Designing CosmosDB workloads for throughput and cost efficiency

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Agenda

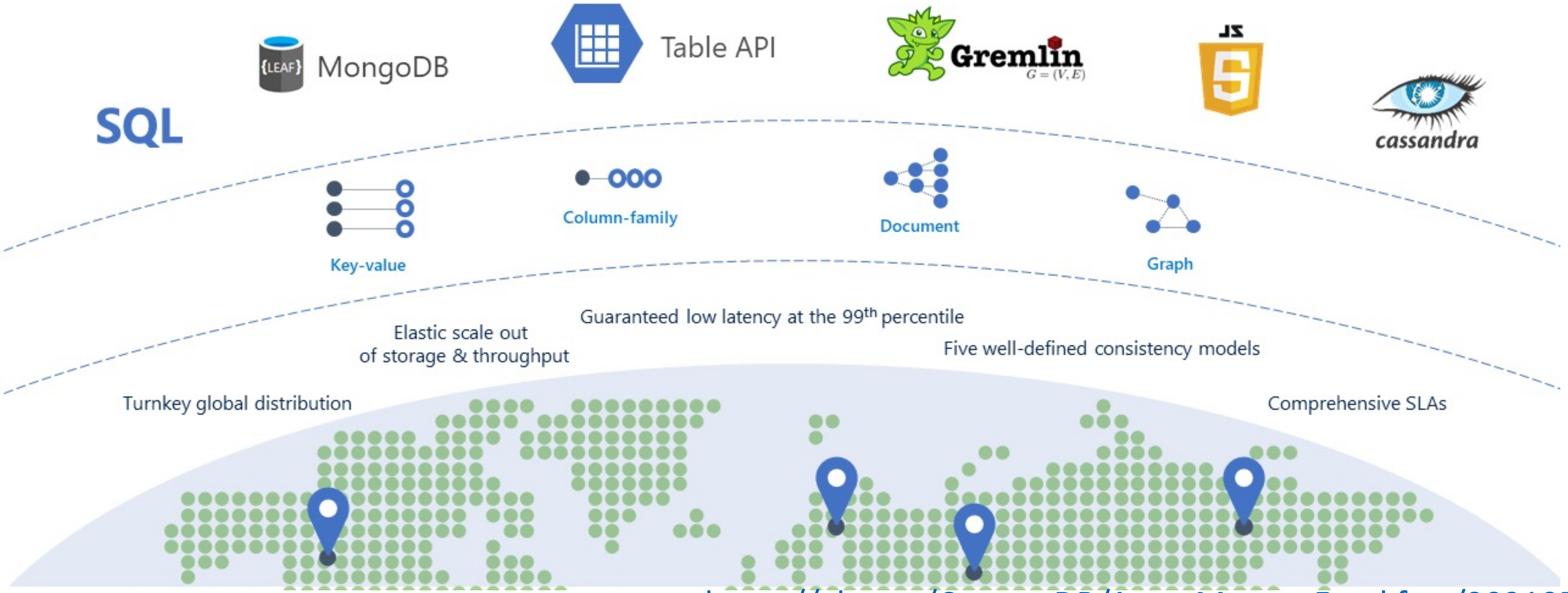
- Introduction
    - Cosmos DB Overview
    - NoSQL vs. RDBMS
    - Common use cases
- High availability (HA) and Disaster Recovery (DR)
    - Overview
    - Consistency models
    - Terminology
    - Trade-offs
- Provisioning model
    - Shared database vs. container specific provisioning
    - Auto Pilot vs. manually provisioned throughput vs. serverless
    - RI – Reserved Instances
- Cosmos DB APIs
    - Introduction
    - Sql/Gremlin/Table
    - Cassandra/Mongo
- Data modeling best practices
    - Partitioning
    - Indexing
    - Point-lookups vs. queries
    - Common patterns
    - Document size considerations
    - Change feed + materialized views
- Getting in touch…

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Introduction: Cosmos DB Overview

Azure Cosmos DB

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



Elastic scale out of storage & throughput

Guaranteed low latency at the 99th percentile

Five well-defined consistency models

Turnkey global distribution

Comprehensive SLAs

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Introduction: NoSQL vs. RDBMS

- CosmosDB – even the Core API (so called Sql Api) is a NoSQL (not only SQL) data service.
  - Schemaless
  - Optimized for horizontal scale
  - Very efficient for point-operations (key/value operation)
  - No support for distributed transactions - ACID only within the scope of logical partitions (not across containers or even multiple documents in the same container with different logical partition key values)
  - Supporting SQL-like query language …
    - … but query engine and indexing works different than in relational databases

- Trying to 1:1 migrate relational workloads or data models to Cosmos DB is a recipe for frustration – it is important to understand the fundamental optimizations for point-operations to come up with a data model that works well (I would claim this is true for all NoSQL databases – not just Cosmos DB)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Introduction: Common use cases

- **Retail/OLTP workloads**: Order management, Payment processing, pricing engines, recommendation engines, Inventory management, Product catalogs, user profiles/auth, real-time personalization
- **IoT:** Device telemetry, Device registry, digital twins, dependency management
- **Financial services**: Audit trail, tac forms, underwriting / risk analysis
- **Gaming**: Leaderboards, Social clans/guilds, Messaging
- **Within Microsoft**: Teams, Xbox (Licensing, Authentication, Order Management), AAD, Office 365 (Subscription management), ICM (Internal incident management tool)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

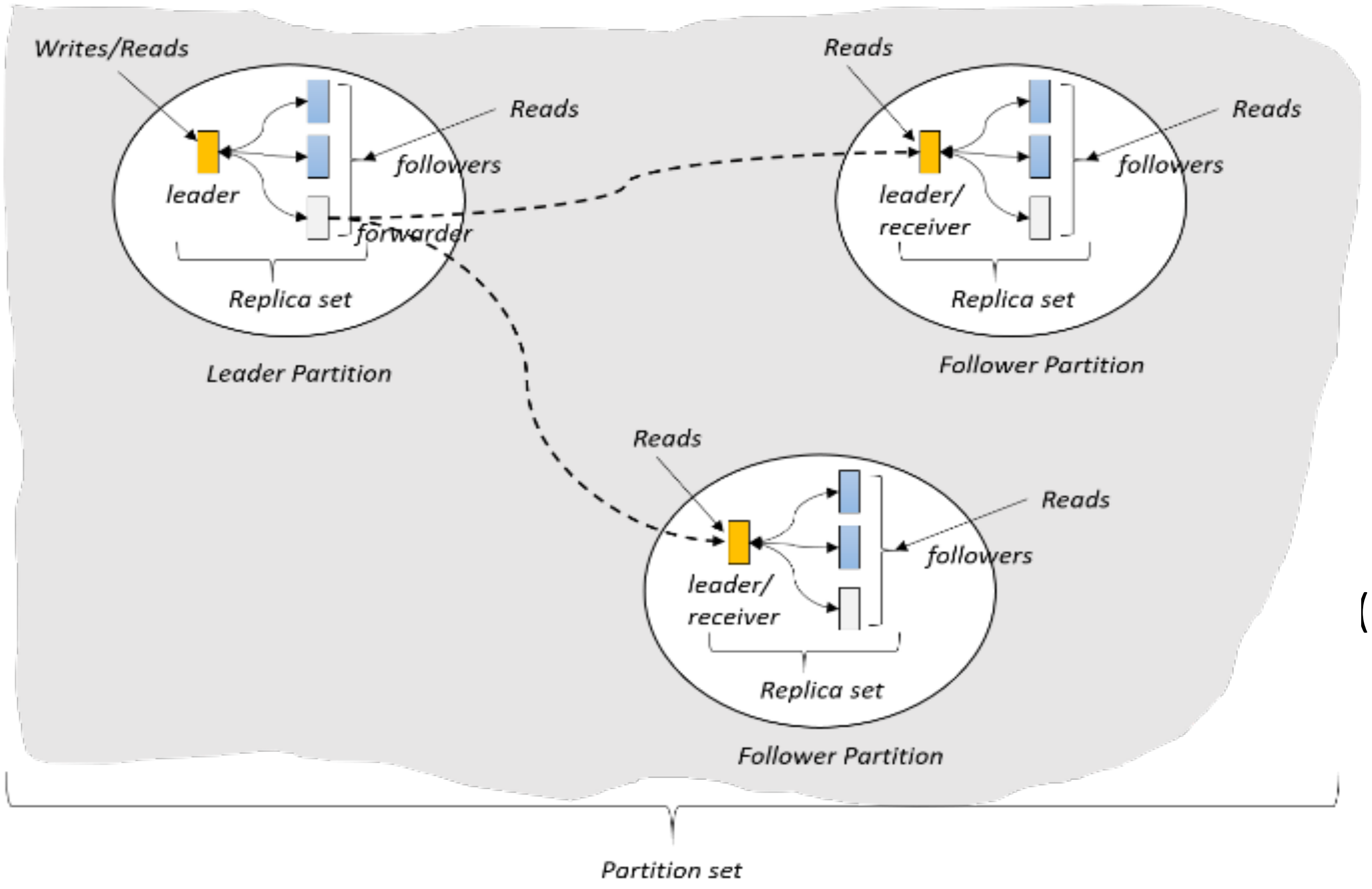
Agenda

- Introduction
  - Cosmos DB Overview
  - NoSQL vs. RDBMS
  - Common use cases
- High availability (HA) and Disaster Recovery (DR)
  - Overview
  - Consistency models
  - Terminology
  - Trade-offs
- Provisioning model
  - Shared database vs. container specific provisioning
  - Auto Pilot vs. manually provisioned throughput vs. serverless
  - RI – Reserved Instances
- Cosmos DB APIs
  - Introduction
  - Sql/Gremlin/Table
  - Cassandra/Mongo
- Data modeling best practices
  - Partitioning
  - Indexing
  - Point-lookups vs. queries
  - Common patterns
  - Document size considerations
  - Change feed + materialized views
- Getting in touch…

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

HA and DR: Overview

(West US)



(East US)

(North Europe)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

HA and DR: Consistency models

Consistency Level	Guarantees	Read cost factor	Comment
Strong	Linearizability (once operation is complete, it will be visible to all)	2	• Quorum read (normalized to cost factor of 2 even when reading from three replicas) • Only allowed in regions close to each other – no latency or availability SLA guarantees for writes – we can't control physics ☺
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)	2	• Quorum read (normalized to cost factor of 2 even when reading from three replicas) • In single write region identical with Strong – but replication happens asynchronously to other regions
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	1	• This is the default consistency model • Ideally mutable session token can flow end-to-end – otherwise only single CosmosDB client benefits from predictable consistency within session
Consistent Prefix	Reads will never see out of order writes (no gaps).	1	• My recommendation (over Session) when it is impossible to flow mutable session token end-to-end because it makes expectations clearer
Eventual	Potential for out of order reads. Lowest cost for reads of all consistency levels.	1	

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

HA and DR: Terminology

- **Single-Master vs. Multi-Master**: Indicates the replication model. Single-Master means one region is chosen a dedicated write region. All Writes have to be made in this region. (Manual) failover allows choosing another region as the write region. Multi-Master allows writes in all regions. This automatically means in theory conflicts can occur – a strategy for resolving conflicts needs to be chosen/implemented
- **Availability zone:** Availability Zones are physically separate locations within an Azure region. Each Availability Zone is made up of one or more datacenters equipped with independent power, cooling, and networking.

