

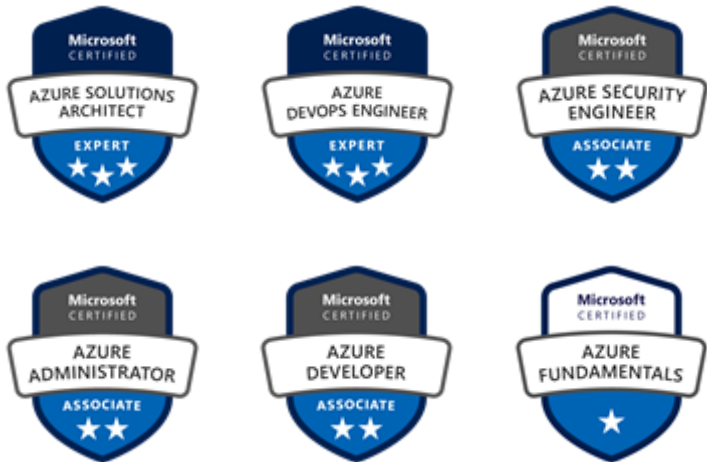
Azure cosmos db for developers



Vaibhav Gujral
<https://vaibhavgujral.com/>

About me

- 13+ years of experience across designing and developing enterprise-class applications
- Microsoft Certified Azure Solutions Architect Expert
- Cloud Architect, Kiewit
- Co-organizer, Omaha Azure User Group
- Speaker | Blogger



<http://www.vaibhavgujral.com>
[@vabgujral](mailto:gujral.vaibhav@gmail.com)
gujral.vaibhav@gmail.com

Agenda

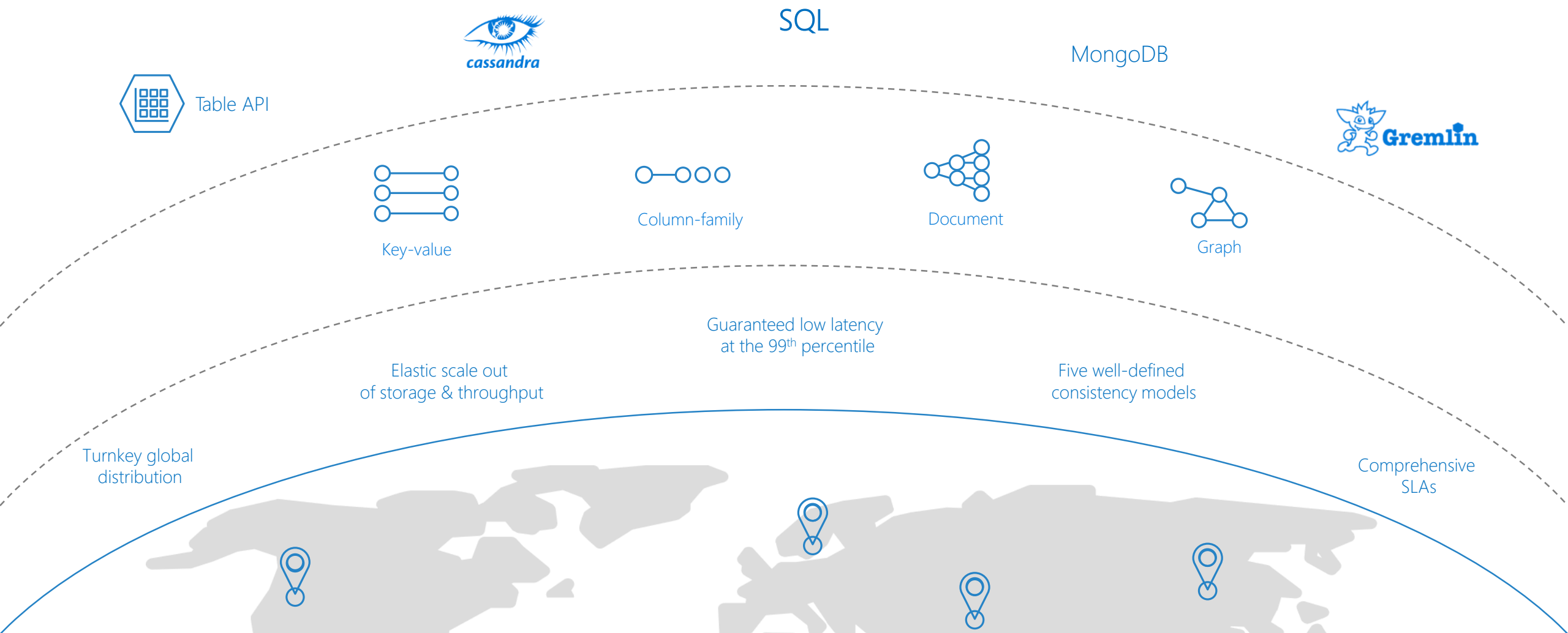
- Azure Cosmos DB Overview
 - Resource Model
 - Partitioning
 - Request Units and Billing
 - Consistency Models
 - Conflict Resolution
 - Change Feed
 - Use Cases

AZURE COSMOS DB

A FULLY-MANAGED GLOBALLY DISTRIBUTED DATABASE SERVICE BUILT TO GUARANTEE
EXTREMELY LOW LATENCY AND MASSIVE SCALE FOR MODERN APPS

AZURE COSMOS DB

A globally distributed, massively scalable, multi-model database service



Multiple Data Models and APIs

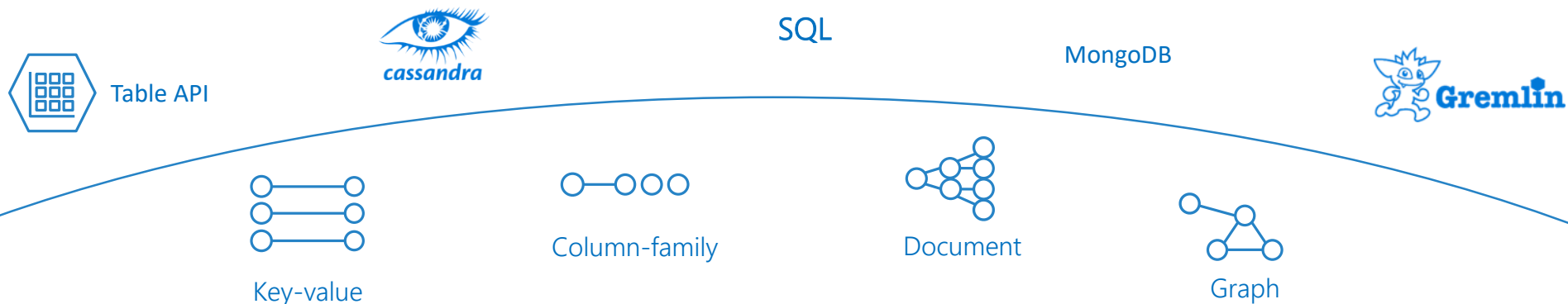
Use the model that fits your requirements, and the APIs, tools and frameworks you prefer

Cosmos DB offers a multitude of APIs to access and query data including, SQL, various popular OSS APIs, and native support for NoSQL workloads.

Use key-value, tabular, graph, and document data

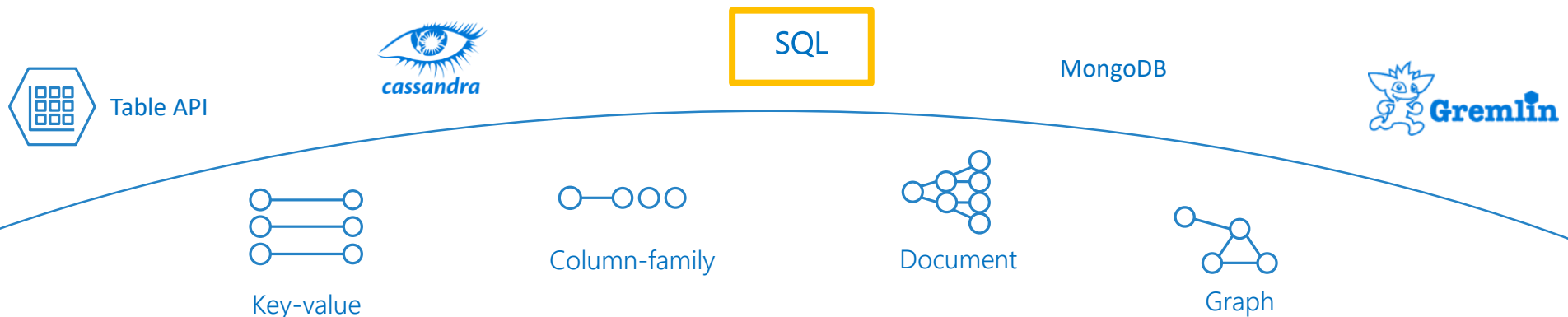
Data is automatically indexed, with no schema or secondary indexes required

Blazing fast queries with no lag



Which API should I use for my App?

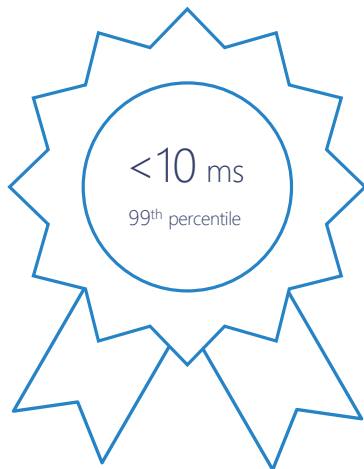
- For all new workloads – Core (SQL) API
 - Best developer experience – Cosmos builds interface, service, and SDKs
 - Gremlin API for graph data
- Other APIs for migration



Comprehensive SLAs

Azure Cosmos DB is the only service with financially-backed SLAs for millisecond latency at the 99th percentile, 99.999% HA and guaranteed throughput and consistency

Latency



HA



Throughput



Consistency



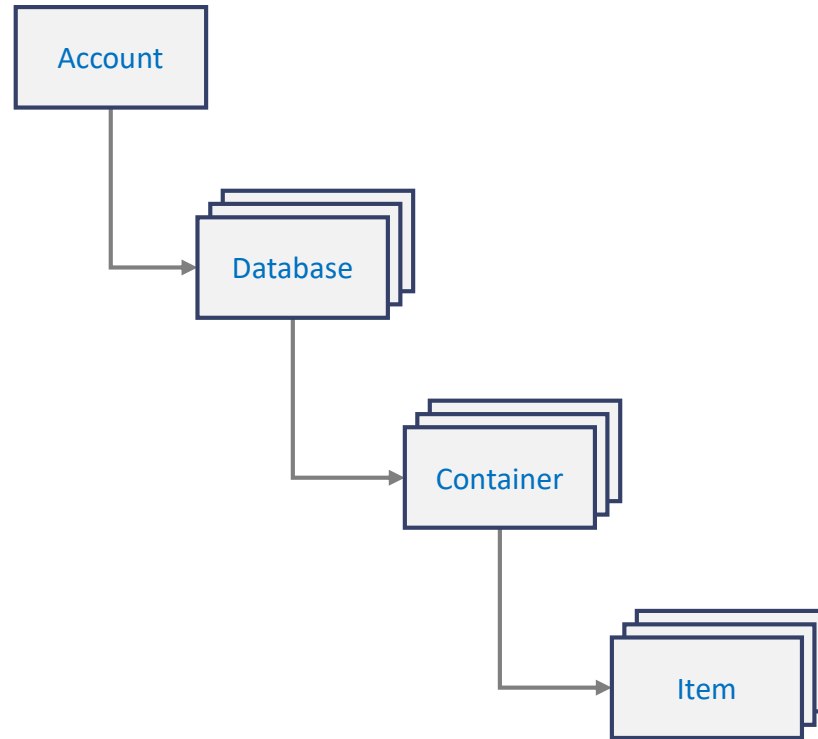
When should I use Azure Cosmos DB?

One or more of these requirements...

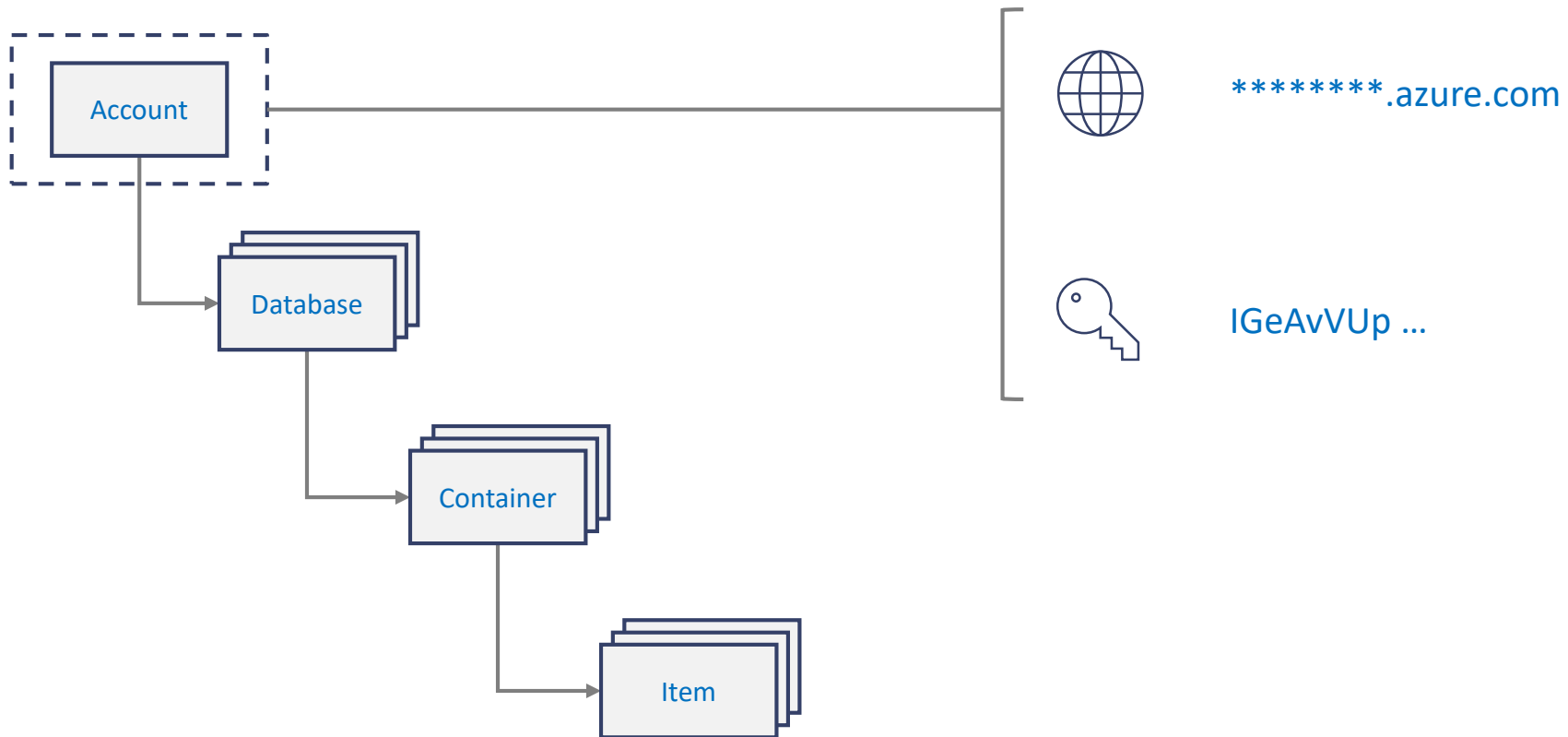
- Flexible schema
- Scalability
 - Especially for writes
- Low latency (fast)
- High availability

