using
 * "Criteria" objects and an Autowired "CosmosTemplate" object.
 *
 * See class TelemetryRepositoryExtensionsImpl in this package, which implements this interface.
 *
 * Chris Joakim, Microsoft, September 2022
 */
2 usages  2 implementations
public interface TelemetryRepositoryExtensions {

    1 usage  1 implementation
    public Iterable<TelemetryEvent> findByStateCodeAndSiteNumbers(String stateCode, ArrayList<String> siteNumbers);
}
```

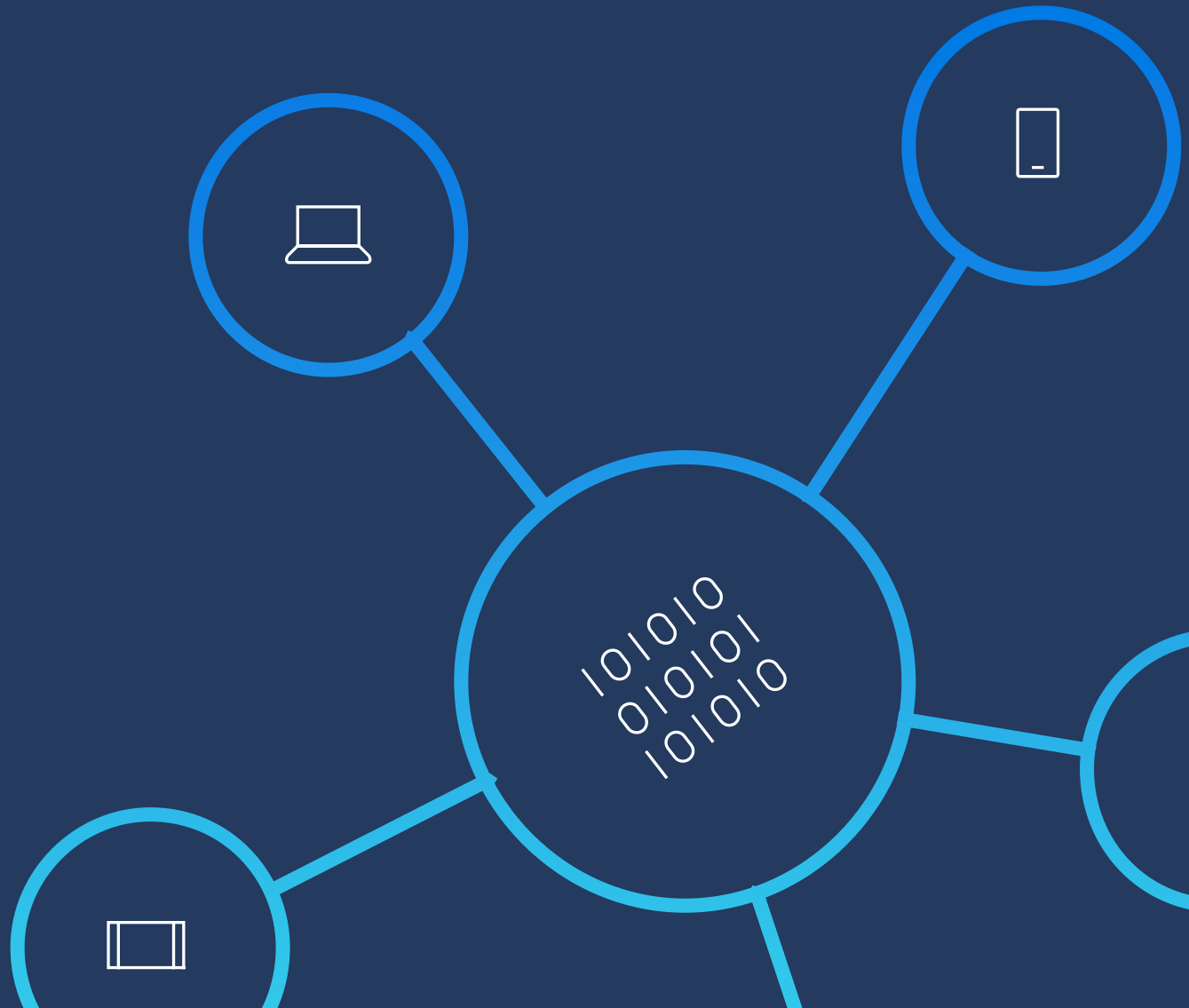
JAVA: SPRING AND SPRING DATA
A REPOSITORY EXTENSION INTERFACE IMPLEMENTATION

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

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

Microsoft Azure DATA & AI

Questions & Discussion



Microsoft

Thank You!