HA and DR: Trade-offs

#Regions	Replication Mode	Consistency Level	AZ redundant	Read Availavility	Write Availability	RPO	RTO	Cost factor
1	Single Master	*	No	99.99	99.99	< 240 Minutes	< 1 Week	1
1	Single Master	*	Yes	99.99	99.99	< 240 Minutes	< 1 Week	1.25
>1	Single Master	Session, Consistent Prefix, Eventual	No	99.999	99.99	< 15 minutes	< 15 minutes	#Regions
>1	Single Master	Bounded Staleness	No	99.999	99.99	K & T	< 15 minutes	#Regions
>1	Single Master	Session, Consistent Prefix, Eventual	Yes	99.999	99.99	< 15 minutes	< 15 minutes	1.25 * #Regions
>1	Single Master	Bounded Staleness	Yes	99.999	99.99	K & T	< 15 minutes	1.25 * #Regions
>1	Multi-Master	Session, Consistent Prefix, Eventual	*	99.999	99.999	< 15 minutes	0	2 * #Regions
>1	Multi-Master	Bounded Staleness	*	99.999	99.999	K & T	0	2 * #Regions
>1	Single-Master	Strong	No	99.99	n/a	0	< 15 minutes	#Regions
>1	Multi-Master	Strong	Yes	99.99	n/a	0	< 15 minutes	2 * #Regions

"K" is the number of versions of a given data item for which the reads lag behind the writes.

"T" is a given time interval.

“RPO" is the recovery point objective – the maximum targeted period of time in which data (transactions) are lost due to major IT incident

“RTO” is the recovery time objective – the targeted duration of time a service needs to be recovered

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Agenda

- Introduction
    - Cosmos DB Overview
    - NoSQL vs. RDBMS
    - Common use cases
- High availability (HA) and Disaster Recovery (DR)
    - Overview
    - Consistency models
    - Terminology
    - Trade-offs
- Provisioning model
    - Shared database vs. container specific provisioning
    - Auto Pilot vs. manually provisioned throughput vs. serverless
    - RI – Reserved Instances
- Cosmos DB APIs
    - Introduction
    - Sql/Gremlin/Table
    - Cassandra/Mongo
- Data modeling best practices
    - Partitioning
    - Indexing
    - Point-lookups vs. queries
    - Common patterns
    - Document size considerations
    - Change feed + materialized views
- Getting in touch…

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Provisioning model

- **Shared database vs. container specific**
  - Rule of thumb: Always use container specific
  - Only exception: Consider shared database model when you have multiple containers requiring < 400 RUs/s – but only for those containers. Use dedicated container throughput for containers with higher throughput requirements
  - Try to avoid enabling shared database throughput for >25 collections
- **AutoPilot vs. manually provisioning throughput**
  - AutoPilot dynamically manages throughput for you
  - No 429s except if throughput is above the configured AutoPilot Maximum
  - Cost-model similar. For every hour: Max(0.1 MaxAPThroughput, MaxConsumedThroughputDuringThatHour)
    - 100 RU/s single master, Manual → 0.08 USD/hour
    - 100 RU/s single master, AP → 0.12 USD/hour
    - 100 RU/s multi master, AP or Manual → 0.16 USD/hour
  - A tour of Azure Cosmos DB database operations models – YouTube
  - How to choose between manual and autoscale on Azure Cosmos DB | Microsoft Docs
- **Serverless (preview)**
  - Good for bursty, intermittent traffic – several restrictions – mostly for Dev/Tests environments → Consumption-based serverless offer in Azure Cosmos DB | Microsoft Docs

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Reserved Capacity Overview

Save money by pre-paying for Request Units.

Discounts are up to 65%

Larger reservations and reservations for longer time periods receive bigger discounts

NEW – Pay for the reservation up-front or with monthly payments

Azure Cosmos DB Reserved Capacity Marginal Discounts

	1 Year		3 Year	
Throughput	**Single Region Writes**	**Multiple Regions Writes**	**Single Region Writes**	**Multiple Regions Writes**
First 100K RU/s	20%	25%	30%	35%
Next 400K RU/s	25%	30%	35%	40%
Next 2.5M RU/s	30%	35%	45%	50%
Over 3M RU/s	45%	50%	60%	65%

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

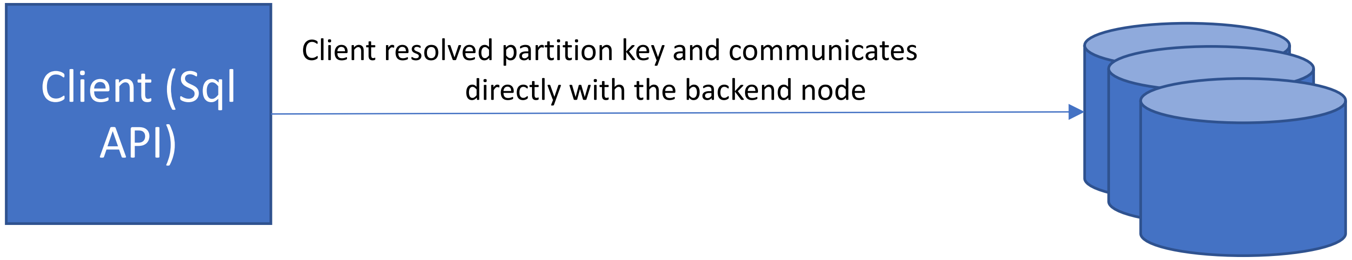
Agenda

- Introduction
  - Cosmos DB Overview
  - NoSQL vs. RDBMS
  - Common use cases
- High availability (HA) and Disaster Recovery (DR)
  - Overview
  - Consistency models
  - Terminology
  - Trade-offs
- Provisioning model
  - Shared database vs. container specific provisioning
  - Auto Pilot vs. manually provisioned throughput vs. serverless
  - RI – Reserved Instances
- Cosmos DB APIs
  - Introduction
  - Sql/Gremlin/Table
  - Cassandra/Mongo
- Data modeling best practices
  - Partitioning
  - Indexing
  - Point-lookups vs. queries
  - Common patterns
  - Document size considerations
  - Change feed + materialized views
- Getting in touch…

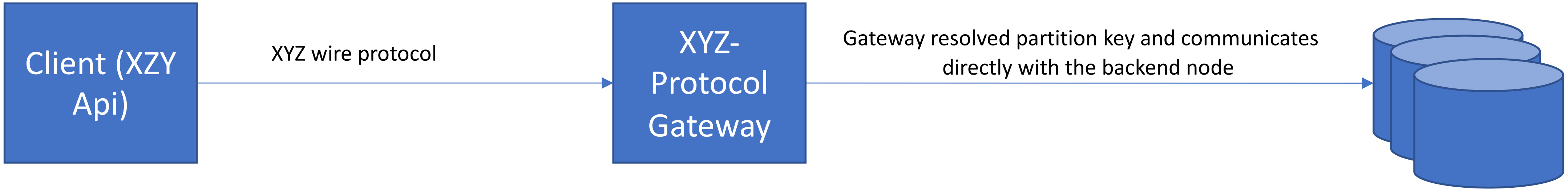
https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Cosmos DB APIs: Introduction

- The “Sql” Api should probably have been named the “Core” Api. It basically rules them all.


Client (Sql API)
Client resolved partition key and communicates directly with the backend node

- Database engine operates on atom-record-sequence (ARS) based type system
    - All data models are translated to ARS
- Overly-Simplified…


Client (XZY Api)
XYZ wire protocol
XYZ-Protocol Gateway
Gateway resolved partition key and communicates directly with the backend node

- Gateway compute layer as protocol head adds additional hop and RU charges

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Cosmos DB APIs: Sql/Gremlin/Table

- **Sql/Core**
  - Recommended as “default” for green-field.
  - Allows access to features early (change feed, bulk executor library, Spark connector etc.)
  - Usually lowest cost and best performance (one less hop)
  - Careful with “lift&shift” migrations from traditional relational Sql (Server) workloads – data model often needs to be optimized for horizontal scale (PK)
- **Gremlin/Graph**
  - Allows Multi-Api access. Meaning Gremlin account can be used via SQL/Core Api client as well. Edge and Vertex documents are stored as “normal” Sql/Core documents with few additional system properties
- **Table**
  - Currently only recommended for lift & shift of hot-storage workloads (heavy load on relatively small storage)
  - Automatic migration form legacy Azure Table Service offering when cold storage offer available in CosmosDB
  - No change-feed support

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Cosmos DB APIs: Cassandra/Mongo

- **Cassandra**
  - Schema required
  - Data stored compressed – until “hybrid row” support for Sql/Core Api and Mogo gets released good candidate for cold-storage scenarios
  - CCX (+ Managed Instance (preview)) allows hybrid-cloud migration. CosmosDB Cassandra API participates in Cassandra replication protocol – so data will be available in Cassandra replicas as well as in Cassandra Api in CosmosDB → Now in preview: Azure Managed Instance for Apache Cassandra | Azure Cosmos DB Blog (microsoft.com)
  - Note: the underlying storage engine is the same as for Sql/Core, so no Append-only storage. As a result partial updates aren’t as efficient as with pure Cassandra (yet)
  - Lift & shift usually works well – only few possible incompatibilities like LWT
  - Compatibility
- **Mongo**
  - Design for horizontal scale (shard key)!
  - Significant improvements to the aggregation pipeline coming
  - Currently support for protocol version 3.2, 3.6 and 4.0 – 4.2 targeted for later this year
  - Lift & shift sometimes problematic, Some feature gaps (aggregation pipeline, see supported features for different Mongo Api versions below, existing solutions often not designed for horizontal scale – can result in hot partitions/high cost)
  - Compatibility:
    - 4.0
    - 3.6
    - 3.2