Resource Model

Resource Model



Account URI and Credentials

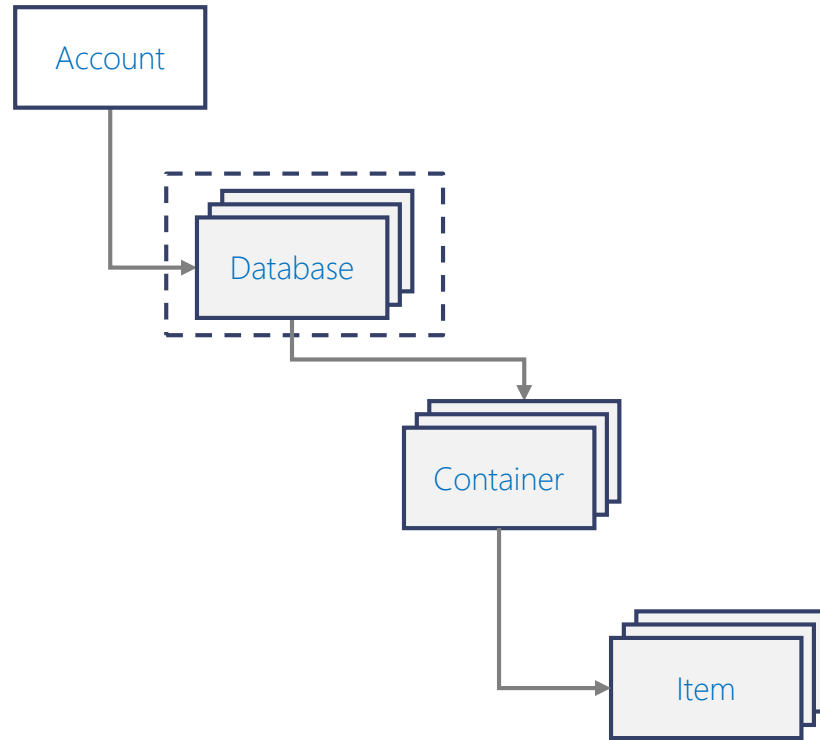


Creating Account

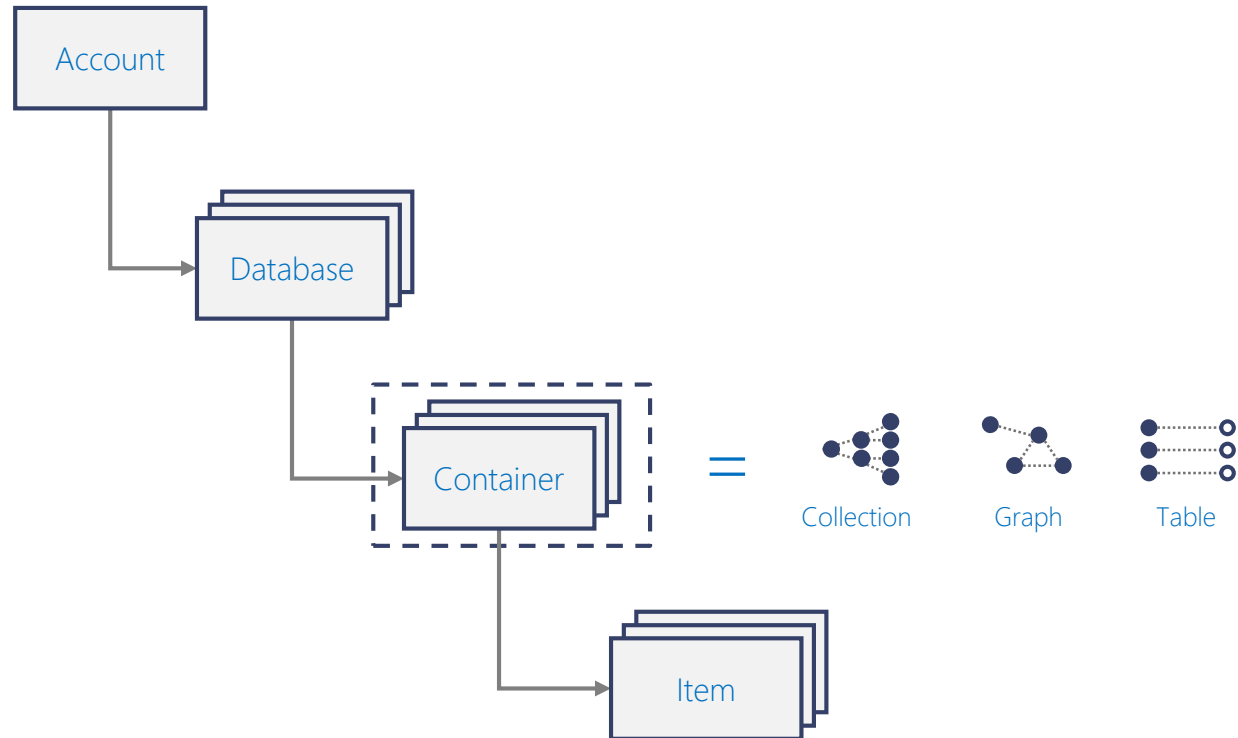
The diagram on the left illustrates the hierarchy of Azure Cosmos DB resources: an **Account** (represented by a dashed box) contains multiple **Databases**, which in turn contain multiple **Containers**, and each container contains multiple **Items**. Arrows indicate the flow from the Account to Databases, then to Containers, and finally to Items.

The screenshot on the right shows the **Create Azure Cosmos DB Account** page in the Microsoft Azure portal. The page includes a sidebar with navigation options like 'Create a resource', 'Home', 'Dashboard', and 'All services'. The main content area is titled 'Create Azure Cosmos DB Account' and features a promotional banner about a 33% discount. Below the banner, there are tabs for 'Basics', 'Networking', 'Tags', and 'Review + create'. The 'Basics' tab is active, showing fields for 'Subscription', 'Resource Group', 'Account Name', 'API' (set to 'Core (SQL)'), 'Location' (set to '(US) Central US'), and options for 'Geo-Redundancy', 'Multi-region Writes', and 'Availability Zones'. At the bottom, there are buttons for 'Review + create', 'Previous', and 'Next: Networking'.

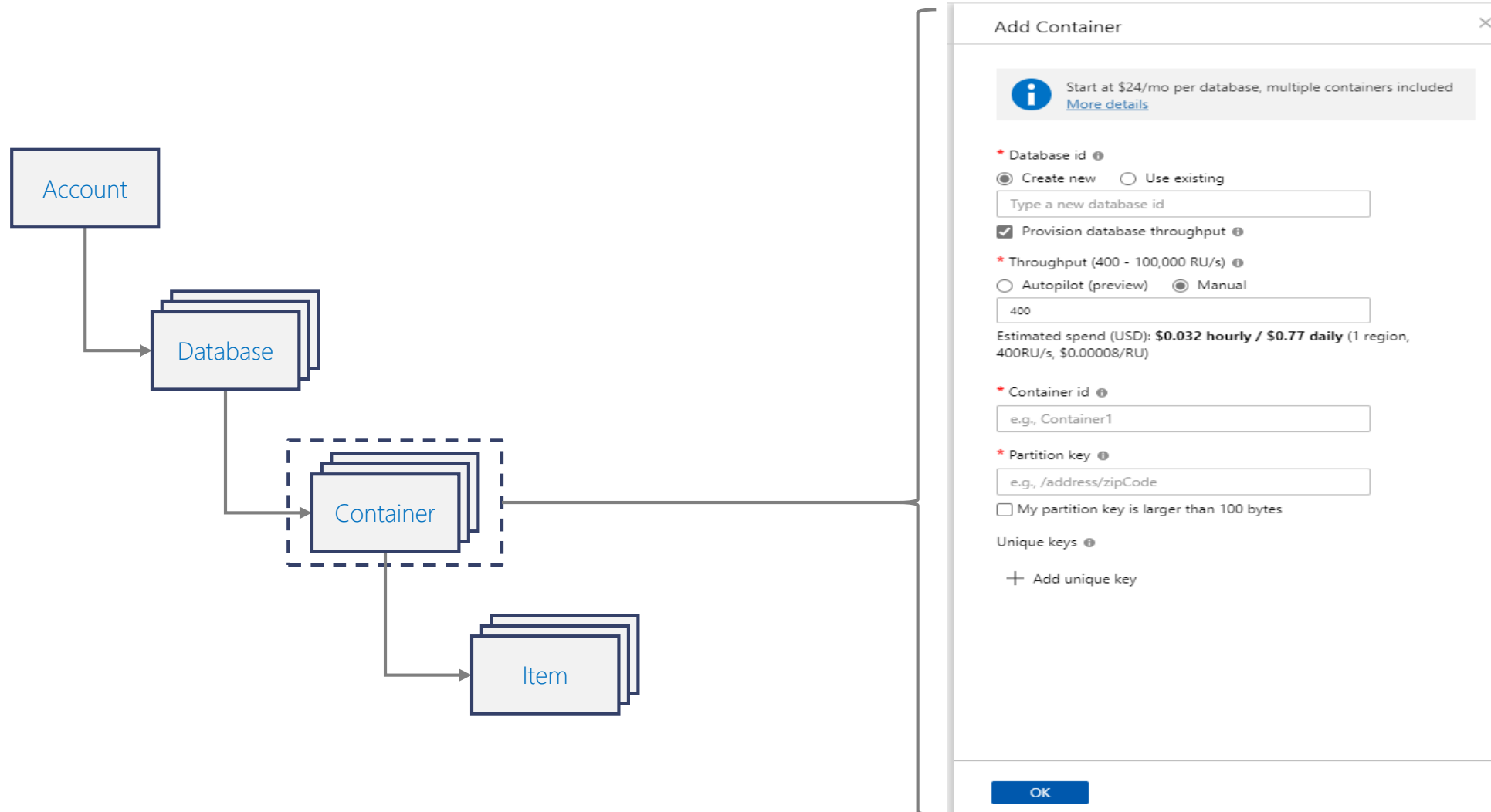
Database Representations



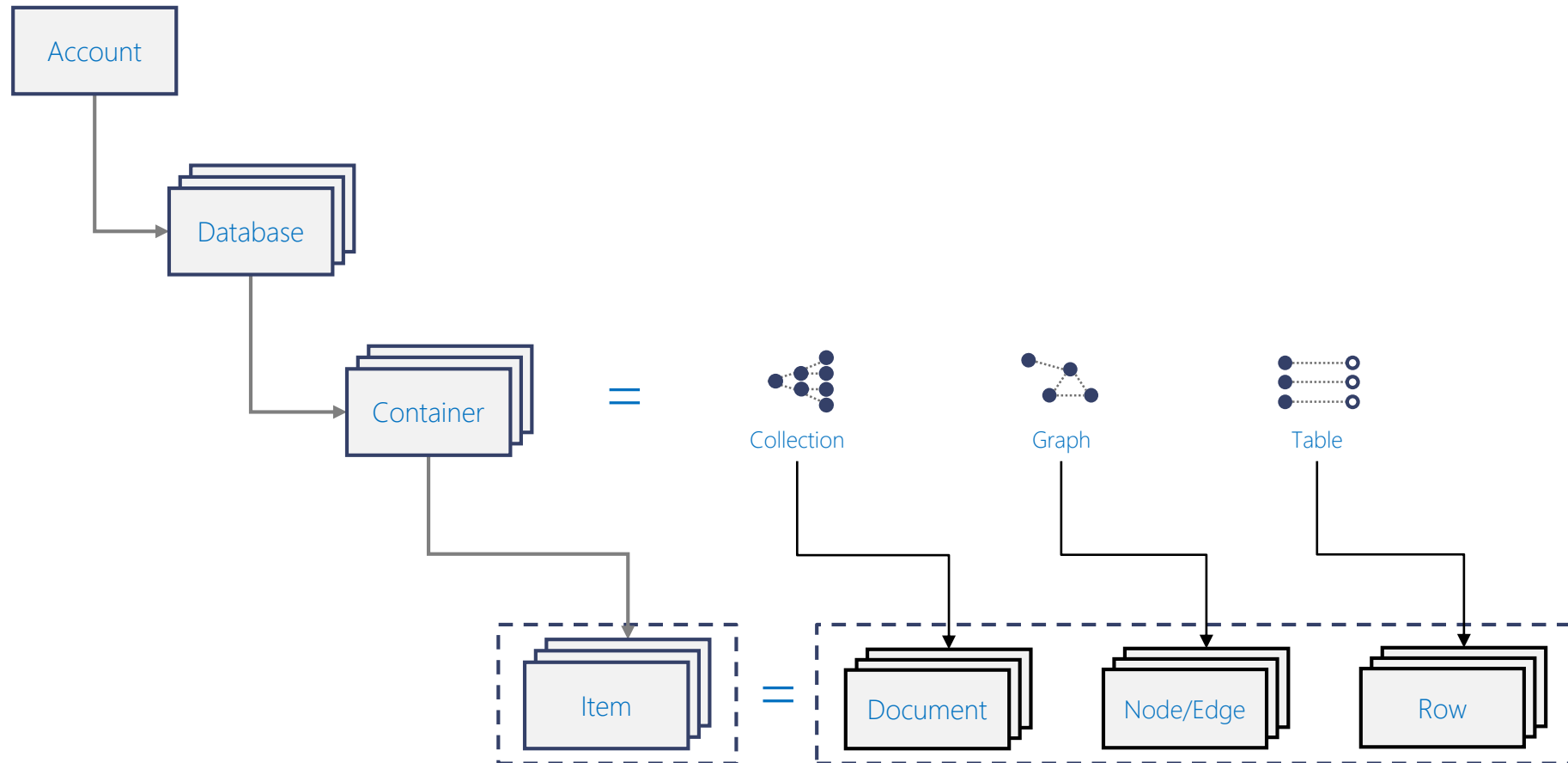
Container Representations



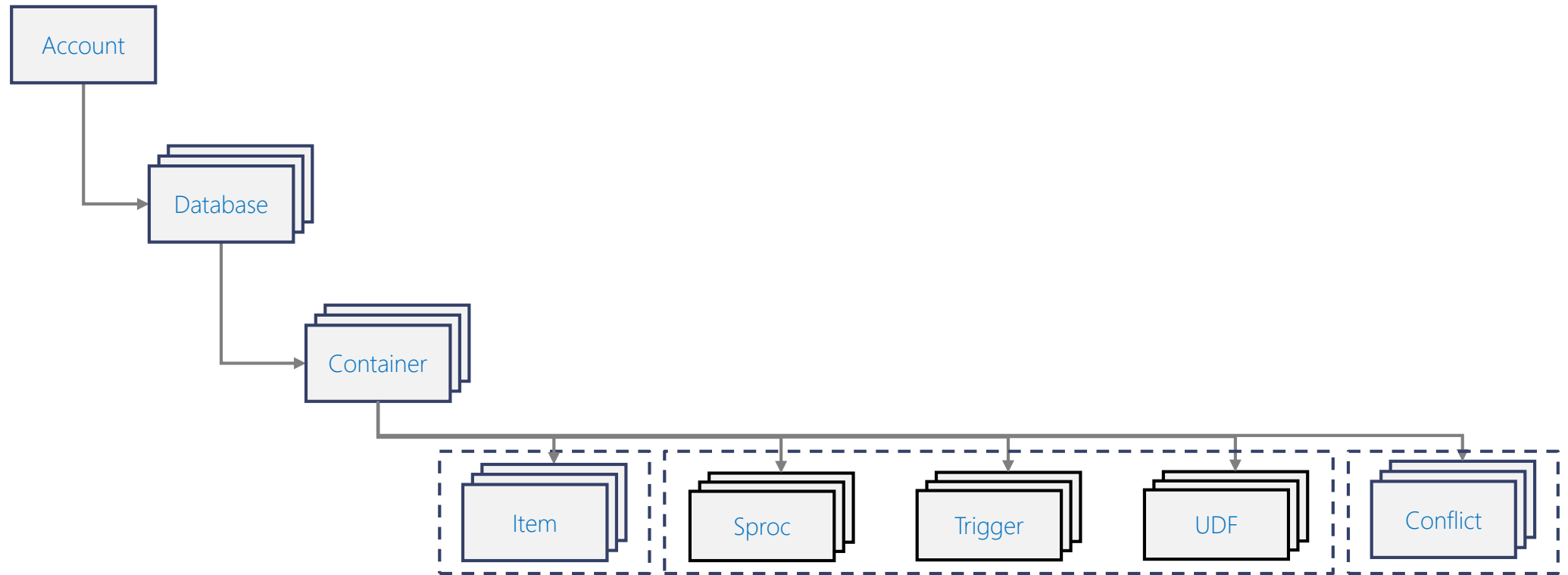
Creating Collections – SQL API



Container-level resources



Container-level resources



Turnkey Global Distribution

High Availability

- Automatic and Manual Failover
- Multi-homing API removes need for app redeployment