Agenda

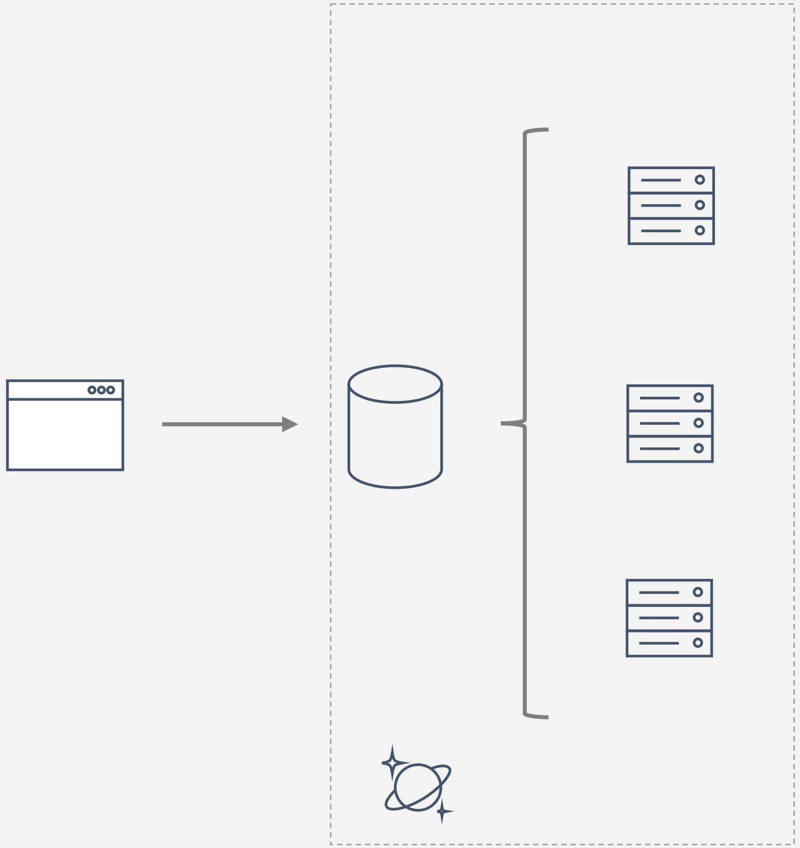
- Introduction
  - Cosmos DB Overview
  - NoSQL vs. RDBMS
  - Common use cases
- High availability (HA) and Disaster Recovery (DR)
  - Overview
  - Consistency models
  - Terminology
  - Trade-offs
- Provisioning model
  - Shared database vs. container specific provisioning
  - Auto Pilot vs. manually provisioned throughput vs. serverless
  - RI – Reserved Instances
- Cosmos DB APIs
  - Introduction
  - Sql/Gremlin/Table
  - Cassandra/Mongo
- Data modeling best practices
  - Partitioning
  - Indexing
  - Point-lookups vs. queries
  - Common patterns
  - Document size considerations
  - Change feed + materialized views
- Getting in touch…

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitioning: Overview

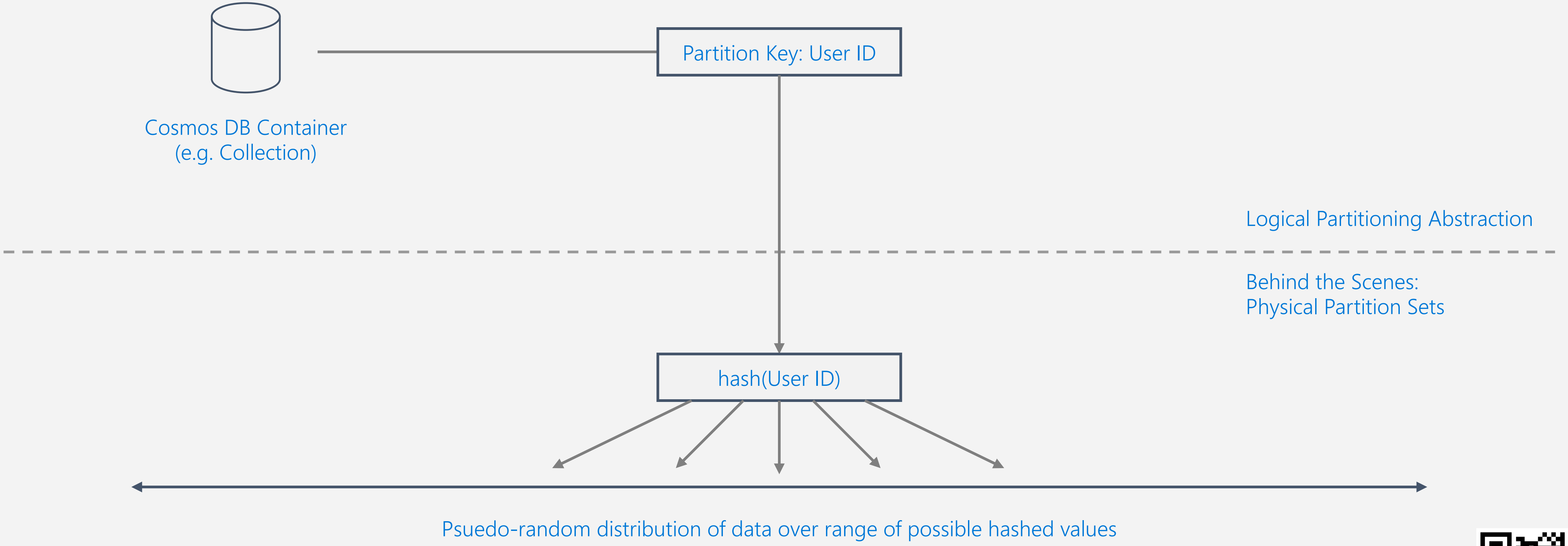
Leveraging Azure Cosmos DB to automatically scale your data across the globe

This module will reference partitioning in the context of all Azure Cosmos DB modules and APIs.



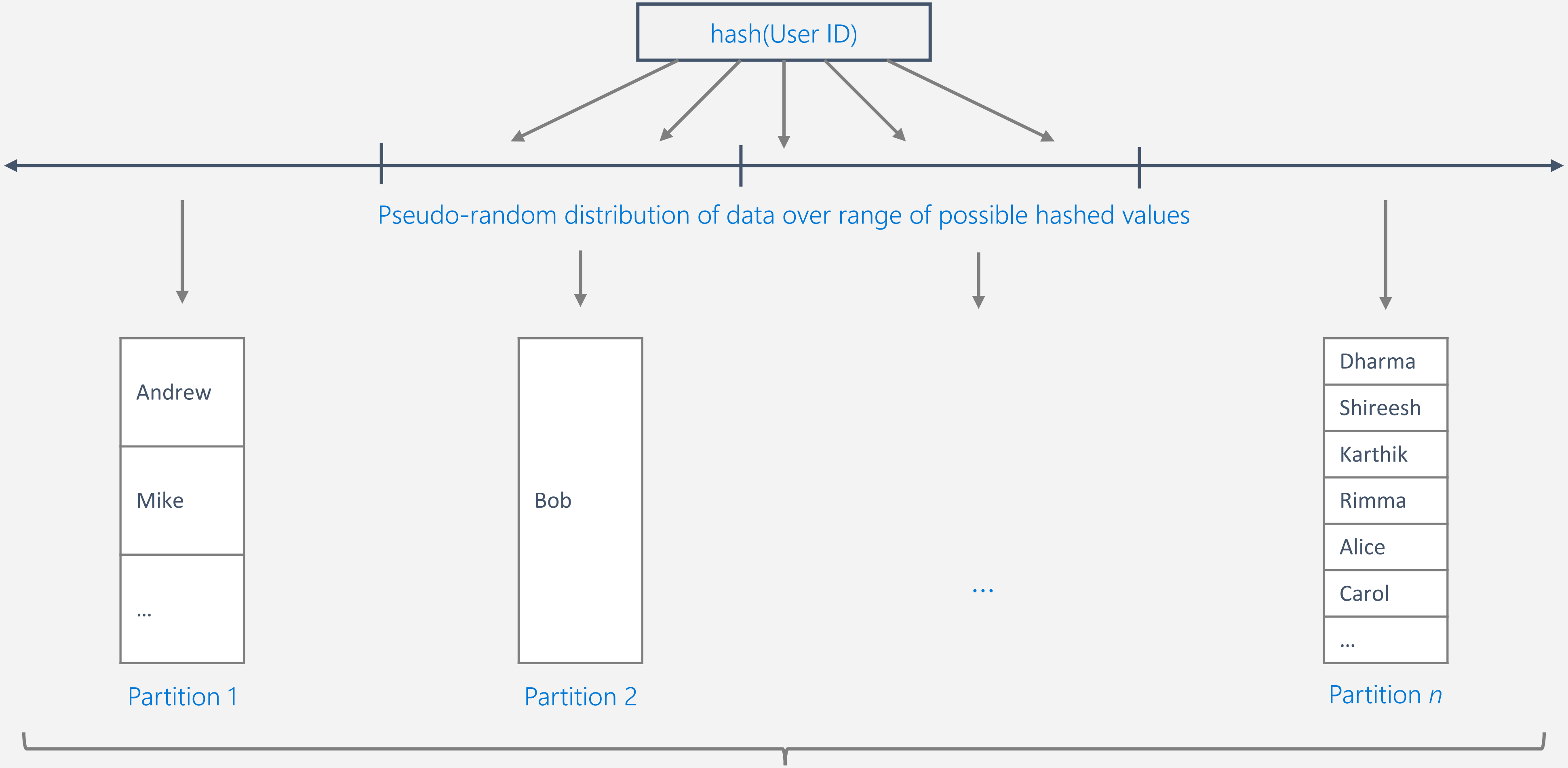
https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitioning: Partitions



https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitioning: Partitions



hash(User ID)

Pseudo-random distribution of data over range of possible hashed values

Andrew

Mike

…

Bob

…

Dharma

Shireesh

Karthik

Rimma

Alice

Carol

…

Partition 1

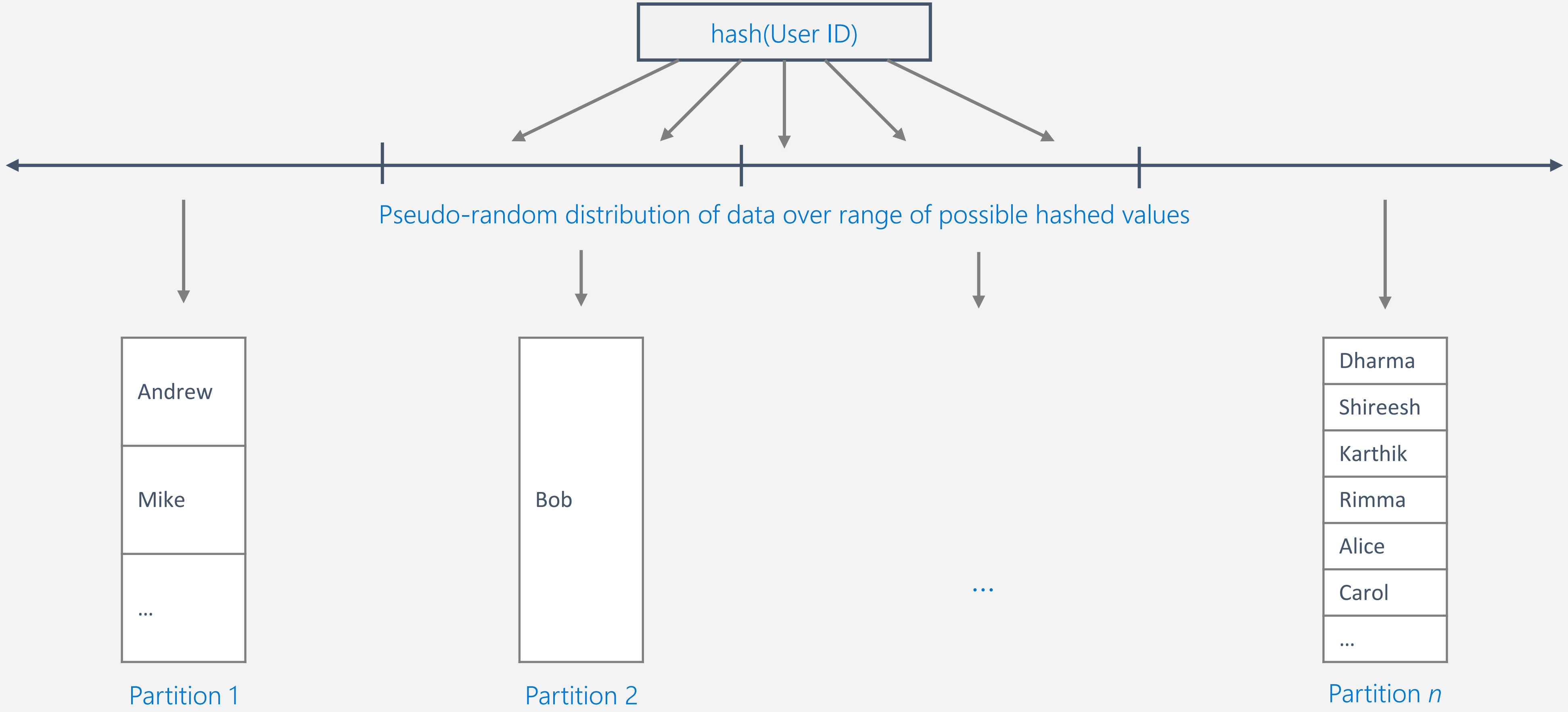
Partition 2

Partition *n*

Frugal # of Partitions based on actual storage and throughput needs
(yielding scalability with low total cost of ownership)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitioning: Partitions



hash(User ID)

Pseudo-random distribution of data over range of possible hashed values

Andrew
Mike
…

Bob

…

Dharma
Shireesh
Karthik
Rimma
Alice
Carol
…

Partition 1

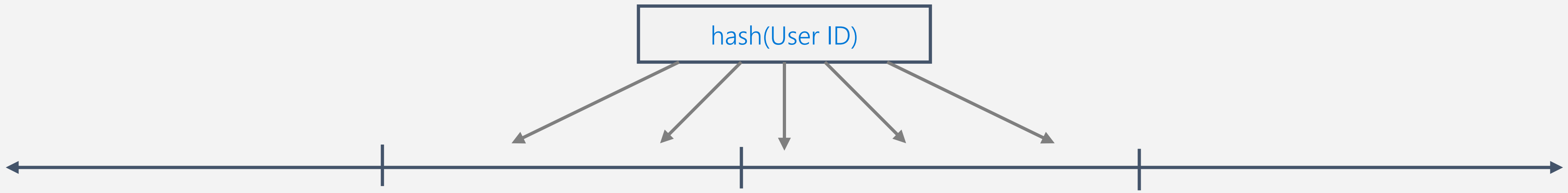
Partition 2

Partition *n*

What happens when partitions need to grow

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

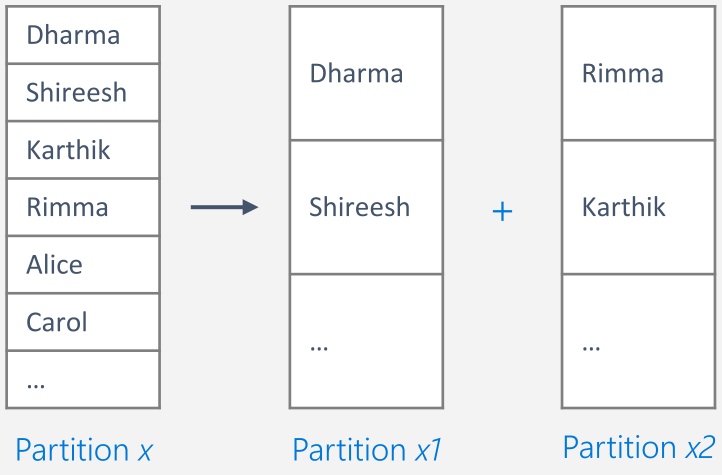
Partitioning: Partitions



Pseudo-random distribution of data over range of possible hashed values

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.



https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

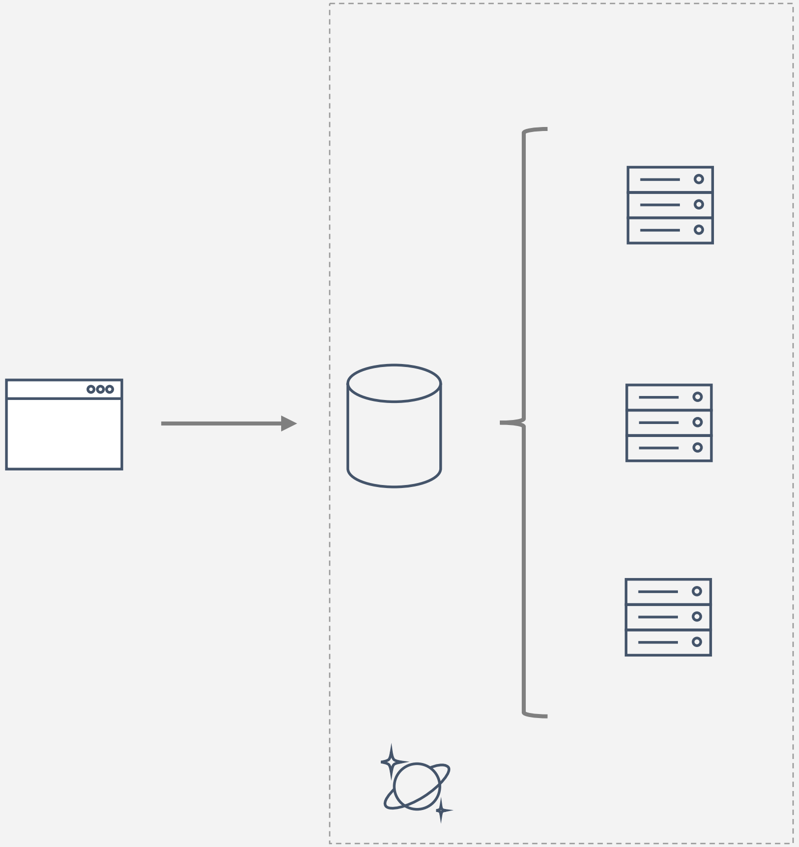
Partition Design

IMPORTANT TO SELECT THE “RIGHT” PARTITION KEY

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



https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitions

Best Practices: Design Goals for Choosing a Good Partition Key

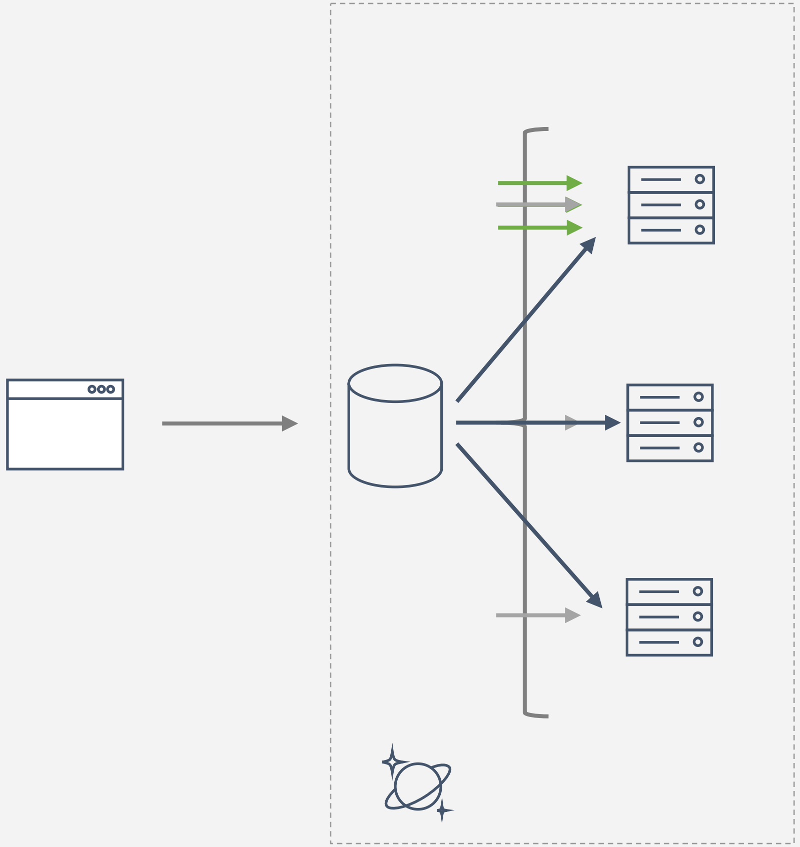
- Distribute the overall request + storage volume
  - Avoid “hot” partition keys
- Partition Key is scope for multi-record transactions and routing queries
  - Queries can be intelligently routed via partition key
  - Omitting partition key on query requires fan-out

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



https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partition Key storage limits

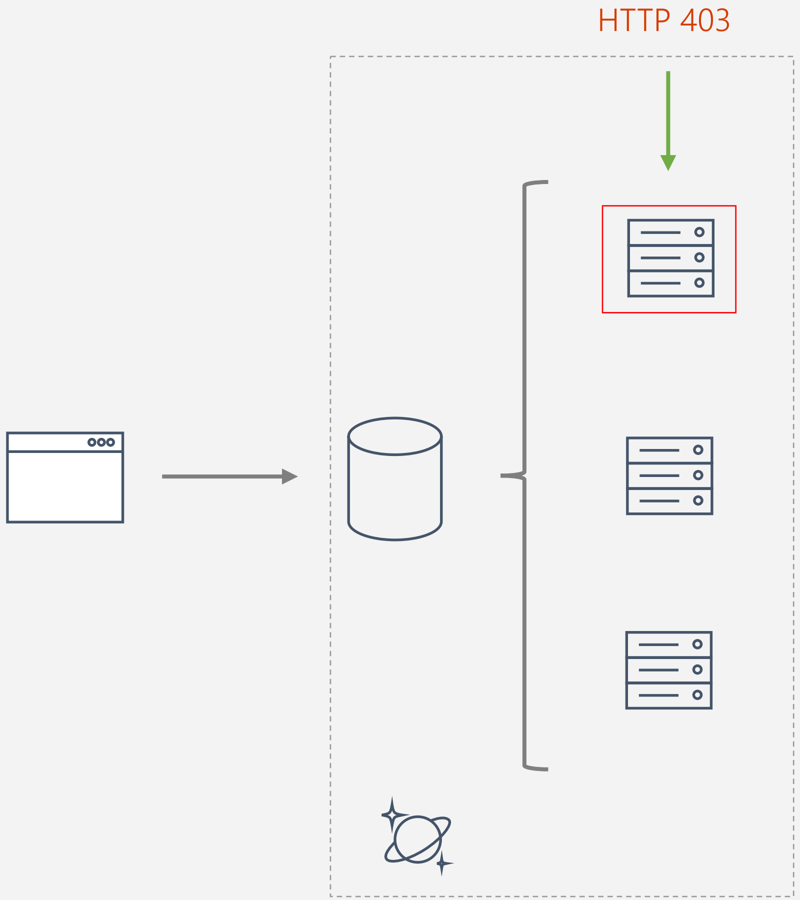
Containers support unlimited storage by dynamically allocating additional physical partitions

Storage for single partition key value (logical partition) is quota'ed to 20GB.