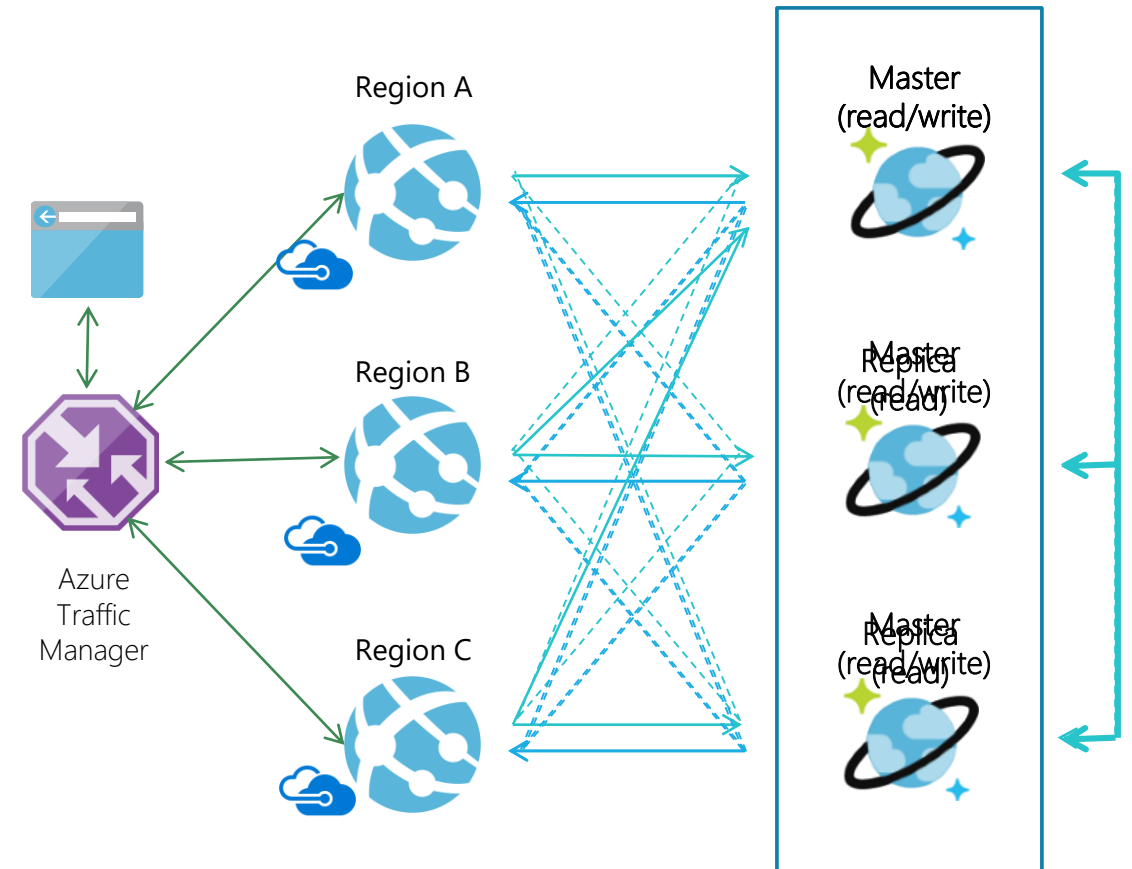
Low Latency (anywhere in the world)

- Packets cannot move faster than the speed of light
- Sending a packet across the world under ideal network conditions takes 100's of milliseconds
- You can cheat the speed of light – using data locality
 - CDN's solved this for static content
 - Azure Cosmos DB solves this for dynamic content



Multi-master

Every region writable
<10 ms read write latency
99.999% Availability
Unlimited endpoint scalability
Flexible conflict resolution



Enabling Multi Master

Create new Cosmos DB account

Enable Multi Master

The screenshot shows the 'Create Azure Cosmos DB Account' page in the Microsoft Azure portal. The left sidebar contains navigation links for various Azure services. The main content area is titled 'Create Azure Cosmos DB Account' and includes a promotional banner, tabs for 'Basics', 'Networking', 'Tags', and 'Review + create', and a 'Project Details' section. The 'Basics' tab is active, showing fields for 'Subscription', 'Resource Group', 'Account Name', 'API', 'Location', 'Geo-Redundancy', 'Multi-region Writes', and 'Availability Zones'. A red arrow points to the 'Multi-region Writes' section, which has 'Enable' and 'Disable' buttons. The 'Disable' button is currently selected. At the bottom, there are buttons for 'Review + create', 'Previous', and 'Next: Networking'.

Microsoft Azure

Search resources, services, and docs (G+J)

Home > New > Create Azure Cosmos DB Account

Create Azure Cosmos DB Account

Create a new Azure Cosmos DB account with multi-region writes in any region by February 29, 2020 and receive up to 33% off for the life of your account. Restrictions apply.*

Basics Networking Tags Review + create

Azure Cosmos DB is a globally distributed, multi-model, fully managed database service. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Visual Studio Enterprise with MSDN

Resource Group * Select existing... [Create new](#)

Instance Details

Account Name * Enter account name

API * Core (SQL)

Apache Spark Notebooks (preview) Notebooks with Apache Spark (preview) **None**
[Sign up for Apache Spark preview](#)

Location * (US) Central US

Geo-Redundancy Enable **Disable**


Multi-region Writes Enable **Disable**

Availability Zones Enable **Disable**

*Up to 33% off multi-region writes is available to qualifying new accounts only. Accounts must be created between December 1, 2019 and February 29, 2020. Offer limited to accounts with both account locations and geo-redundancy, and applies only to multi-region writes in those same regions. Both Geo-Redundancy and Multi-region Writes must be enabled in account settings. Actual discount will vary based on number of qualifying regions selected.

[Review + create](#) [Previous](#) [Next: Networking](#)

Replicating Data Globally

 **andri-global - Replicate data globally**
Azure Cosmos DB account

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Quick start

Data Explorer


SETTINGS


Replicate data globally


Default consistency


Firewall

Keys

 Save

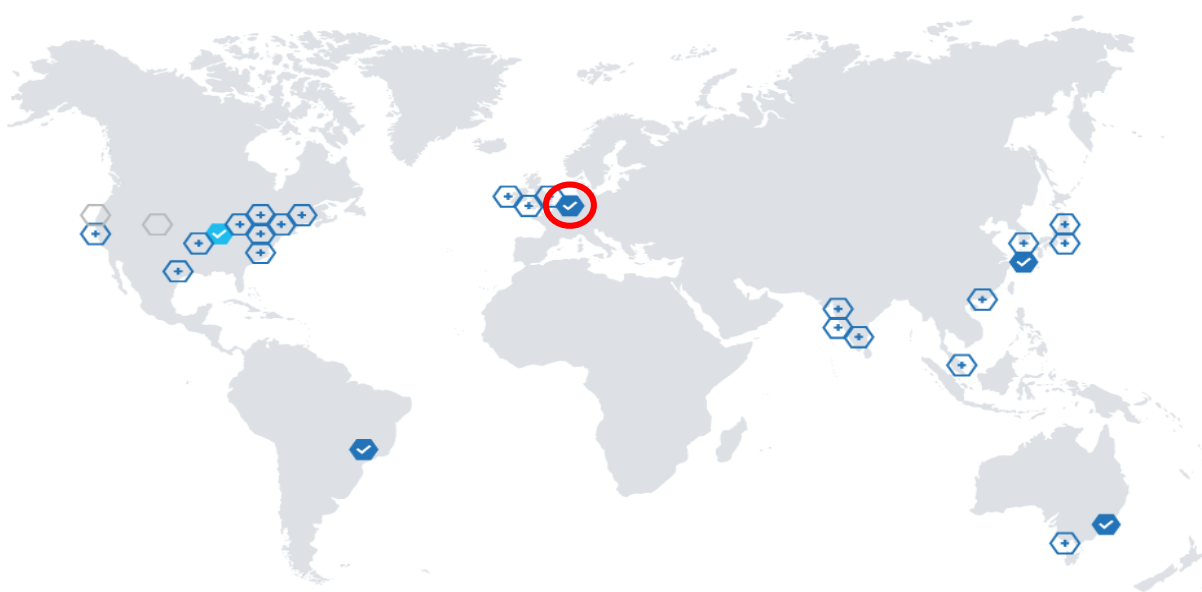
 Discard

 Manual Failover

 Automatic Failover

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](#)



Automatic Failover


Automatic Failover

Enable Automatic Failover ⓘ

ON

OFF

Drag-and-drop read regions items to reorder the failover priorities.

Tip: Drag  on the left of the hovered row to reorder the list.

WRITE REGION	
Central US	
READ REGIONS	PRIORITIES
North Europe	1
Southeast Asia	2

Manual Failover

Manual Failover

×

Select a Read Region to become the new Write Region.

Tip: Identify all dependent services leveraging this account and ensure that triggering a failover will not jeopardize your production application.

WRITE REGION

Central US

READ REGIONS

Southeast Asia

North Europe

☒

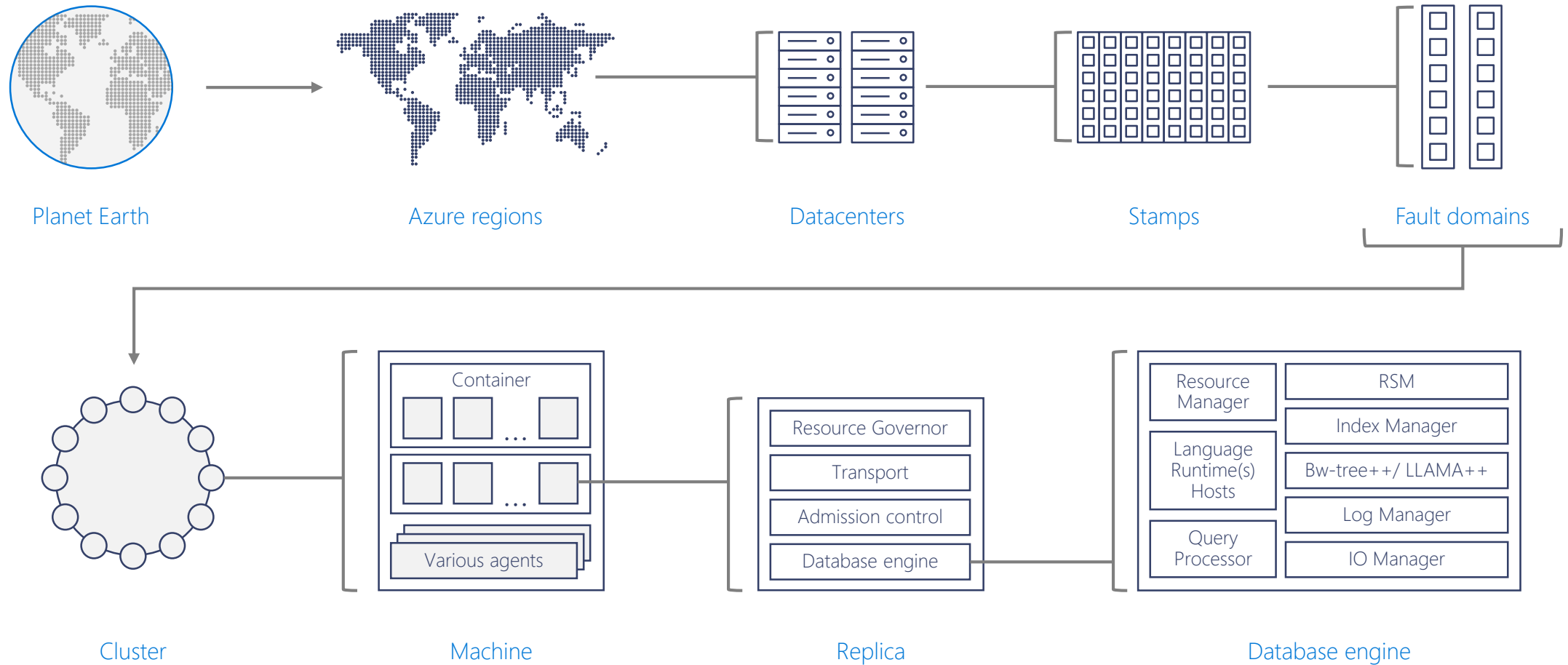
 I understand and agree to trigger a failover on my current Write Region.

OK

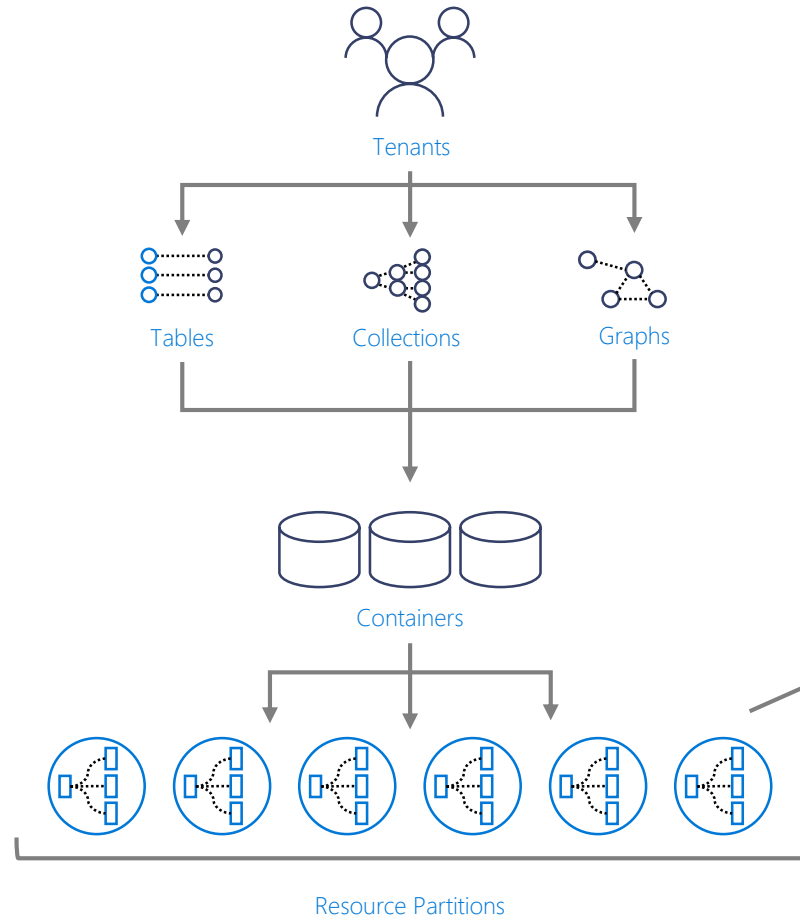
Demo-1

Partitioning

System Topology

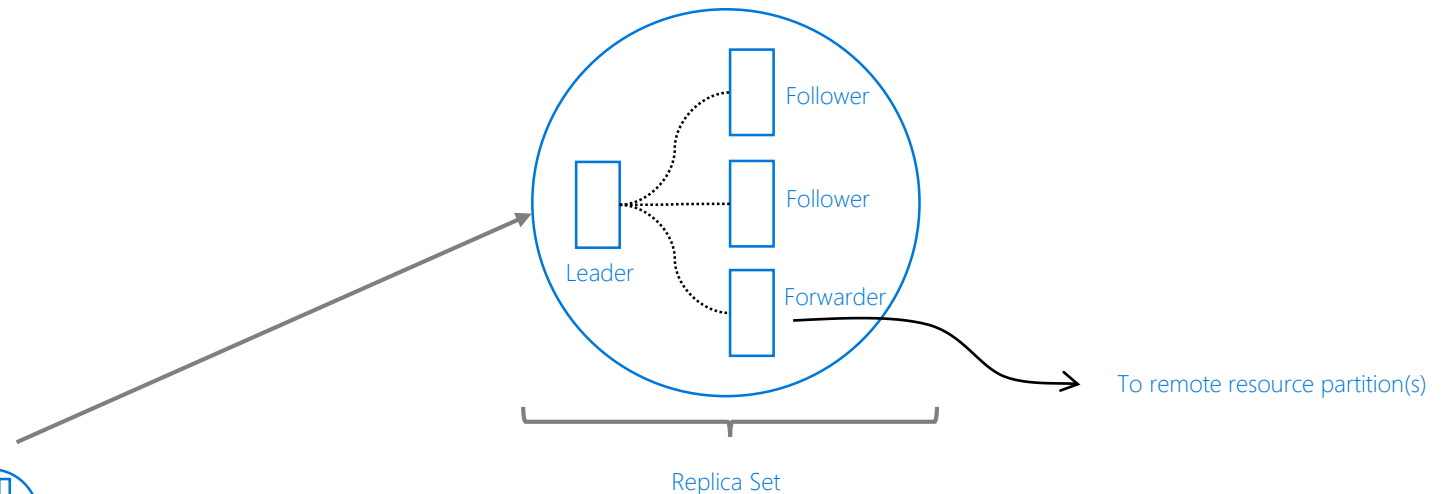


Resource Hierarchy



Containers

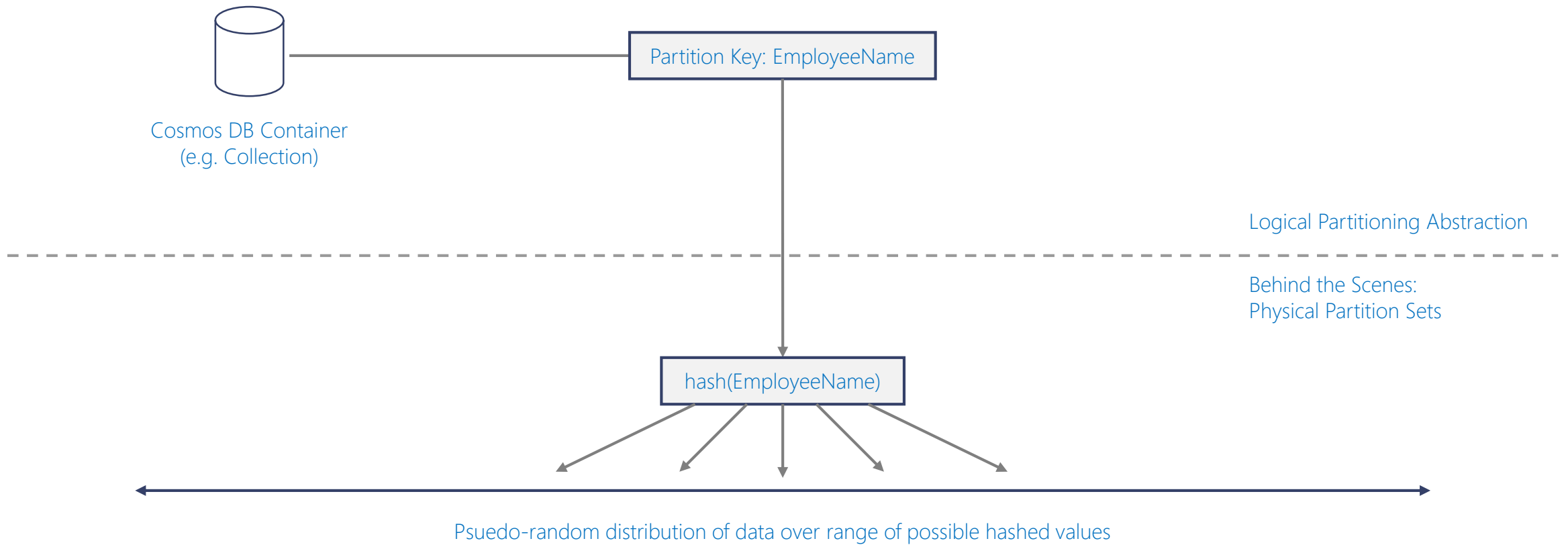
Logical resources “surfaced” to APIs as tables, collections or graphs, which are made up of one or more physical partitions or servers.



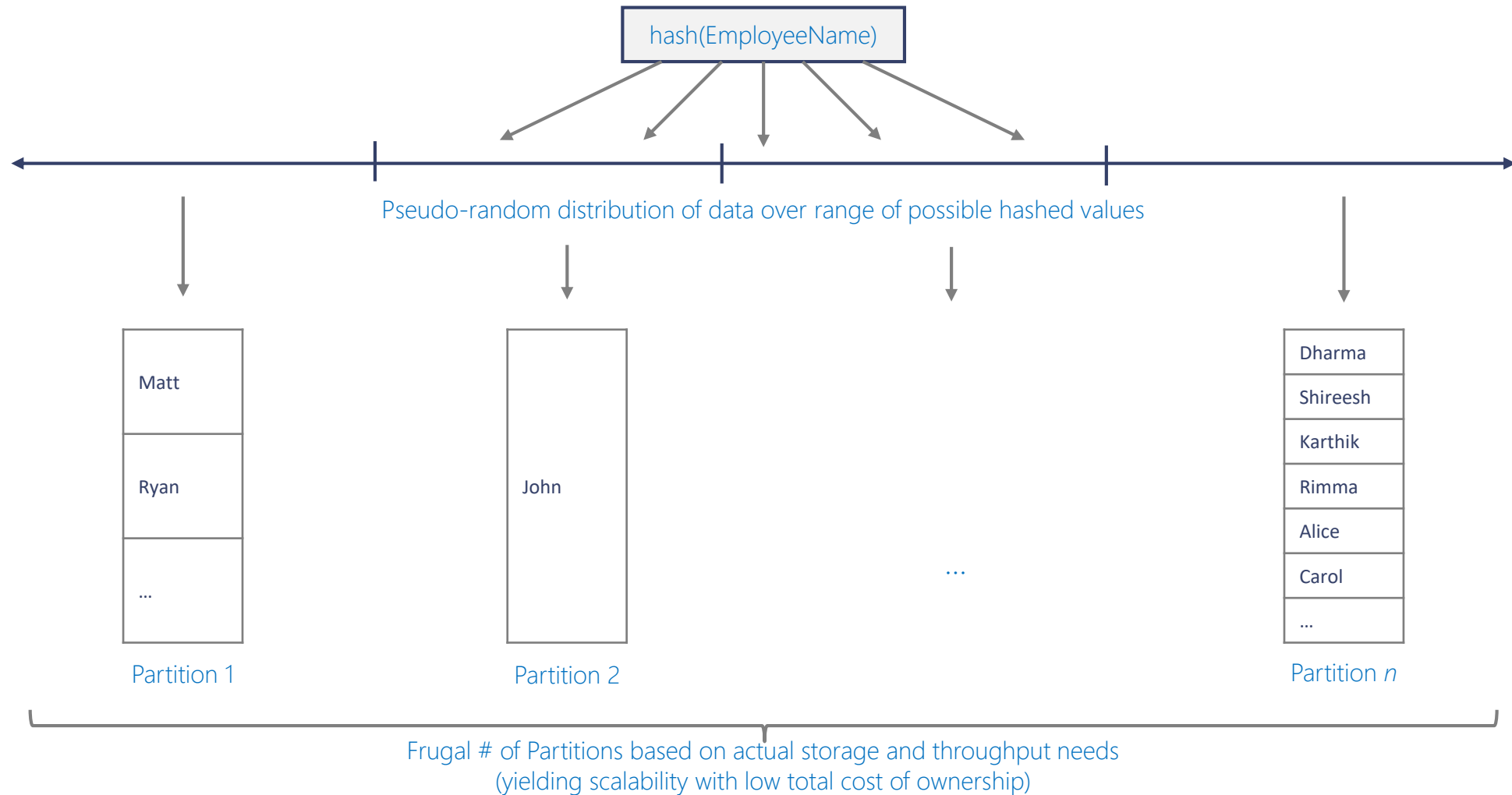
Resource Partitions

- Consistent, highly available, and resource-governed coordination primitives
- Consist of replica sets, with each replica hosting an instance of the database engine

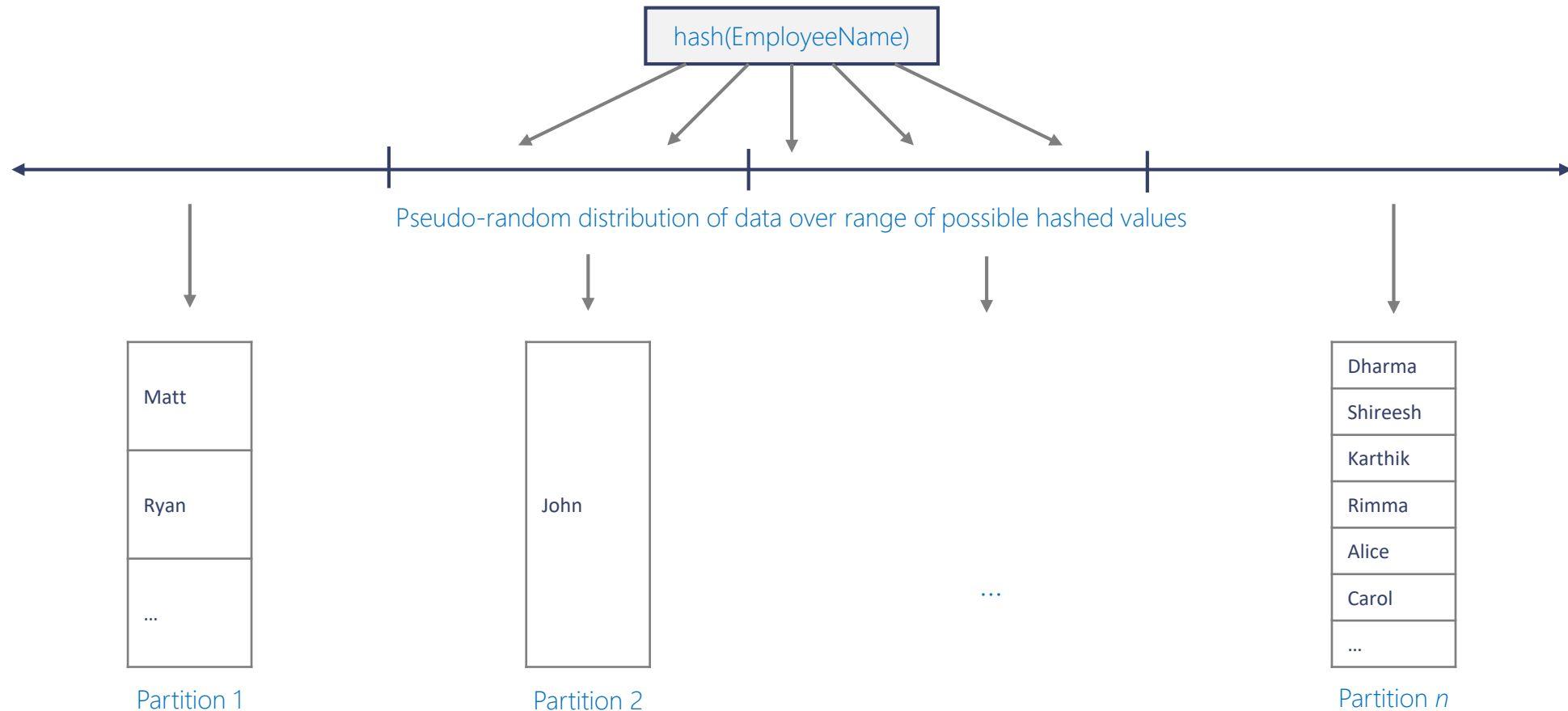
Partitions



Partitions

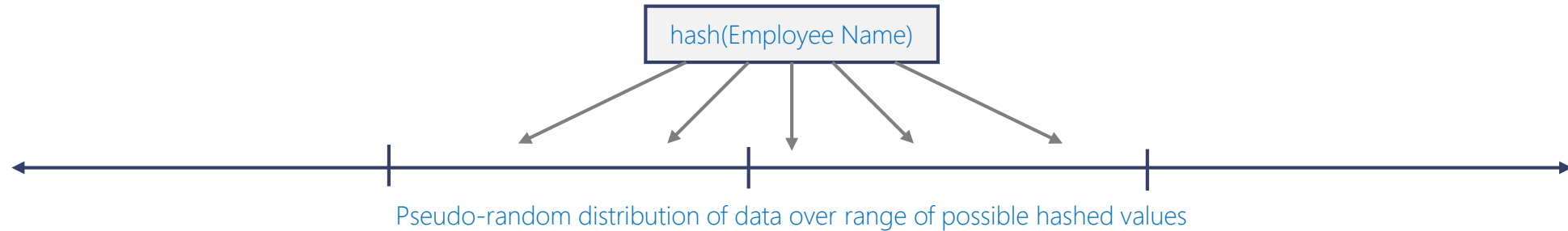


Partitions



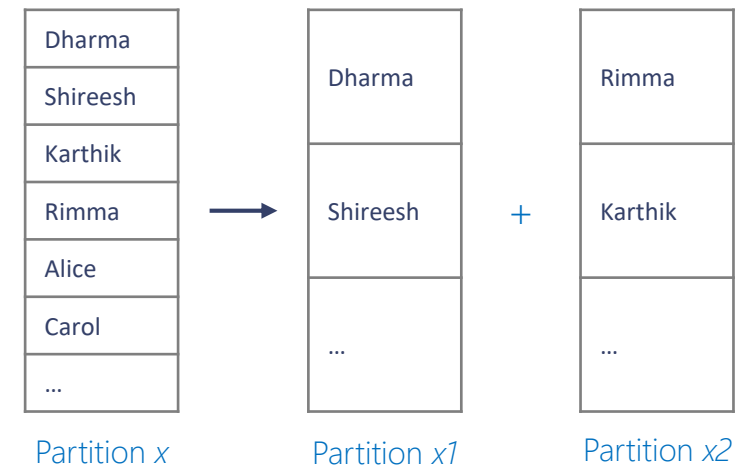
What happens when partitions need to grow?