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



https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

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

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

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.

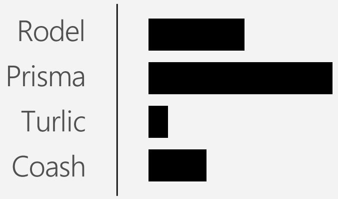
Depending how uniform sales are across various models – this introduces possibilities for hot partition keys on both storage and throughput.

CURRENT MONTH (e.g. 2018-04)

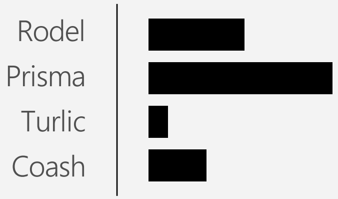
Auto manufacturers have transactions occurring throughout the year. This will create a more balanced distribution of storage across partition key values.

However, most business transactions occur on recent data creating the possibility of a hot partition key for the current month on throughput.

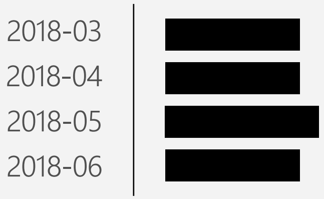
Storage Distribution



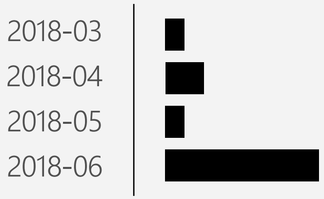
Throughput Distribution



Storage Distribution



Throughput Distribution



Example – Contoso Connected Car

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partition Key Choices

DEVICE ID (e.g. Device123)

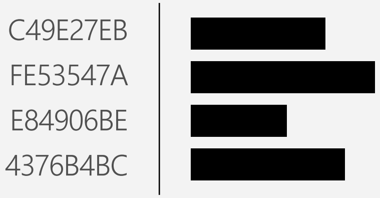
Each car would have a unique device ID. This creates a large number of partition key values and would have a significant amount of granularity.

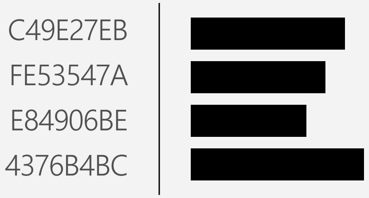
Depending on how many transactions occur per vehicle, it is possible to a specific partition key that reaches the storage limit per partition key

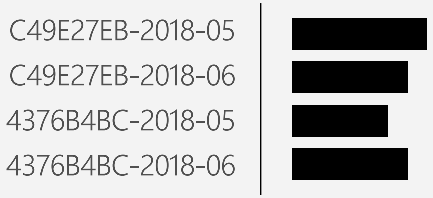
COMPOSITE KEY (Device ID + Time)

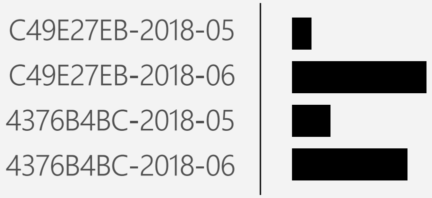
This composite option increases the granularity of partition key values by combining the current month and a device ID. Specific partition key values have less of a risk of hitting storage limitations as they only relate to a single month of data for a specific vehicle.

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

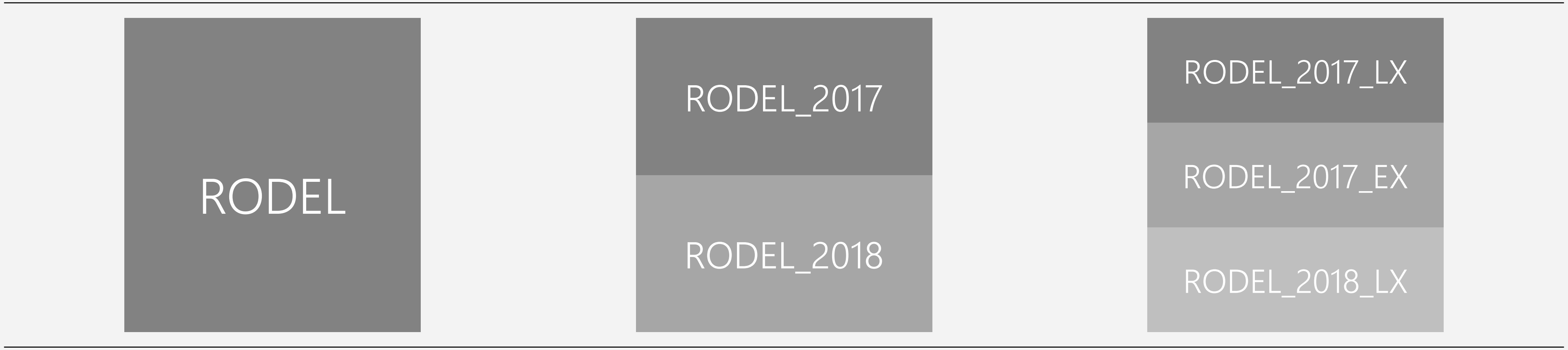
Example – Contoso Connected Car

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partition Granularity

SELECT THE “RIGHT” LEVEL OF GRANULARITY FOR YOUR PARTITIONS

Partitions should be based on your most often occurring query and transactional needs. The goal is to **maximize granularity** and **minimize cross-partition requests**.



RODEL

RODEL_2017

RODEL_2018

RODEL_2017_LX

RODEL_2017_EX

RODEL_2018_LX

Don’t be afraid to have more partitions!

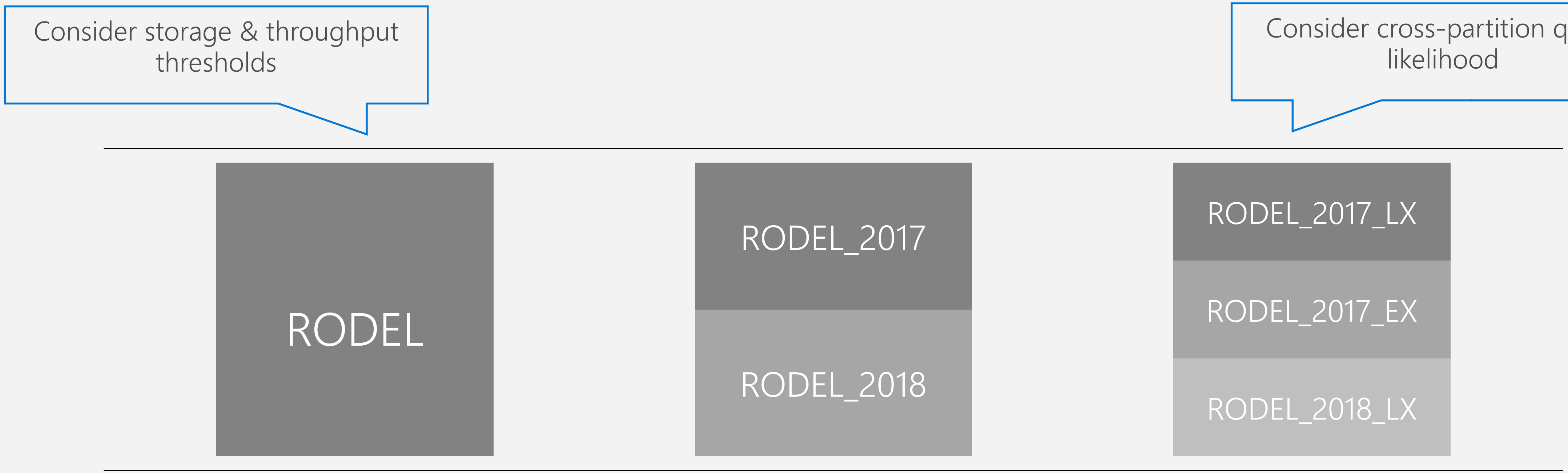
More partition keys = More scalability

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Example – Contoso Connected Car

Partition Granularity

SELECT THE “RIGHT” LEVEL OF GRANULARITY FOR YOUR PARTITIONS



Consider storage & throughput thresholds

Consider cross-partition query likelihood

Don’t be afraid to have more partitions!