Partitions



Partition Ranges can be dynamically sub-divided to seamlessly grow database as the application grows while simultaneously maintaining high availability.

Partition management is fully managed by Azure Cosmos DB, so you don't have to write code or manage your partitions.



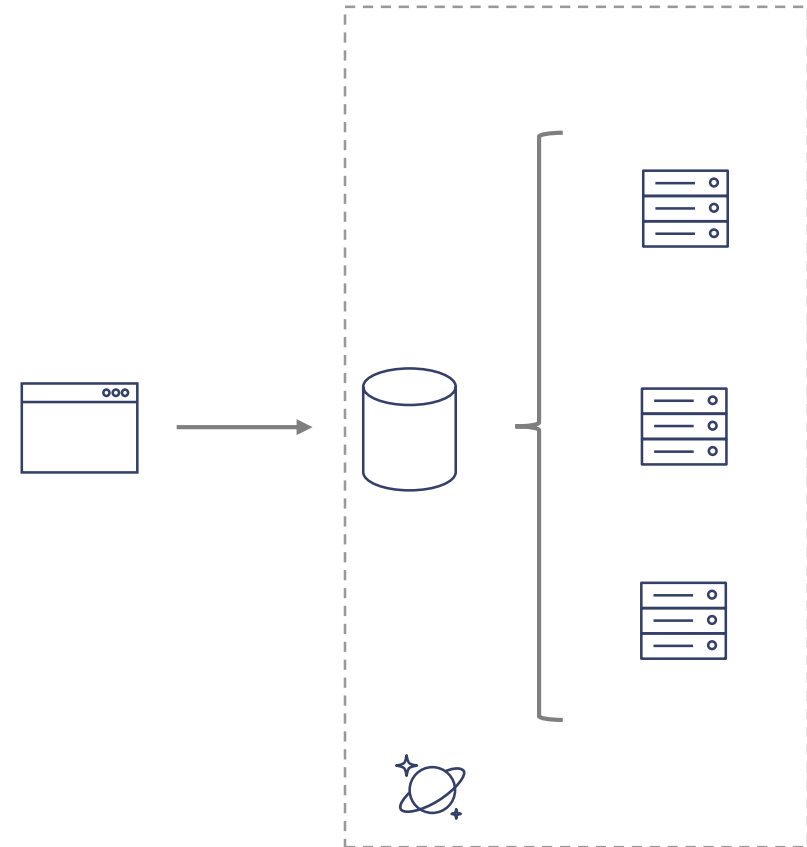
Partition Design

Important to select the “**right**” partition key as it’s the key to scaling

Partition keys acts as a **means for efficiently routing queries** and as a boundary for **multi-record** transactions.

Key Motivations

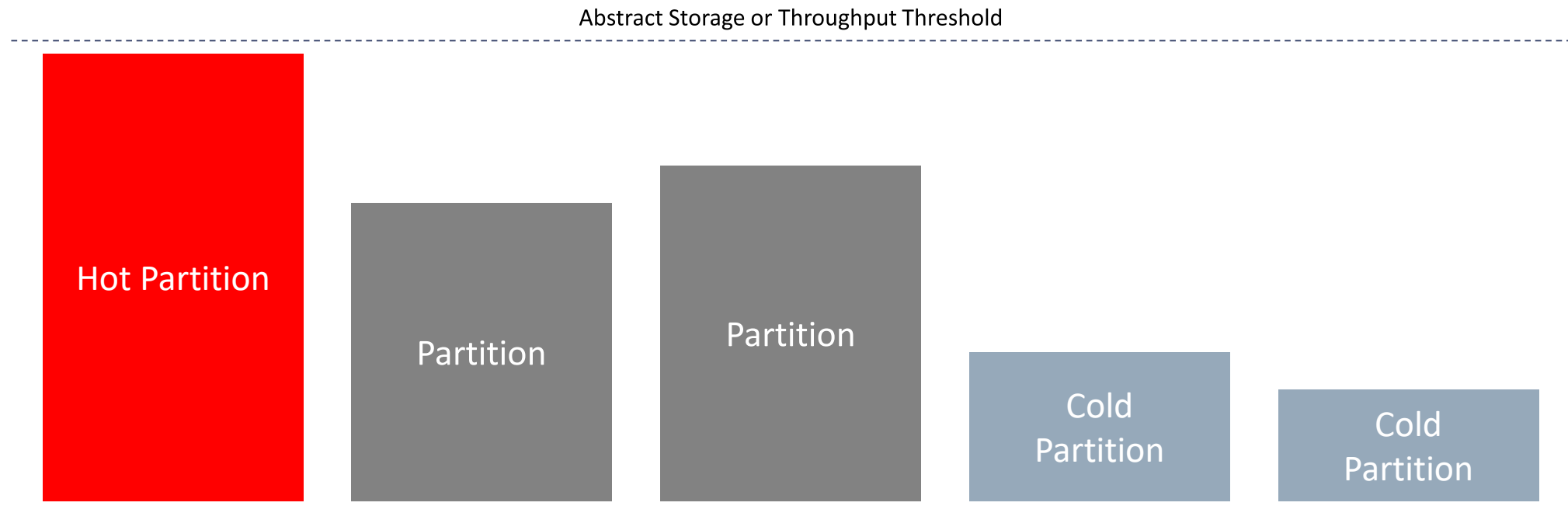
- Distribute Requests
- Distribute Storage
- Intelligently Route Queries for Efficiency



Hot/Cold Partitions

Partition usage can vary over time

Partitions that are approaching thresholds are referred to as hot. Partitions that are underutilized are referred to as cold.



Partition Design

Example Scenario

Contoso Connected Car is a vehicle telematics company. They are planning to store vehicle telemetry data from millions of vehicles every second in Azure Cosmos DB to power predictive maintenance, fleet management, and driver risk analysis.

The partition key we select will be the scope for multi-record transactions.

What are a few potential partition key choices?

- Vehicle Model
- Current Time
- Device Id
- Composite Key – Device ID + Current Time



Partition Key Choices

VEHICLE MODEL (e.g. Model A)

Most auto manufactures only have a couple dozen models. This will create a fixed number of logical partition key values; and is potentially the least granular option.

Depending how uniform sales are across various models – this introduces possibilities for hot partition keys on both storage and throughput.



CURRENT MONTH (e.g. 2018-04)

Auto manufacturers have transactions occurring throughout the year. This will create a more balanced distribution of storage across partition key values. However, most business transactions occur on recent data creating the possibility of a hot partition key for the current month on throughput.

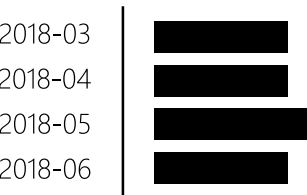
Storage Distribution



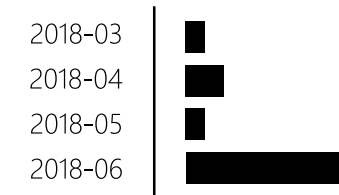
Throughput Distribution



Storage Distribution



Throughput Distribution



Partition Key Choices

DEVICE ID (e.g. Device123)

Each car would have a unique device ID. This creates a large number of partition key values and would have a significant amount of granularity.

Depending on how many transactions occur per vehicle, it is possible to a specific partition key that reaches the storage limit per partition key



COMPOSITE KEY (Device ID + Time)

This composite option increases the granularity of partition key values by combining the current month and a device ID. Specific partition key values have less of a risk of hitting storage limitations as they only relate to a single month of data for a specific vehicle.

Throughput in this example would be distributed more to logical partition key values for the current month.

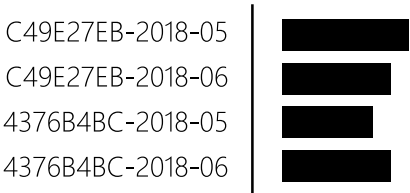
Storage Distribution



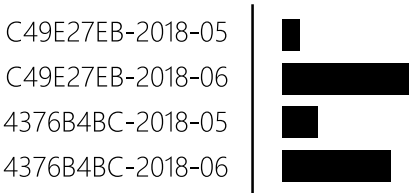
Throughput Distribution



Storage Distribution



Throughput Distribution



Partitioning Best Practices

- Distribute the overall request + storage volume
 - Avoid “hot” partition keys
- Select right Granularity for your partition keys
- Partition key is scope for multi-record transactions and routing queries
 - Queries can be intelligently routed via partition key
 - Omitting partition key on query requires fan-out

Partitioning Best Practices

Steps for Success

- Ballpark scale needs (size/throughput)
- Understand the workload
- # of reads/sec vs writes per sec
 - Use pareto principal (80/20 rule) to help optimize bulk of workload
 - For reads – understand top 3-5 queries (look for common filters)
 - For writes – understand transactional needs

General Tips

- Build a POC to strengthen your understanding of the workload and iterate (avoid analyses paralysis)
- Don't be afraid of having too many partition keys
 - Partitions keys are logical
 - More partition keys = more scalability

Partition Key storage limits

Containers support unlimited storage by dynamically allocating additional physical partitions

Storage for single partition key value (logical partition) is quota'ed to 10GB.

When a partition key reaches its provisioned storage limit, requests to create new resources will return a HTTP Status Code of 403 (Forbidden).

Azure Cosmos DB will automatically add partitions, and may also return a 403 if:

- An authorization token has expired
- A programmatic element (UDF, Stored Procedure, Trigger) has been flagged for repeated violations

Request Units (RUs) & Pricing

Billing Model

Two components: Consumed Storage + Provisioned Throughput

Unit	Price (for most Azure regions)
SSD Storage (per GB)	\$0.25 per month
Provisioned Throughput (single region writes)	\$0.008/hour per 100 RU/s
Provisioned Throughput (multi-region writes)	\$0.016/hour per 100 multi-region write RU/s

You are billed on consumed storage and provisioned throughput

Containers in a database can share throughput

What are Request Units (RUs)?

In Cosmos DB, you **provision** the expected capacity/performance

Expressed in **Request Units per second** (RU/s)

- Represents the "cost" of a request in terms of CPU, memory and I/O

Performance can be provisioned:

- at the database-level
- at the collection-level
- or both

You can change RU/s **programmatically** with API calls or...

NEW: Autopilot (preview) to scale based on usage

What are Request Units (RUs)?

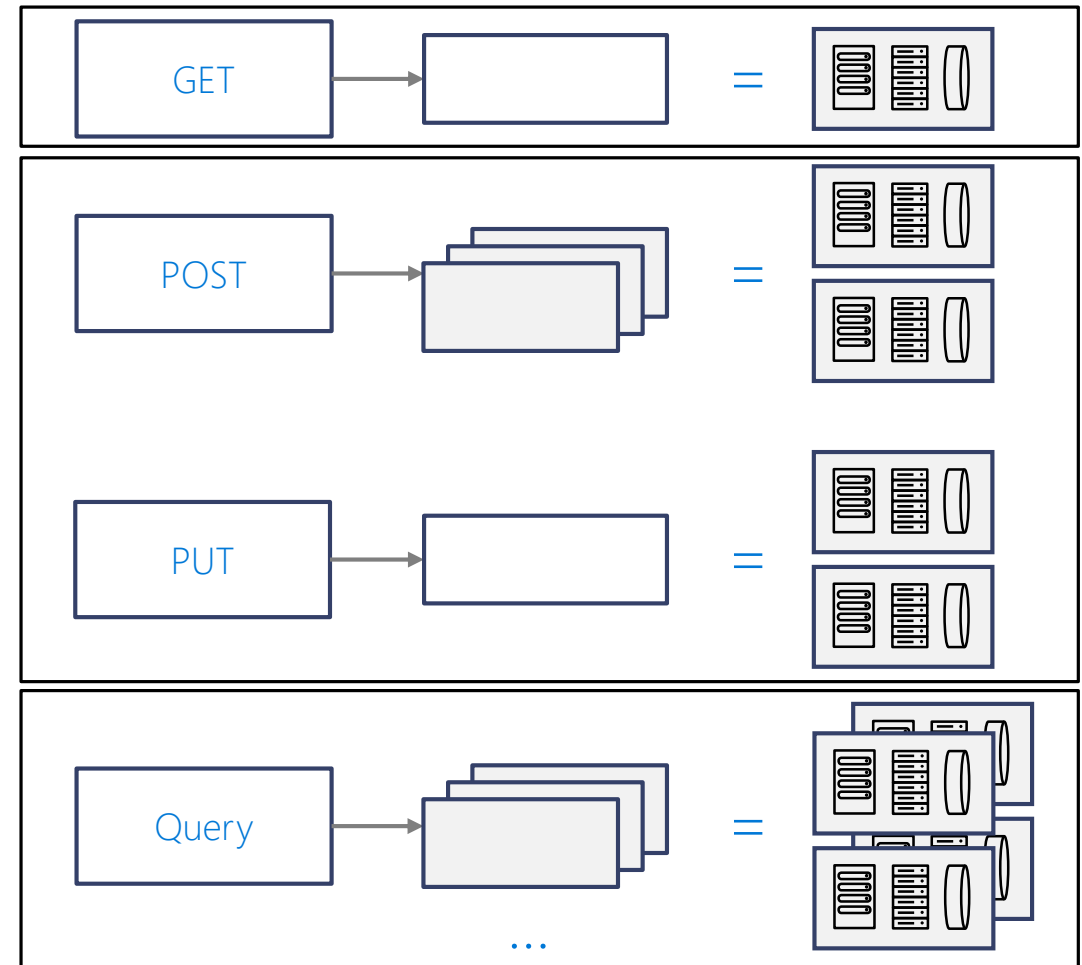
Each request consumes # of RU

1 RU = 1 read of 1 KB document

5 RU = 1 write of a 1KB document

Query: Depends on query & documents involved

(Indexing also affects RU cost)



Sample telemetry event document

```
{
  "partitionKey": "MP80QE3HPJPGAI4IG-2019-10",
  "entityType": "VehicleTelemetry",
  "vin": "MP80QE3HPJPGAI4IG",
  "state": "CT",
  "outsideTemperature": 84,
  "engineTemperature": 296,
  "speed": 20,
  "fuel": 5,
  "fuelRate": 12.99,
  "engineoil": 45,
  "tirepressure": 3,
  "odometer": 209150,
  "timestamp": "2019-10-20T04:46:52.5395469Z",
  "id": "87701651-fc64-40b5-bc4c-7be2b7f655e2"
}
```