More partition keys = More scalability

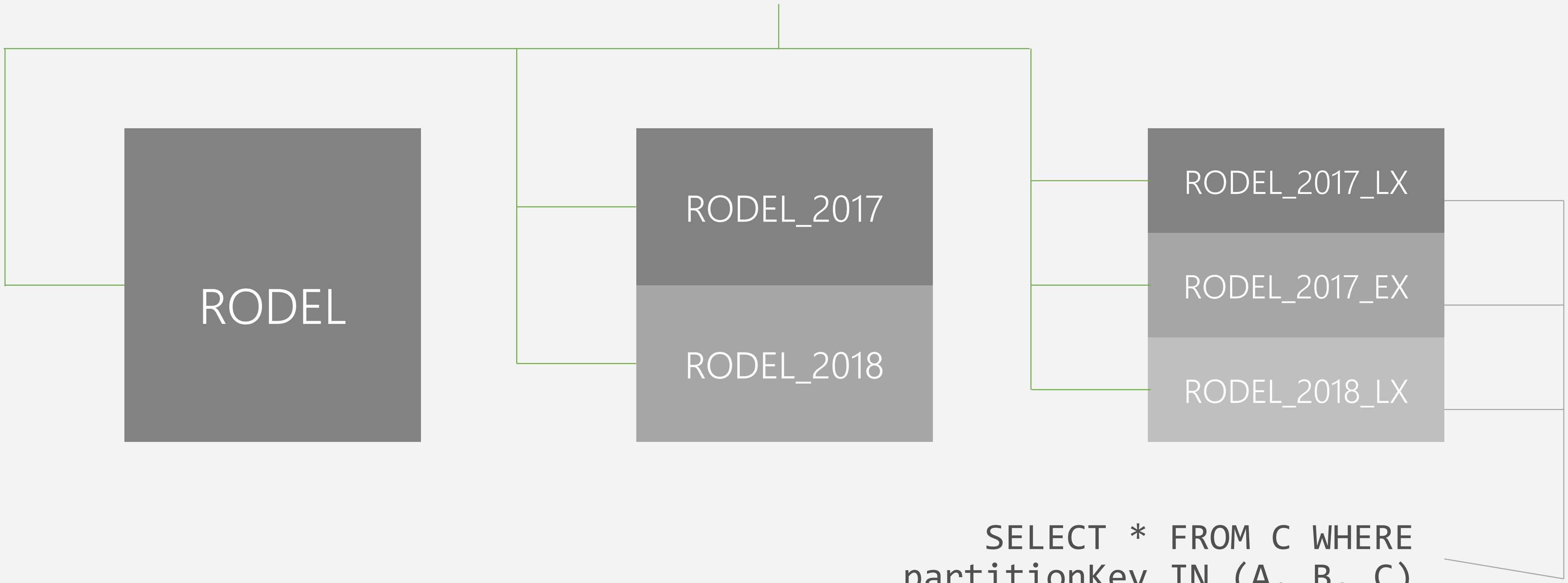
Example – Contoso Connected Car

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partition Granularity

A CROSS-PARTITION QUERY IS NOT ALWAYS A BLIND FAN OUT QUERY

```
SELECT * FROM C
```



```
SELECT * FROM C WHERE
partitionKey IN (A, B, C)
```

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Partitioning: Sub-Partitioning

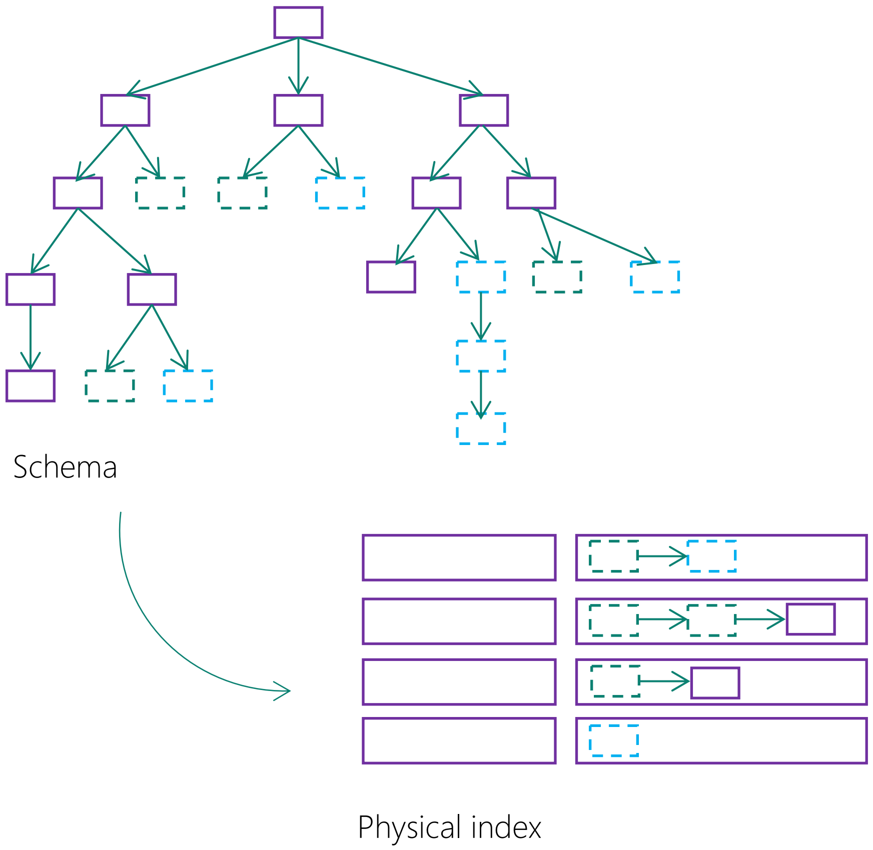
- New capability – SubPartitioning – targeted for end-of-year 2021
  - Allows specifying a partition key based on multiple fields - for example Device Model and Device Id
  - This would result in all Devices of a certain model being co-located on a single or multiple physical partitions. Allows to scale for a single Device Model > 20 GB data / 10,000 RU – but still allows queries to only target a small subset of physical partitions

Partitioning: Summary

- A good partition key is expected to have the following characteristics-
  - The data distribution is reasonably even across all the keys, i.e., you don’t want to have a select few keys to have the majority of the data.
  - The workload is reasonably even across all the keys, i.e., you don’t want the (majority of the) workload to be focused on a few specific keys.
  - Generally prefer to have more keys. The large the keys the better.
  - PartitionKey is known and can be provided for CRUD operations and ideally most common queries
  - If you have the need to be able to update multiple documents within the same atomic transaction (via triggers or Stored Procedures) they need to share the same logical partition key.

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Indexing



Schema-agnostic, automatic indexing : Handle any data without manual schema or index management

At global scale, schema/index management is painful

Automatic and synchronous index management

Hash, range, and geospatial

Works across every data model

Highly write-optimized database engine

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Indexing

- **Tuning Index policy**
  - By default all nodes are indexed. Request units are charged for indexing during Writes (POST/PUT) – each term being indexed costs a fragment of an RU – but especially for documents larger 1 KB tuning the index policy to only index nodes that are queries by reduces cost for Writes

Point-lookups vs. queries

- **Point-lookups vs. queries**
  - For heavy read/query scenarios it can be beneficial to denormalize data to allow retrieval by point-lookups vs. queries to avoid the RU charges for compute overhead with queries
    - RU charge for point-lookup of 1 KB document (with Session or lower consistency) is 1 RU.
    - Query ‘SELECT * FROM c WHERE c.id == “<RowKey>” and c.pk == “<PartitionKey>”’ for the same document would cost nearly 3 RU
    - Often only useful when certain queries are executed very often. Example: Xbox Licensing service with around 100 K requests per second saved roughly 25% of RUs with this optimization

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Data model: Common patterns

- IoT Write-Heavy, in many cases the intuitive PK DeviceId would result in skweness, often pretty much free form query requirements with only common filtering base don time (last x days)
  - Consider: Model where time-fragment is part of CollectionName – like “Events20191121”, “Events20191122” etc. (could be based on week, month or year based on volume instead). Consider using “/id” as PK (very well distribution for ingestion, makes cross partition keys required, but with the Collection model above number of partitions is very limited)
- Multi-Tenancy
  - Hard trade-off between different options:
    - TenantId as (part of) partition key
    - Different Containers for each tenant
  - Consider a mixed model – like shared Container where TenantId is (part of) partition key for free beta testers, but dedicated containers for you whale accounts

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Data model: Common patterns

- Read-heavy:
  - Optimize data-model for point lookups vs. queries
  - Understand and optimize the Top N queries
  - Implement pagination logic in your REST endpoints that can be efficient
- Example:
  - UST Entitlements Service

Data model: Common patterns

- Example: Original pseudo contract

```
{
        “id”: “<Guid>”,
        “EntitlementKey”: “big:ProductId:SkuId”,
        “UserId”: “SomeUser”,
        “Status”: “Active”
        […]
}
```

- ProductId/SkuId identify the digital good, could be a game, a subscriptions, consumables like bullets that can be used in a game (1 user could own multiple entitlements for the same ProductId/SkuId combination)
- Typical query pattern: SELECT e.* FROM e WHERE e.UserId in (“User123”, “UserABC”, “Device789”) AND e.Status == “Active” AND e.EntitlementKey == “big:Halo5:GoldEdition”
- Problems:
    - Query vs. Point-lookup
    - Query plan/Execution order:
        - Complexity, Order of Filter criteria vs. statistics (* to be changed)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Data model: Common patterns

- Example: Optimized pseudo contract

```
{
        “id”: “big:ProductId:SkuId”,
        “UserId”: “SomeUser”,
        “Entitlements”: [
                {
                        “EntitlementId”: “<Guid>”,
                        “Status”: “Active”,
                        […]
                }[, …]
        ]
}
```

Caution: Not a general best practice – mileage will vary based on use case. But worth considering…

- Most common Query pattern (by EntitlementKey for a set of users) would be executable by point lookups + some in-memory filtering (like status==active)
- For remaining queries ensure Single UserId as first filter (to make sure it is always processed first)
- Some very unselective filter criteria (>99% of entitlement “Active”) consider filtering in memory instead