Telemetry ingest... let's estimate the RU's we need

Contoso Logistics has 1000 vehicles

They send 1 document (write) every second

Each write requires 10 RUs

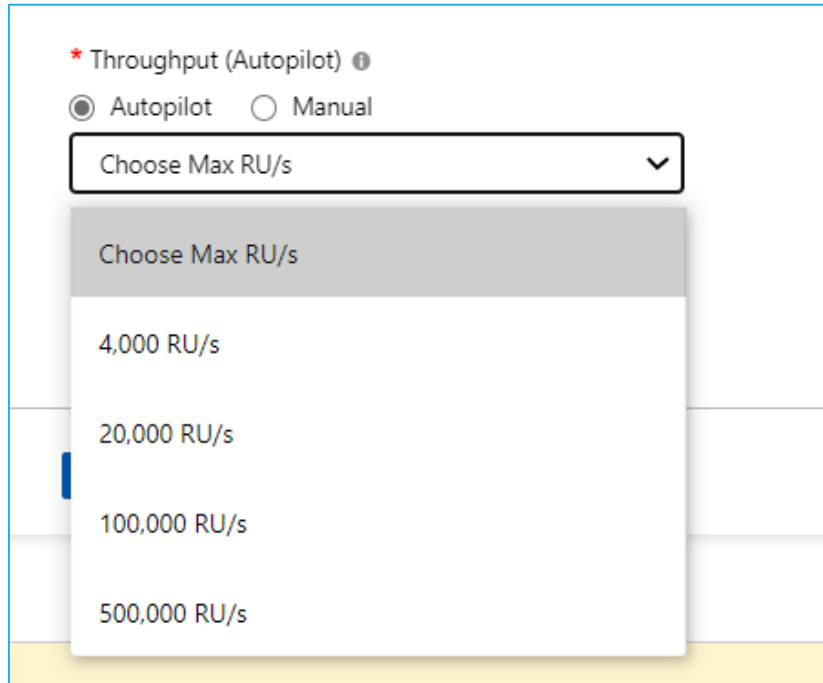
$$\text{Total RU/s} = \frac{1000 \text{ writes}}{\text{sec}} * \frac{\text{10}}{\text{write}} \text{ RU} = \boxed{\frac{10,000 \text{ RU}}{\text{sec}}}$$

AutoPilot vs manual provisioning

Autopilot (preview)

Good for bursty, unpredictable workloads

Scale between 10% and Max



* Throughput (Autopilot) ⓘ

☒ Autopilot ☐ Manual

Choose Max RU/s ▼

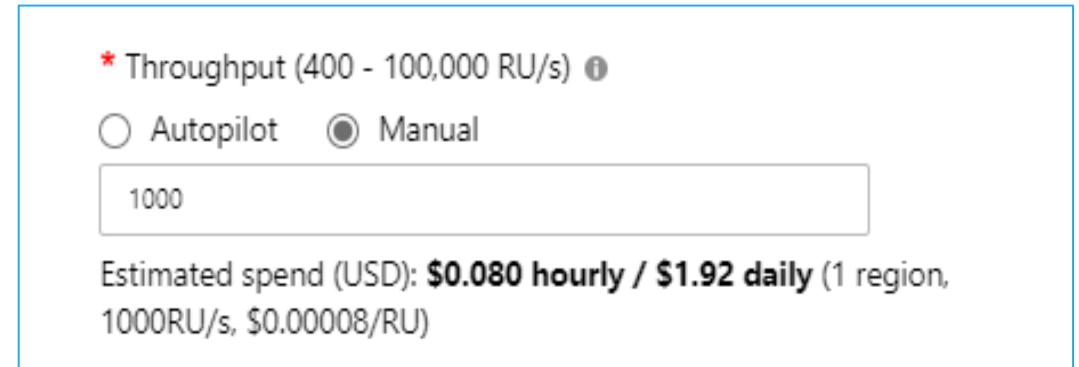
Choose Max RU/s

- 4,000 RU/s
- 20,000 RU/s
- 100,000 RU/s
- 500,000 RU/s

Manual

Good for predictable workloads

Change RU/s programmatically with Azure function



* Throughput (400 - 100,000 RU/s) ⓘ

☐ Autopilot ☒ Manual

1000

Estimated spend (USD): **\$0.080 hourly / \$1.92 daily** (1 region, 1000RU/s, \$0.00008/RU)

Measuring RU Charge

Analyze Query Complexity

The complexity of a query impacts how many Request Units are consumed for an operation. The number of predicates, nature of the predicates, number of system functions, and the number of index matches / query results all influence the cost of query operations.

Measure Query Cost

To measure the cost of any operation (create, update, or delete):

- Inspect the x-ms-request-charge header
- Inspect the RequestCharge property in ResourceResponse or FeedResponse in the SDK

Number of indexed terms impacts write RU charges

Every write operation will require the indexer to run. The more indexed terms you have, the more indexing will be directly influencing the RU charge.

You can optimize for this by fine-tuning your index policy to include only fields and/or paths certain to be used in queries.

Bulk of query RU charges is I/O

Query RU is directly proportional to the quantity of query results.

Request Unit Pricing Example

Storage Cost

Avg Record Size (KB)	1
Number of Records	100,000
Total Storage (GB)	100
Monthly Cost per GB	\$0.25
Expected Monthly Cost for Storage	\$25.00

Throughput Cost

Operation Type	Number of Requests per Second	Avg RU's per Request	RU's Needed
Create	100	5	500
Read	400	1	400

Total RU/sec	900
Monthly Cost per 100 RU/sec	\$6.00
Expected Monthly Cost for Throughput	\$54.00

Total Monthly Cost

$$\begin{aligned} \text{[Total Monthly Cost]} &= \text{[Monthly Cost for Storage]} + \text{[Monthly Cost for Throughput]} \\ &= \$25 \quad \quad \quad + \$54 \\ &= \$79 \text{ per month} \end{aligned}$$

* pricing may vary by region; for up-to-date pricing, see: <https://azure.microsoft.com/pricing/details/cosmos-db/>

Demo-2

Consistency Models

CAP theorem

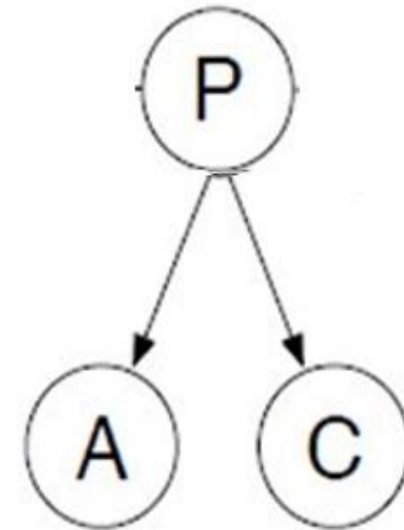
In case of network partitioning (P) in a distributed computer system, one has to choose between availability (A) and consistency (C).

It is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

Consistency: Every read receives the most recent write or an error

Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write

Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



PACELC theorem

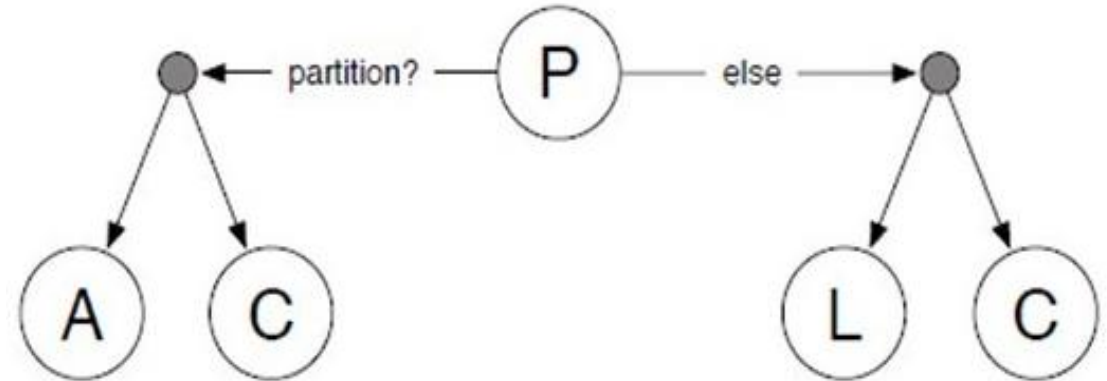
In case of network partitioning (P) in a distributed computer system, one has to choose between availability (A) and consistency (C) (as per the CAP theorem), but else (E), even when the system is running normally in the absence of partitions, one has to choose between latency (L) and consistency (C).

Consistency: Every read receives the most recent write or an error

Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write

Latency: Every request receives a delayed response

Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes



Five well-defined consistency models

CHOOSE THE BEST CONSISTENCY MODEL FOR YOUR APP

Five well-defined, consistency models

Overridable on a per-request basis

Provides control over performance-consistency tradeoffs, backed by comprehensive SLAs.

An intuitive programming model offering low latency and high availability for your planet-scale app.

CLEAR TRADEOFFS

- Latency
- Availability
- Throughput



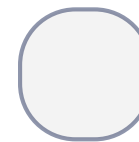
Strong



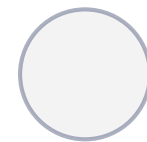
Bounded-staleness



Session



Consistent prefix



Eventual



Consistency models - breakdown

Consistency Level	Guarantees
Strong	Linearizability (once operation is complete, it will be visible to all)
Bounded Staleness	Consistent Prefix. Reads lag behind writes by at most k prefixes or t interval Similar properties to strong consistency (except within staleness window), while preserving 99.99% availability and low latency.
Session	Consistent Prefix. Within a session: monotonic reads, monotonic writes, read-your-writes, write-follows-reads Predictable consistency for a session, high read throughput + low latency
Consistent Prefix	Reads will never see out of order writes (no gaps).
Eventual	Potential for out of order reads. Lowest cost for reads of all consistency levels.

Strong Consistency

- Strong consistency offers a linearizability guarantee with the reads guaranteed to return the most recent version of an item.
- Strong consistency guarantees that a write is only visible after it is committed durably by the majority quorum of replicas.
- A client is always guaranteed to read the latest acknowledged write.
- The cost of a read operation (in terms of request units consumed) with strong consistency is higher than session and eventual, but the same as bounded staleness.
- Guarantees linearizability. Once an operation is complete, it will be visible to all readers in a strongly-consistent manner across replicas.

Eventual Consistency

- Eventual consistency guarantees that in absence of any further writes, the replicas within the group eventually converge.
- Eventual consistency is the weakest form of consistency where a client may get the values that are older than the ones it had seen before.
- Eventual consistency provides the weakest read consistency but offers the lowest latency for both reads and writes.
- The cost of a read operation (in terms of RUs consumed) with the eventual consistency level is the lowest of all the Azure Cosmos DB consistency levels.
- Replicas are eventually consistent with any operations. There is a potential for out-of-order reads. Lowest cost and highest performance for reads of all consistency levels.

Bounded Staleness Consistency

- Bounded staleness consistency guarantees that the reads may lag behind writes by at most K versions or prefixes of an item or t time-interval.
- Bounded staleness offers total global order except within the "staleness window." The monotonic read guarantees exist within a region both inside and outside the "staleness window."
- Bounded staleness provides a stronger consistency guarantee than session, consistent-prefix, or eventual consistency.
- The cost of a read operation (in terms of RUs consumed) with bounded staleness is higher than session and eventual consistency, but the same as strong consistency.
- Consistent prefix. Reads lag behind writes by at most k prefixes or t interval. Similar properties to strong consistency except within staleness window.

Bounded Staleness Consistency

Bounds are set server-side via the Azure portal

The screenshot shows the Azure portal interface for configuring the default consistency level of an Azure Cosmos DB account. The account name is 'artrejo-tables' and it is an 'Azure Cosmos DB account'.

Consistency Level Selection: The 'BOUNDED STALENESS' option is selected among the available consistency levels: STRONG, BOUNDED STALENESS, SESSION, CONSISTENT PREFIX, and EVENTUAL.

Informational Text: A message states: "Bounded staleness consistency is most frequently chosen by globally distributed applications expecting low write latencies but total global order guarantees. Unlike strong consistency which is scoped to a single region, you can choose bounded staleness consistency with any number of read regions (along with a write region)." It also notes: "Bounded staleness is great for applications featuring group collaboration and sharing, stock ticker, publish-subscribe/queueing etc." and provides a link for more information on Azure Cosmos DB consistency levels.

Maximum Lag (Operations): The value is set to 1000.

Maximum Lag (Time): The values are set as follows:

Days	Hours	Minutes	Seconds
0	0	0	5

Session Consistency

- Unlike the global consistency models offered by strong and bounded staleness consistency levels, session consistency is scoped to a client session.
- Session consistency is ideal for all scenarios where a device or user session is involved since it guarantees monotonic reads, monotonic writes, and read your own writes (RYW) guarantees.
- Session consistency provides predictable consistency for a session, and maximum read throughput while offering the lowest latency writes and reads.
- The cost of a read operation (in terms of RUs consumed) with session consistency level is less than strong and bounded staleness, but more than eventual consistency.
- Consistent prefix. Within a session, reads and writes are monotonic. This is referred to as “read-your-writes” and “write-follows-reads”. Predictable consistency for a session. High read throughput and low latency outside of session.

Session Consistency in code

Session is controlled using a "session token":

- Session tokens are automatically cached by the Client SDK
- Can be pulled out and used to override other requests (to preserve session between multiple clients)

```
string sessionToken;

using (DocumentClient client = new DocumentClient(new Uri(""), ""))
{
    ResourceResponse<Document> response = client.CreateDocumentAsync(
        collectionLink,
        new { id = "an id", value = "some value" }
    ).Result;
    sessionToken = response.SessionToken;
}
```

```
using (DocumentClient client = new DocumentClient(new Uri(""), ""))
{
    ResourceResponse<Document> read = client.ReadDocumentAsync(
        documentLink,
        new RequestOptions { SessionToken = sessionToken }
    ).Result;
}
```

Consistent Prefix Consistency

- Consistent prefix guarantees that in absence of any further writes, the replicas within the group eventually converge.
- Consistent prefix guarantees that reads never see out of order writes. If writes were performed in the order A, B, C, then a client sees either A, A,B, or A,B,C, but never out of order like A,C or B,A,C.
- Reads will never see out of order writes.