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

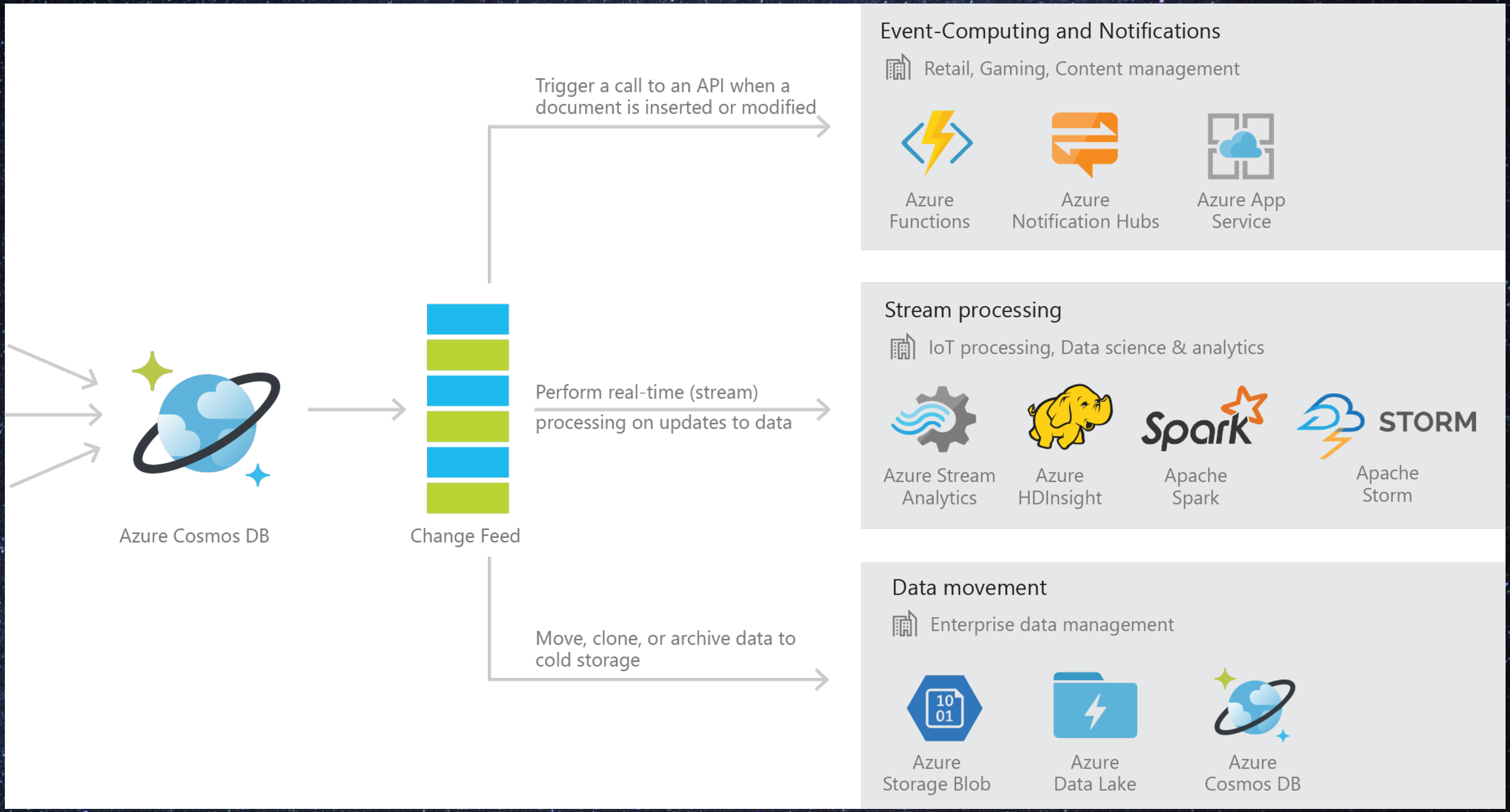
Document size considerations

- **Modeling for smaller document sizes**
    - CosmosDB has a hard limit of 2MB (soon increasable to up-to 16 MB) per document
    - Request units being charged for read/query as well as writes depend on the document payload. So especially in scenarios where documents are often retrieved or updated it is often preferable to split documents.
    - When storing multiple documents with the same logical partition key within the same collections Transactional Batch API or stored procedures can be used for updates across documents within atomic transactions

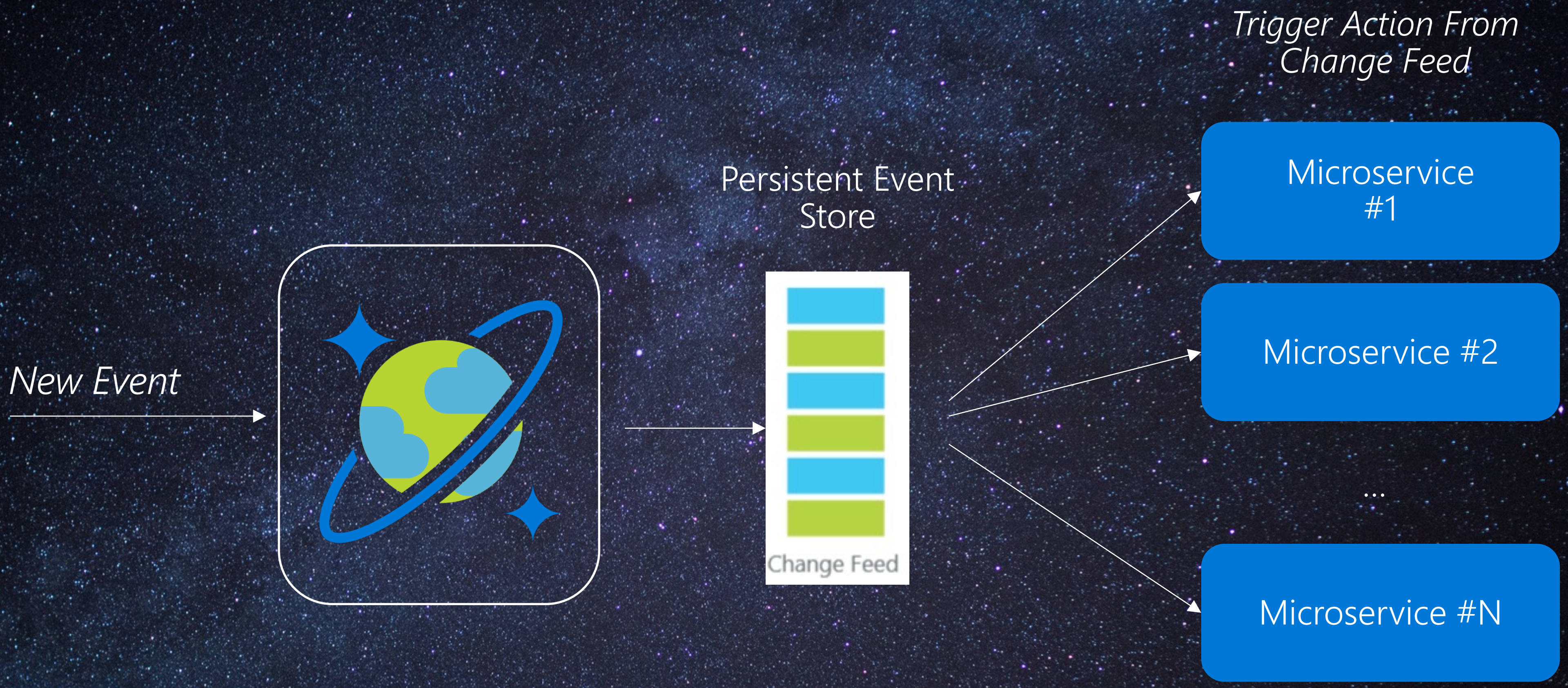
https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Cosmos DB change feed

Persistent log of records within an Azure Cosmos DB container in the order in which they were modified


Azure Cosmos DB
Change Feed
Trigger a call to an API when a document is inserted or modified
Perform real-time (stream) processing on updates to data
Move, clone, or archive data to cold storage
Event-Computing and Notifications
Retail, Gaming, Content management
Azure Functions
Azure Notification Hubs
Azure App Service
Stream processing
IoT processing, Data science & analytics
Azure Stream Analytics
Azure HDInsight
Spark
Apache Spark
STORM
Apache Storm
Data movement
Enterprise data management
10 01
Azure Storage Blob
Azure Data Lake
Azure Cosmos DB