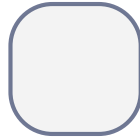
Relaxing Consistency in code



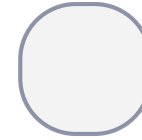
Strong



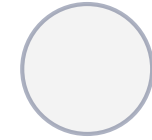
Bounded-staleness



Session



Consistent prefix



Eventual



Consistency can be relaxed on a per-request basis

```
client.ReadDocumentAsync(  
    documentLink,  
    new RequestOptions { ConsistencyLevel = ConsistencyLevel.Eventual }  
);
```

Conflict Resolution

Conflict Resolution

- Conflicts and conflict resolution policies are applicable if Azure Cosmos DB account is configured with multiple write regions.
- Conflict Types
 - Insert conflicts
 - Replace conflicts
 - Delete conflicts
- Conflict Resolution Policies
 - Last Write Wins (LWW)
 - Custom resolution using merge stored procedure
 - Manual resolution using conflicts feed

Conflict Resolution

```
FeedIterator<ConflictProperties> conflictFeed = container.Conflicts.GetConflictQueryIterator();
while (conflictFeed.HasMoreResults)
{
    FeedResponse<ConflictProperties> conflicts = await conflictFeed.ReadNextAsync();
    foreach (ConflictProperties conflict in conflicts)
    {
        // Read the conflicted content
        MyClass intendedChanges = container.Conflicts.ReadConflictContent<MyClass>(conflict);
        MyClass currentState = await container.Conflicts.ReadCurrentAsync<MyClass>(conflict, new PartitionKey(intendedChanges.MyPartitionKey));

        // Do manual merge among documents
        await container.ReplaceItemAsync<MyClass>(intendedChanges, intendedChanges.Id, new PartitionKey(intendedChanges.MyPartitionKey));

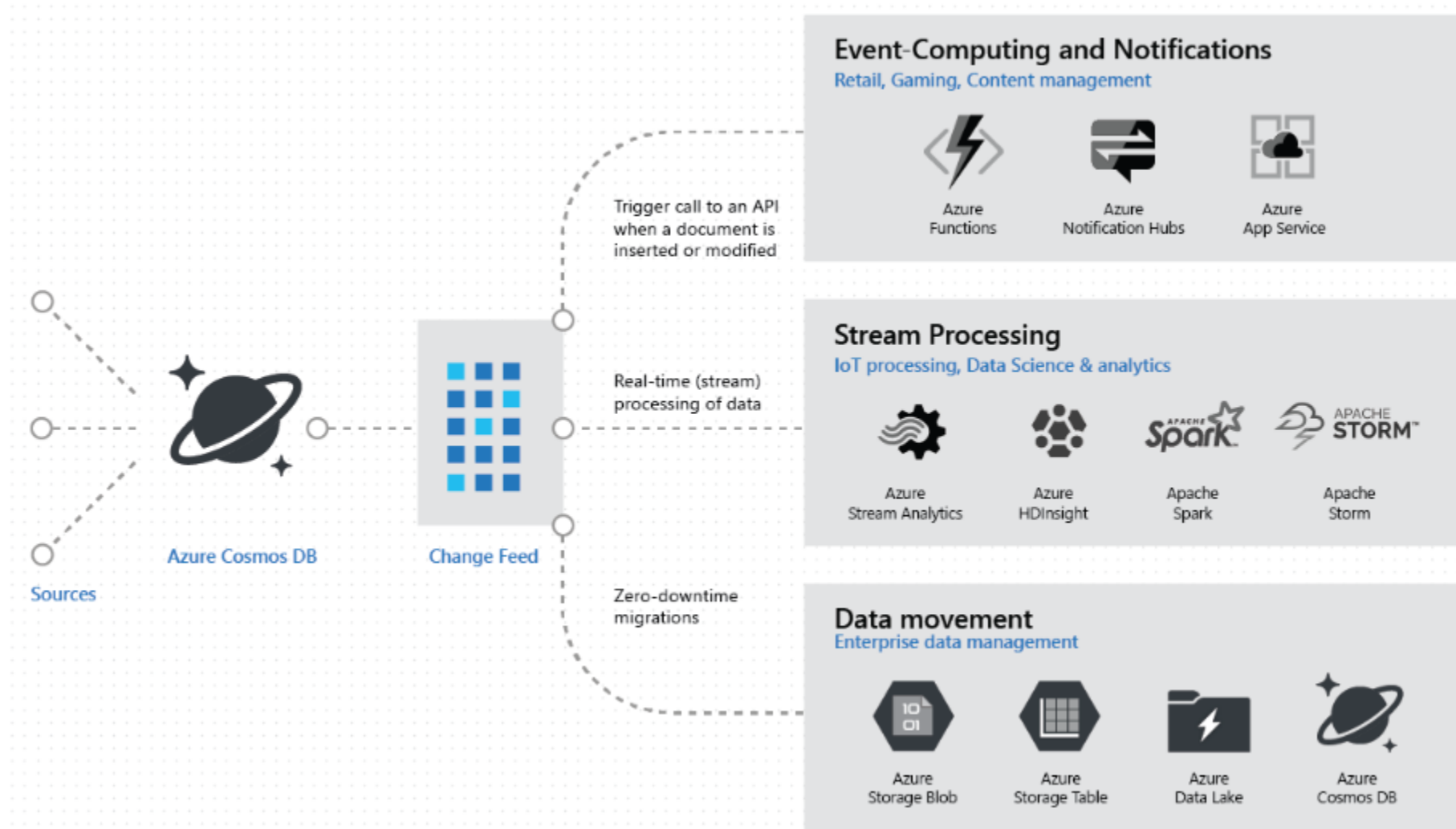
        // Delete the conflict
        await container.Conflicts.DeleteAsync(conflict, new PartitionKey(intendedChanges.MyPartitionKey));
    }
}
```

Change Feed

Change Feed

- Change feed listens to an Azure Cosmos container for any changes.
- Change feed outputs the sorted list of documents that were changed in the order in which they were modified.
- The changes are persisted, can be processed asynchronously and incrementally, and the output can be distributed across one or more consumers for parallel processing.
- Change feed is enabled by default for all Azure Cosmos accounts.
- The change feed includes inserts and update operations made to items within the container.
- Set a "soft-delete" flag within documents in place of deletes.

Change Feed



Troubleshooting

Diagnostic Logs

Cosmos DB automatically sends events to Azure Monitor, which you can view through:

- Log Analytics
- Event Hub
- Blob storage

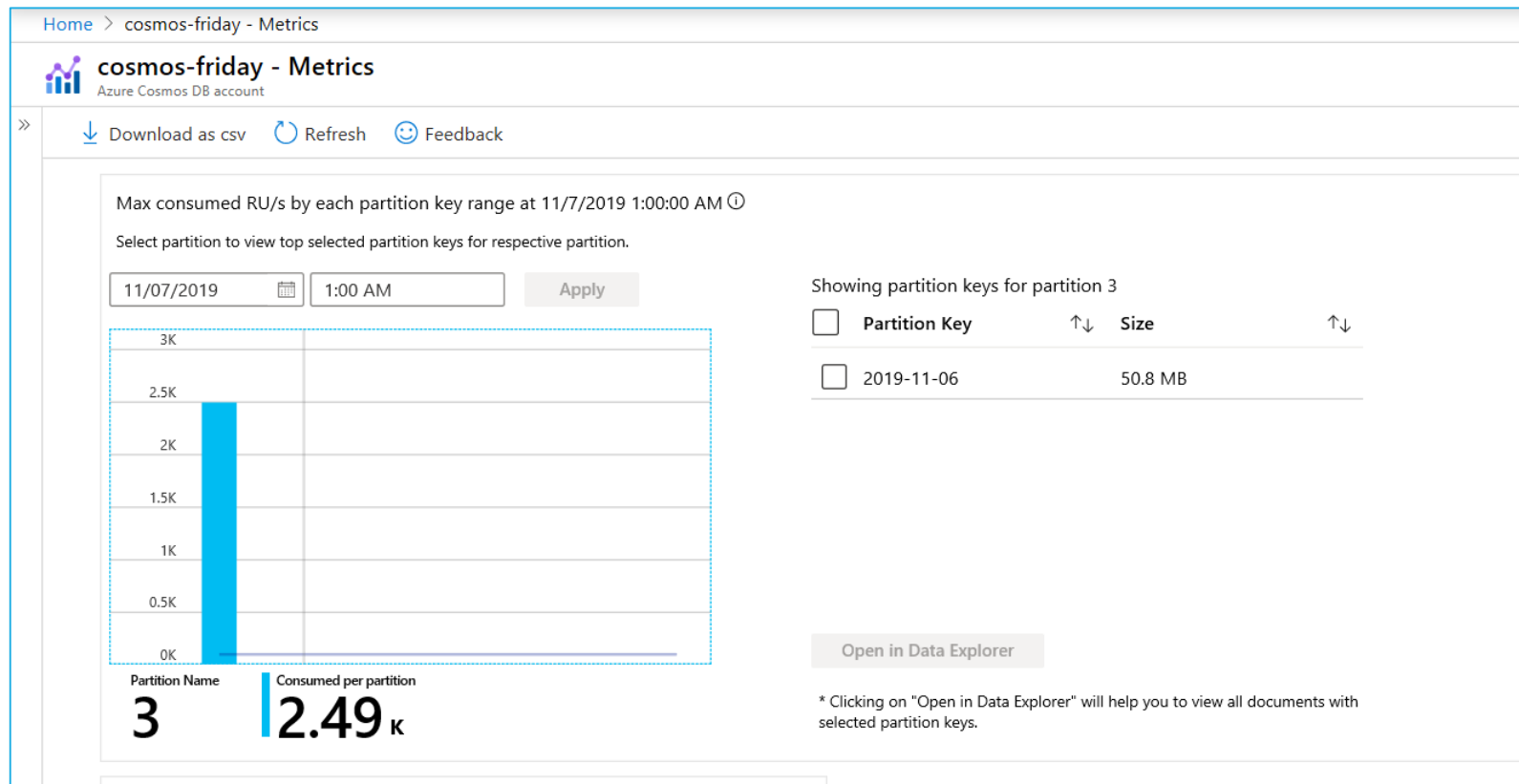
The screenshot shows the Azure Log Analytics interface for a workspace named 'dech-cosmos-oms'. The top bar includes the workspace name and a 'Log Analytics workspace' label. Below this, there's a search bar with 'HotPartition...' and a plus icon. The main area is divided into two sections: 'Schema' and 'Filter'. The 'Schema' section has a search bar with the text 'Filter by name or type...' and a magnifying glass icon. The 'Filter' section has a 'Collapse all' button. The main query editor displays a Kusto query:

```
// See all the 429s
AzureDiagnostics
| where TimeGenerated >= ago(8hr)
| where Category == "DataPlaneRequests"
| where statusCode_s == 429
| summarize count() by databaseName_s, collectionName_s
```

At the top of the query editor, there's a 'Run' button and a 'Time range : Last 24 hours' dropdown. On the right side, there are 'Save' and 'Copy' buttons.

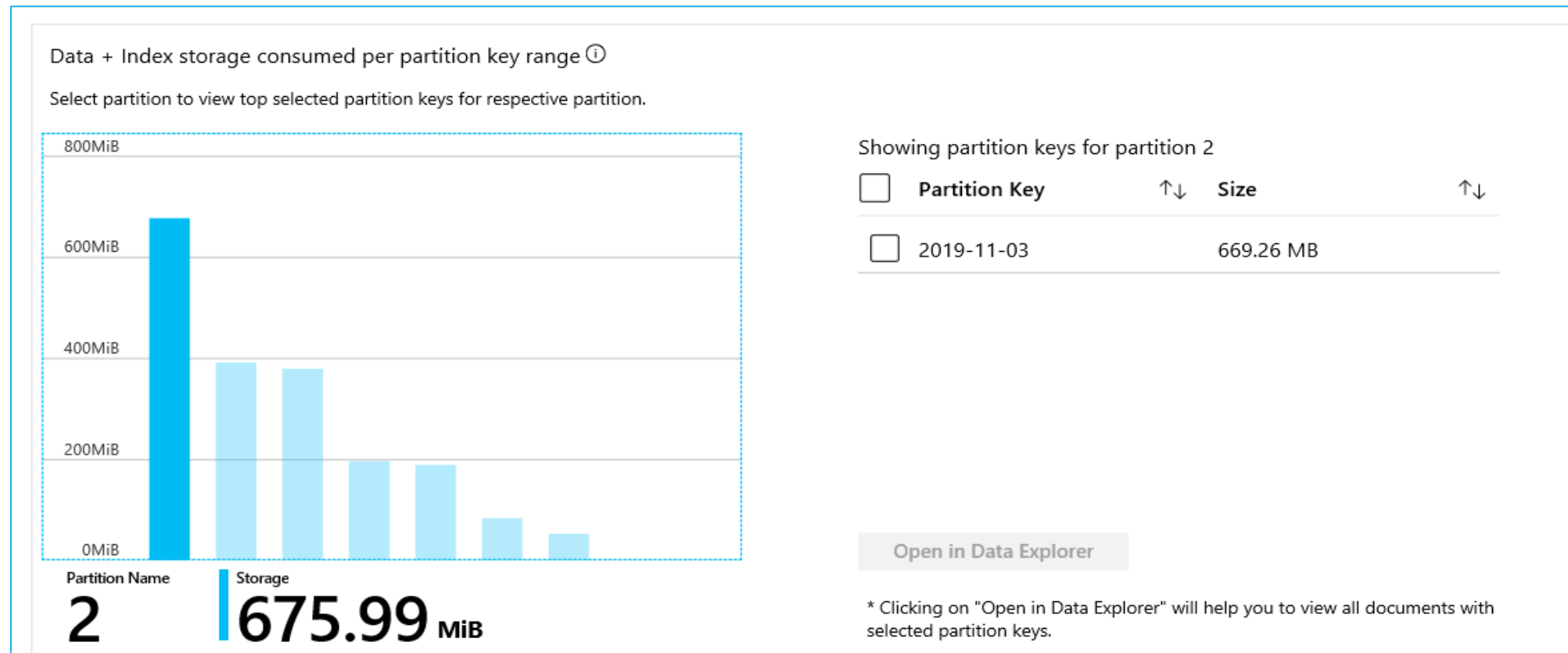
Use Metrics to identify hot partitions

Throughput metrics: view RU/s consumption at points in time



Monitor storage distribution in Metrics

Example of uneven storage distribution + partition key values on each partition



SDK – client-side tips and tricks

- Use singleton instance of CosmosClient
- Use direct mode (default for V3)
- Set ApplicationRegion (or PreferredLocations)
- Use point reads (key value lookups)
- Tune page size
- Tune max degree of parallelism
- Use Stream API – if applicable

Production readiness checklist

Add a second region for 99.999% High Availability


Single region: 99.99% availability (52.6 min / year)

2+ regions: Always 99.999% read availability

- Single-region write – 99.99% write availability
- Multi-region write – 99.999% write availability (5.26min / year)

Production readiness checklist

Add a second region for 99.999% High Availability

 **cosmos-friday - Replicate data globally**
Azure Cosmos DB account

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Quick start

Notifications

Data Explorer

Settings


Replicate data globally


Default consistency


Firewall and virtual networks


Private Endpoint Connections

CORS

 Save


 Discard

 Manual Failover

 Automatic Failover

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 (supported in selected regions and can only be configured when a new region is added). [+ Add region](#)

Regions	Reads Enabled	Writes Enabled	Availability Zo...	Action
East US 2	✓	✓		
West US 2	✓	✓	<input type="checkbox"/>	

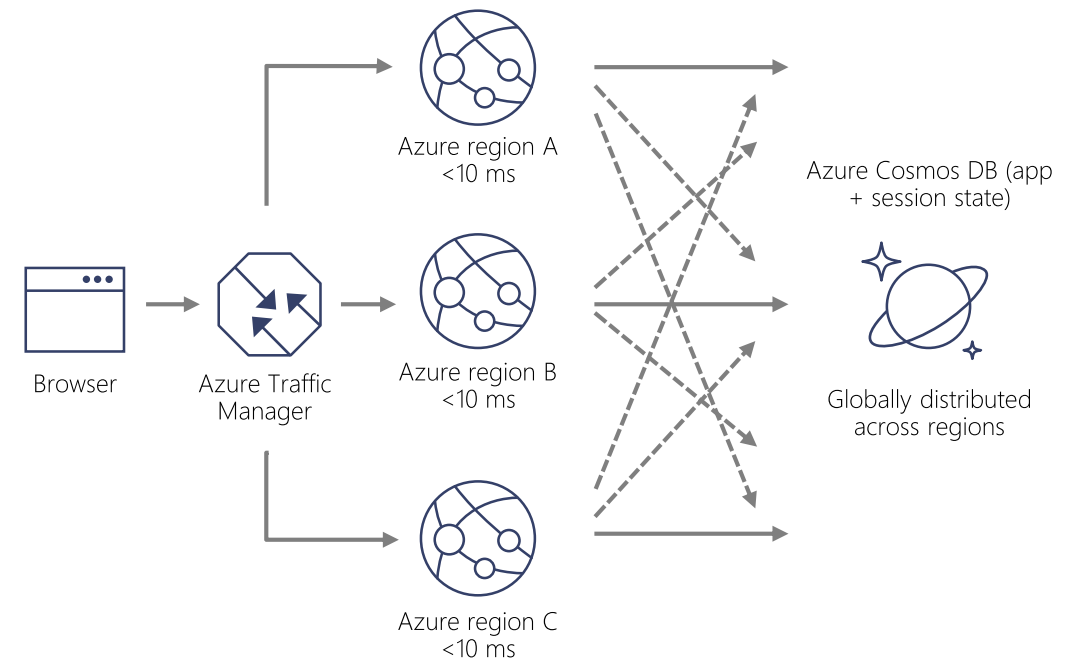
Demo-3

Use Cases

Data distributed and available globally

Put your data where your users are to give real-time access and uninterrupted service to customers anywhere in the world.