Event Sourcing for Microservices



New Event

Persistent Event Store

Trigger Action From Change Feed

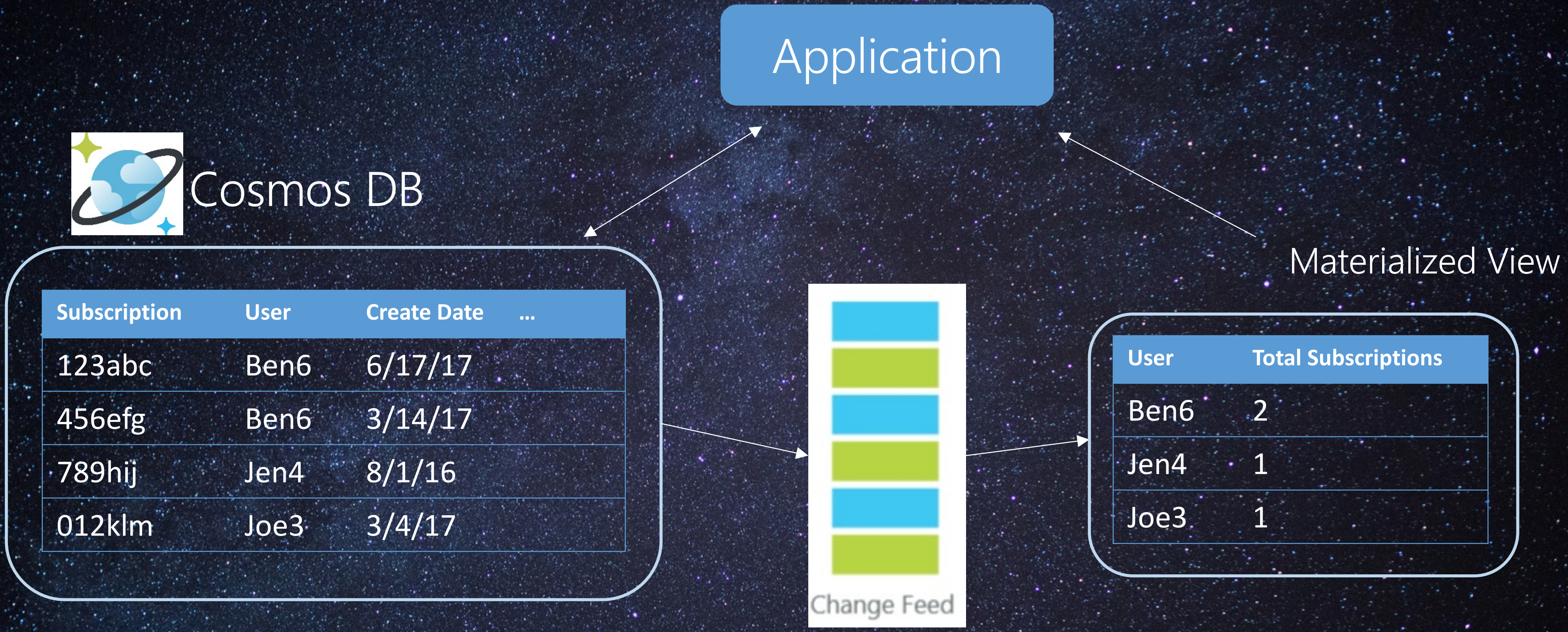
Microservice #1

Microservice #2

…

Microservice #N

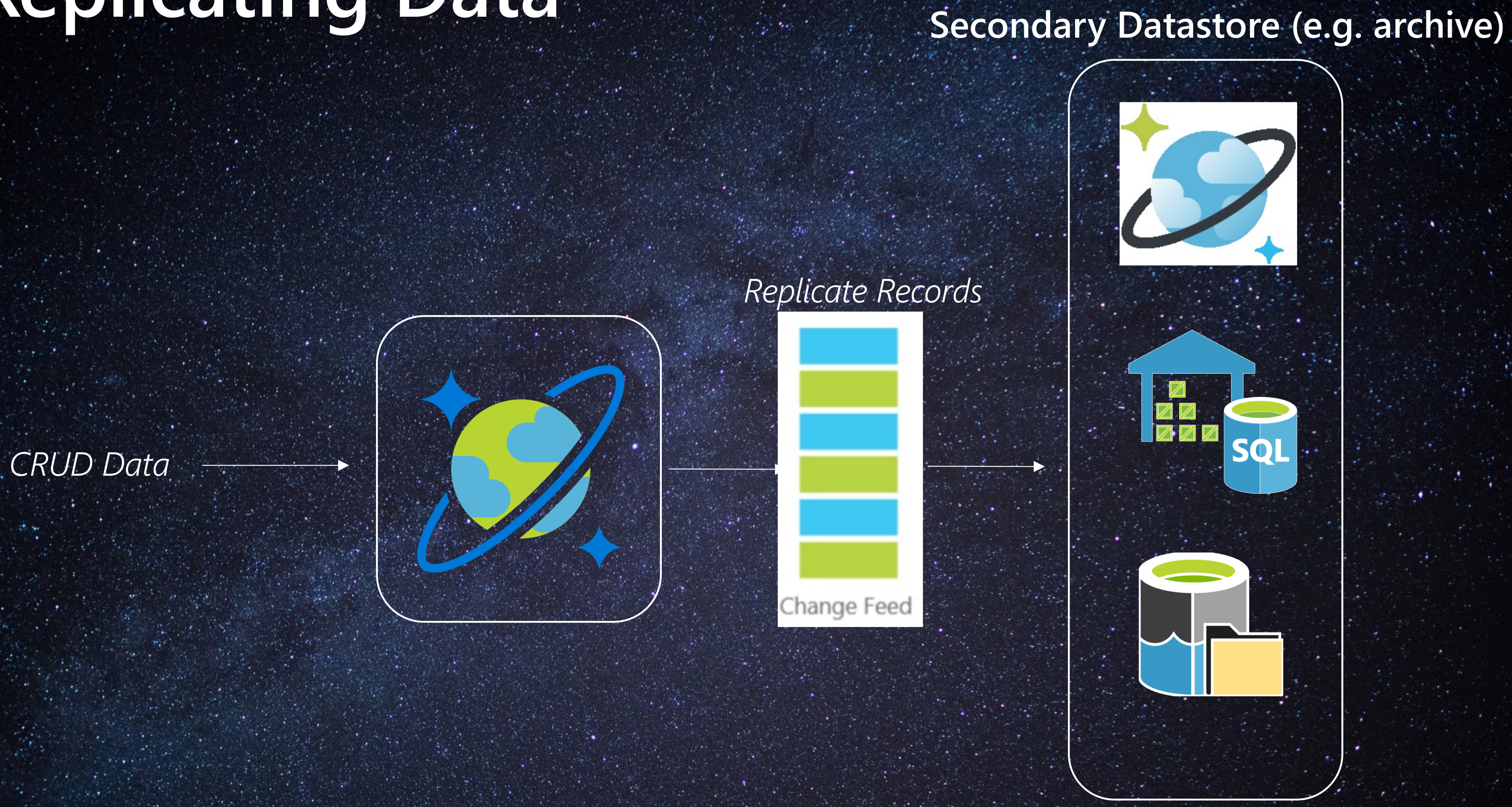
Materializing Views



Subscription	User	Create Date	...
123abc	Ben6	6/17/17	
456efg	Ben6	3/14/17	
789hij	Jen4	8/1/16	
012klm	Joe3	3/4/17	

User	Total Subscriptions
Ben6	2
Jen4	1
Joe3	1

Replicating Data

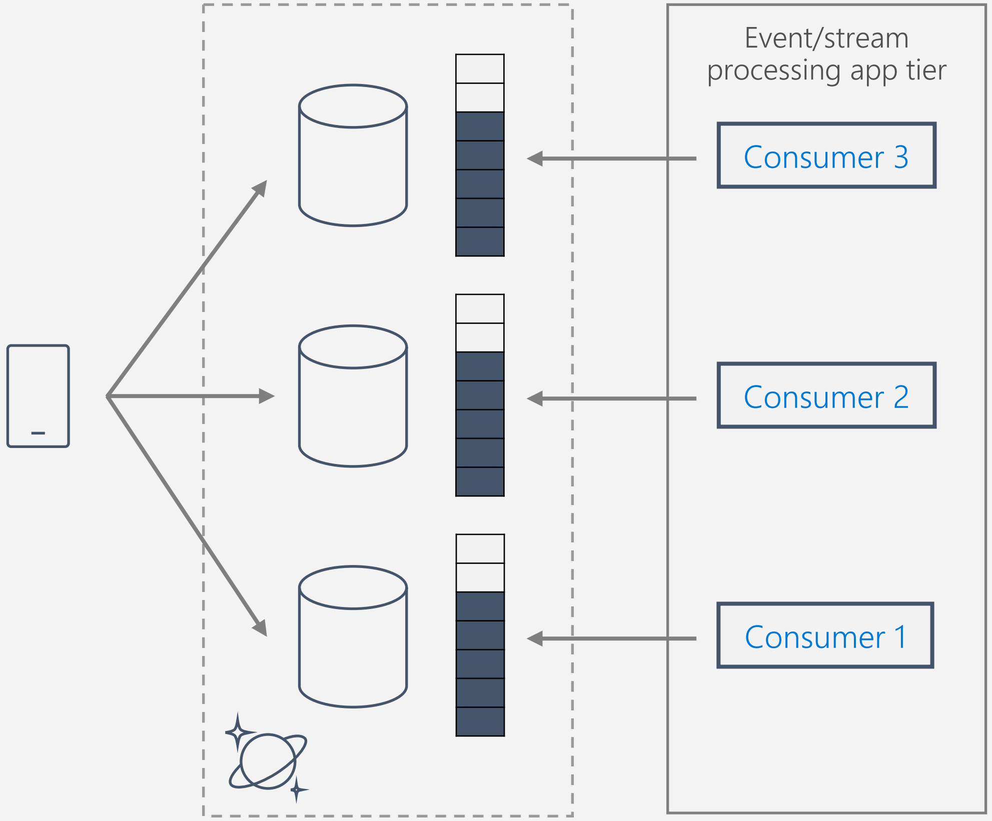


Change Feed with partitions

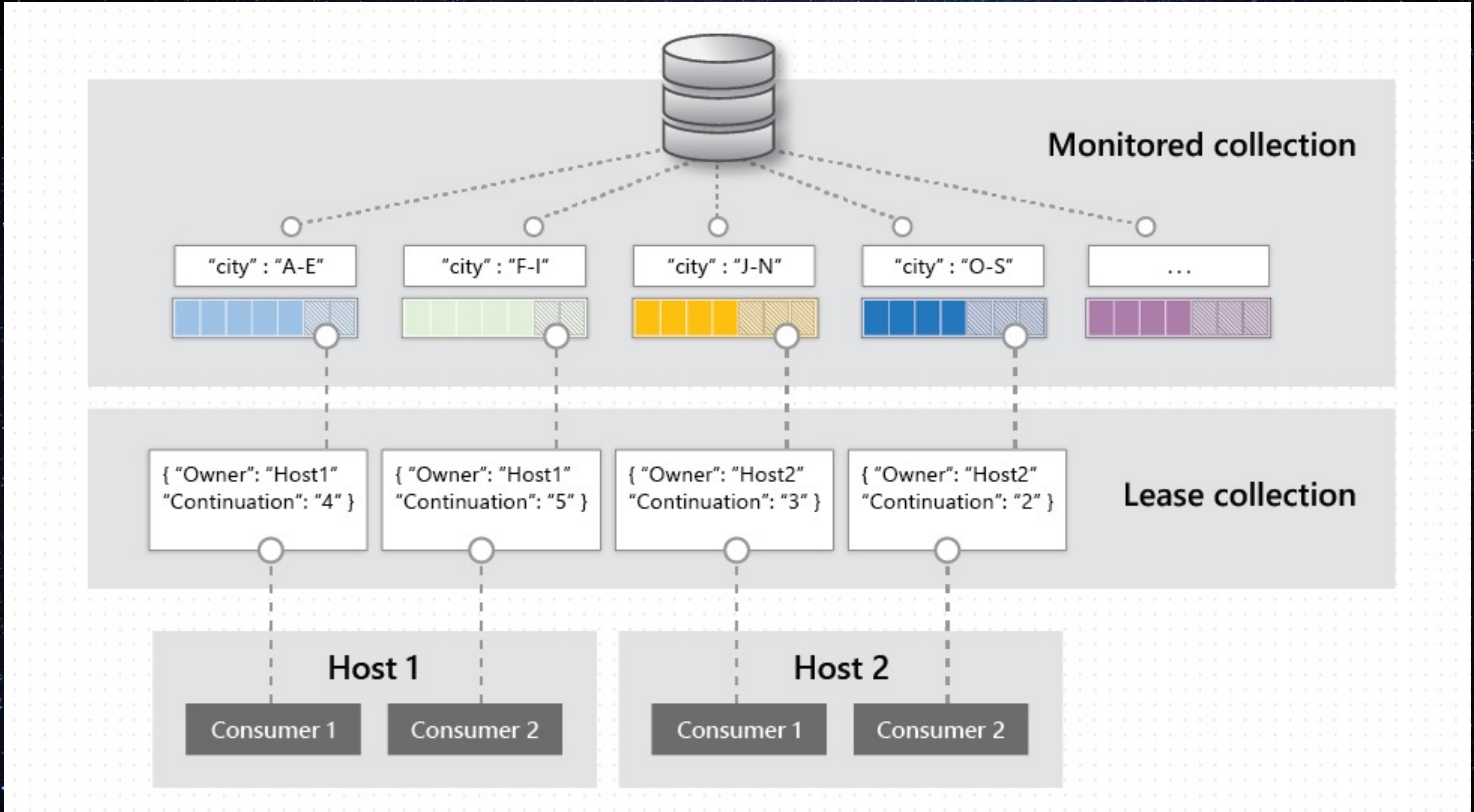
Consumer parallelization

Change feed listens for any changes in Azure Cosmos DB collection. It then 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. The change feed is available for each partition key range within the document collection, and thus **can be distributed across one or more consumers for parallel processing.**



Behind the Scenes



Monitored collection

"city" : "A-E"

"city" : "F-I"

"city" : "J-N"

"city" : "O-S"

...

Lease collection

{ "Owner": "Host1" "Continuation": "4" }

{ "Owner": "Host1" "Continuation": "5" }

{ "Owner": "Host2" "Continuation": "3" }

{ "Owner": "Host2" "Continuation": "2" }

Host 1

Consumer 1

Consumer 2

Host 2

Consumer 1

Consumer 2

Agenda

- Introduction
  - Cosmos DB Overview
  - NoSQL vs. RDBMS
  - Common use cases
- High availability (HA) and Disaster Recovery (DR)
  - Overview
  - Consistency models
  - Terminology
  - Trade-offs
- Provisioning model
  - Shared database vs. container specific provisioning
  - Auto Pilot vs. manually provisioned throughput vs. serverless
  - RI – Reserved Instances
- Cosmos DB APIs
  - Introduction
  - Sql/Gremlin/Table
  - Cassandra/Mongo
- Data modeling best practices
  - Partitioning
  - Indexing
  - Point-lookups vs. queries
  - Common patterns
  - Document size considerations
  - Change feed + materialized views
- Getting in touch…

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520

Getting in touch …

- Engineering
  - Fabian Meiswinkel (fabianm@microsoft.com)
- Product management
  - Theo van Kraay (theo.van@microsoft.com)

https://aka.ms/CosmosDB/AzureMeetupFrankfurt/20210520