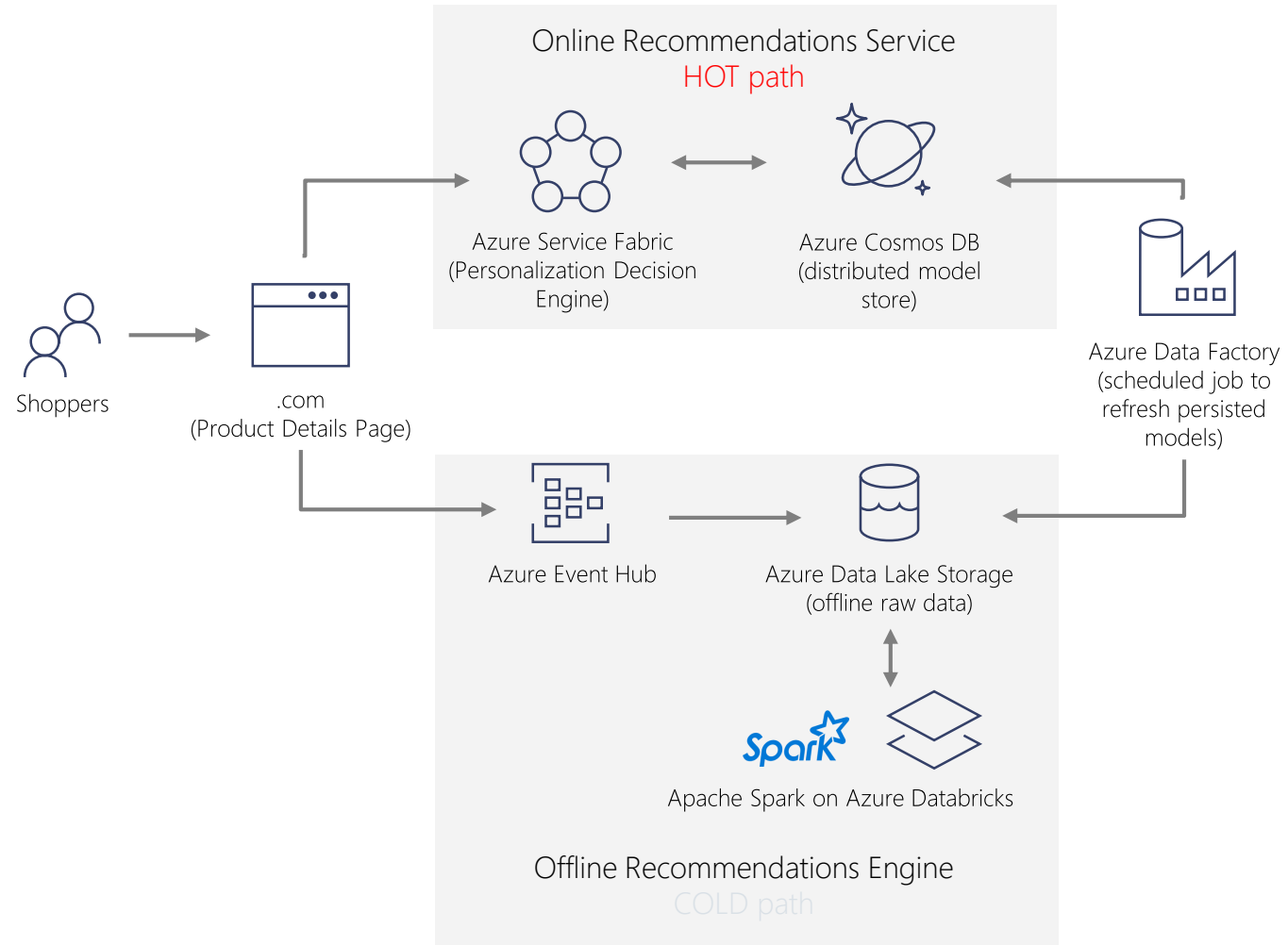
- Turnkey global data replication across all Azure regions
- Guaranteed low-latency experience for global users
- Resiliency for high availability and disaster recovery



Build real-time customer experience

Offer latency-sensitive applications with personalization, bidding, and fraud-detection.

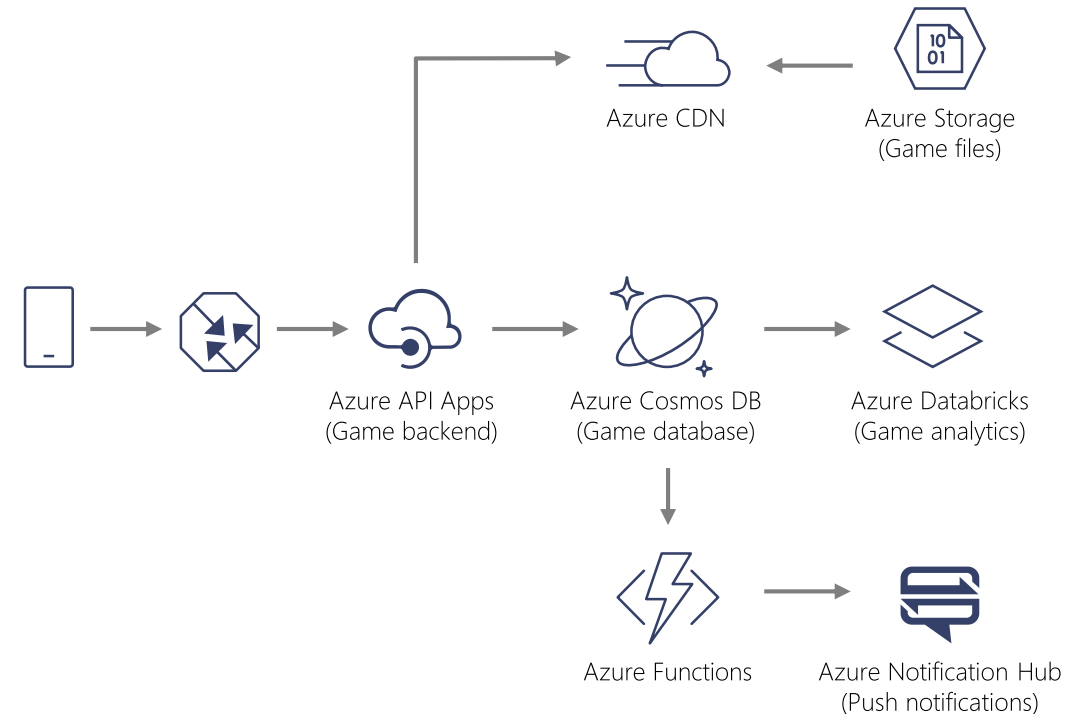
- Machine learning models generate real-time recommendations across product catalogues
- Product analysis in milliseconds
- Low-latency ensures high app performance worldwide
- Tunable consistency models for rapid insight



Gaming, IoT and eCommerce

Maintain service quality during high-traffic periods requiring massive scale and performance.

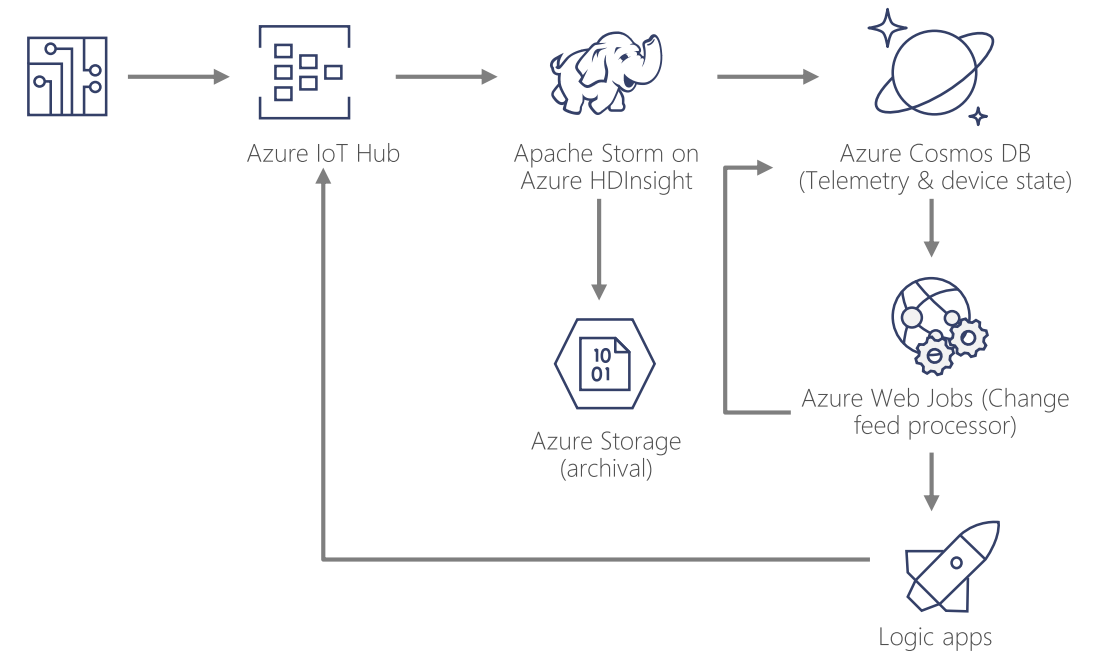
- Instant, elastic scaling handles traffic bursts
- Uninterrupted global user experience
- Low-latency data access and processing for large and changing user bases
- High availability across multiple data centers



Massive scale telemetry stores for IoT

Diverse and unpredictable IoT sensor workloads require a responsive data platform

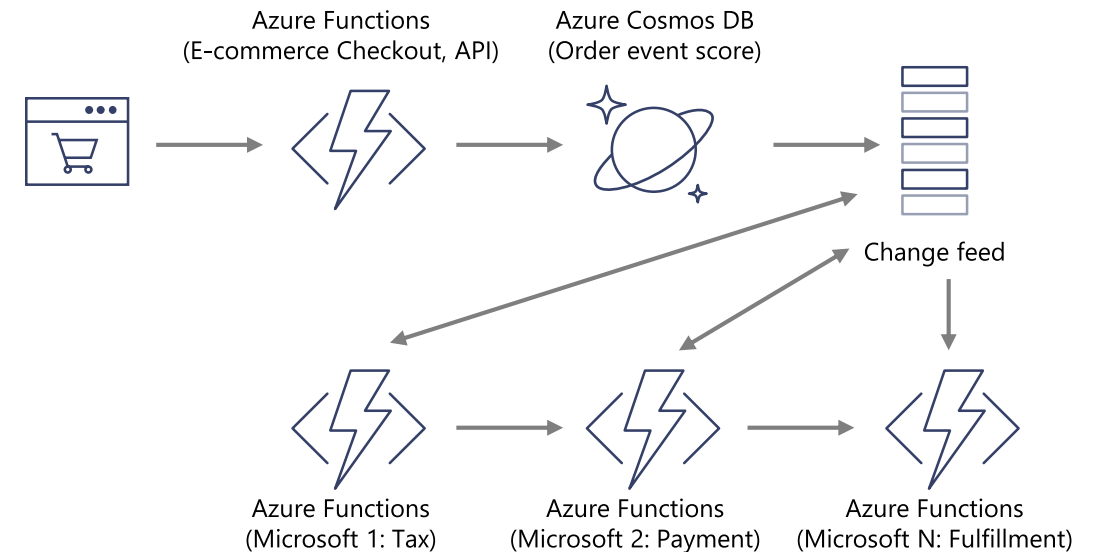
- Seamless handling of any data output or volume
- Data made available immediately, and indexed automatically
- High writes per second, with stable ingestion and query performance



Serverless Architecture

Experience decreased time-to-market, enhanced scalability, and freedom from framework management with event-driven micro-services.

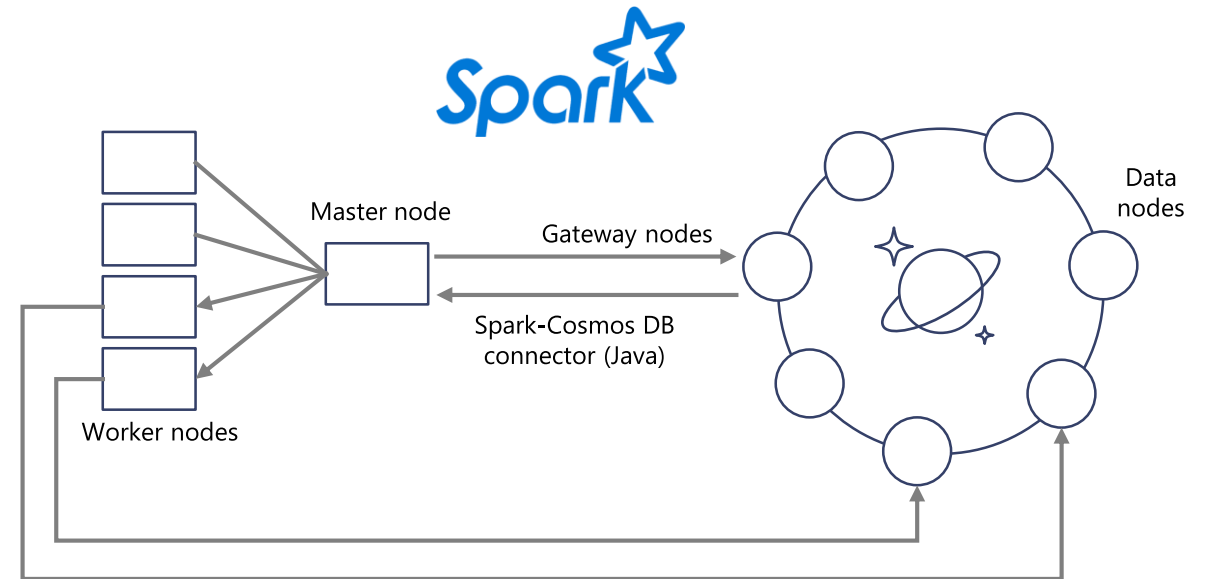
- Seamless handling of any data output or volume
- Data made available immediately, and indexed automatically
- High writes per second, with stable ingestion and query performance
- Real-time, resilient change feeds logged forever and always accessible
- Native integration with Azure Functions



Run Spark over operational data

Accelerate analysis of fast-changing, high-volume, global data.

- Real-time big data processing across any data model
- Machine learning at scale over globally-distributed data
- Speeds analytical queries with automatic indexing and push-down predicate filtering
- Native integration with Spark Connector



Questions

Global Azure Day 2020 - Omaha



The Global Azure Day 2020 – Omaha will take place on April 24, 2020 at Kiewit University.

Free full-day event hosted by community members for community.

Event is open to IT professionals, architects, developers, administrators and anyone who is interested in Azure.

Registration is now open - <https://globalazureday.omahaaug.com/>

If you are interested in sponsoring/volunteering, please reach out the organizers in-person, on slack channel or email at info@omahaaug.com

Thank you
