



# COSMOS DB

Leave a  
Review

Info

Eshant Garg

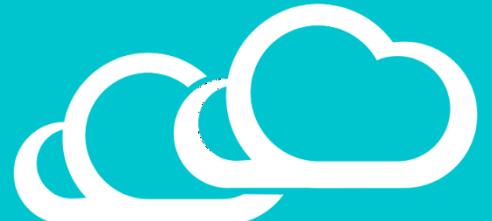
Azure Data Engineer, Architect, Advisor

[eshant.garg@gmail.com](mailto:eshant.garg@gmail.com)

# Cosmos DB

## Azure NoSQL Database





**LearnCloud.Info**



**Eshant Garg**

Data Engineer, Architect, Advisor



LearnCloud.Info

## Why Cosmos DB?

What traditional databases were lacking?

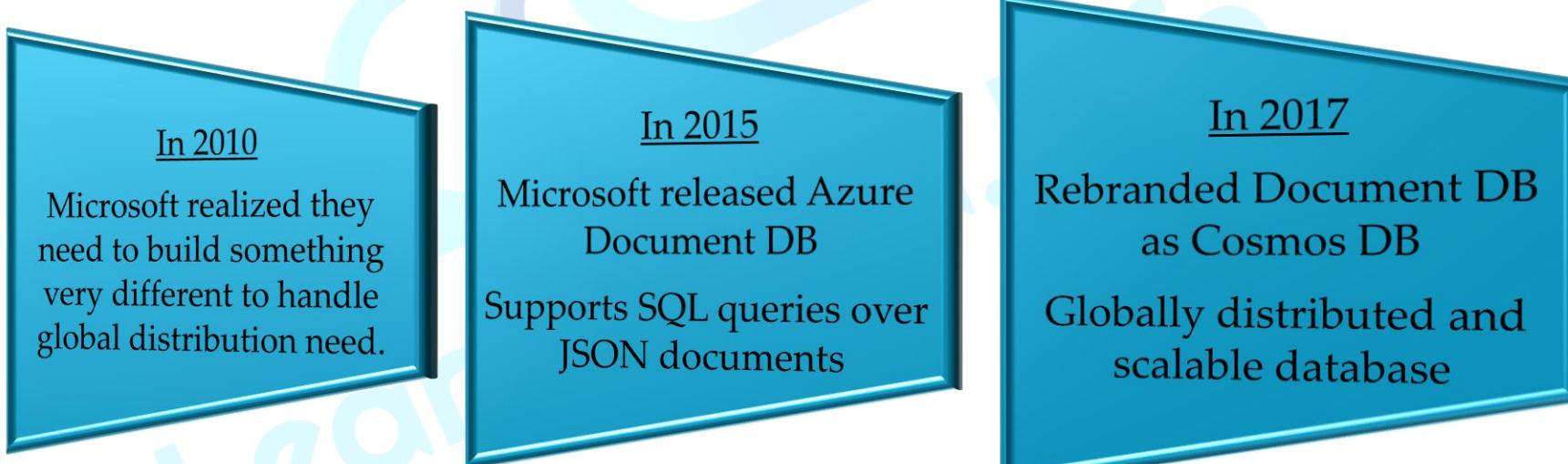


# Challenges with globally distributed Databases

- Long time
- Lot of effort
- Need own infrastructure
  - Teams
  - Data centers etc.



# How Cosmos DB evolved?



LearnCloud



# Why Cosmos DB?

- 
- ### FULLY MANAGED
- Database as a service (DaaS)
    - Serverless architecture
    - No operational overhead
  - No schema or index management

### GLOBALLY DISTRIBUTED

- Turnkey global distribution

### MULTIMODEL & MULTI-LANGUAGE

- Supports Jason documents, table graph and columnar data models
  - Java, .NET, Python, Node.js, JavaScript, etc.

### CONSISTENCY CHOICES

- Azure Cosmos DB's support for consistency levels like strong, eventual, consistent prefix, session, and bounded-staleness.

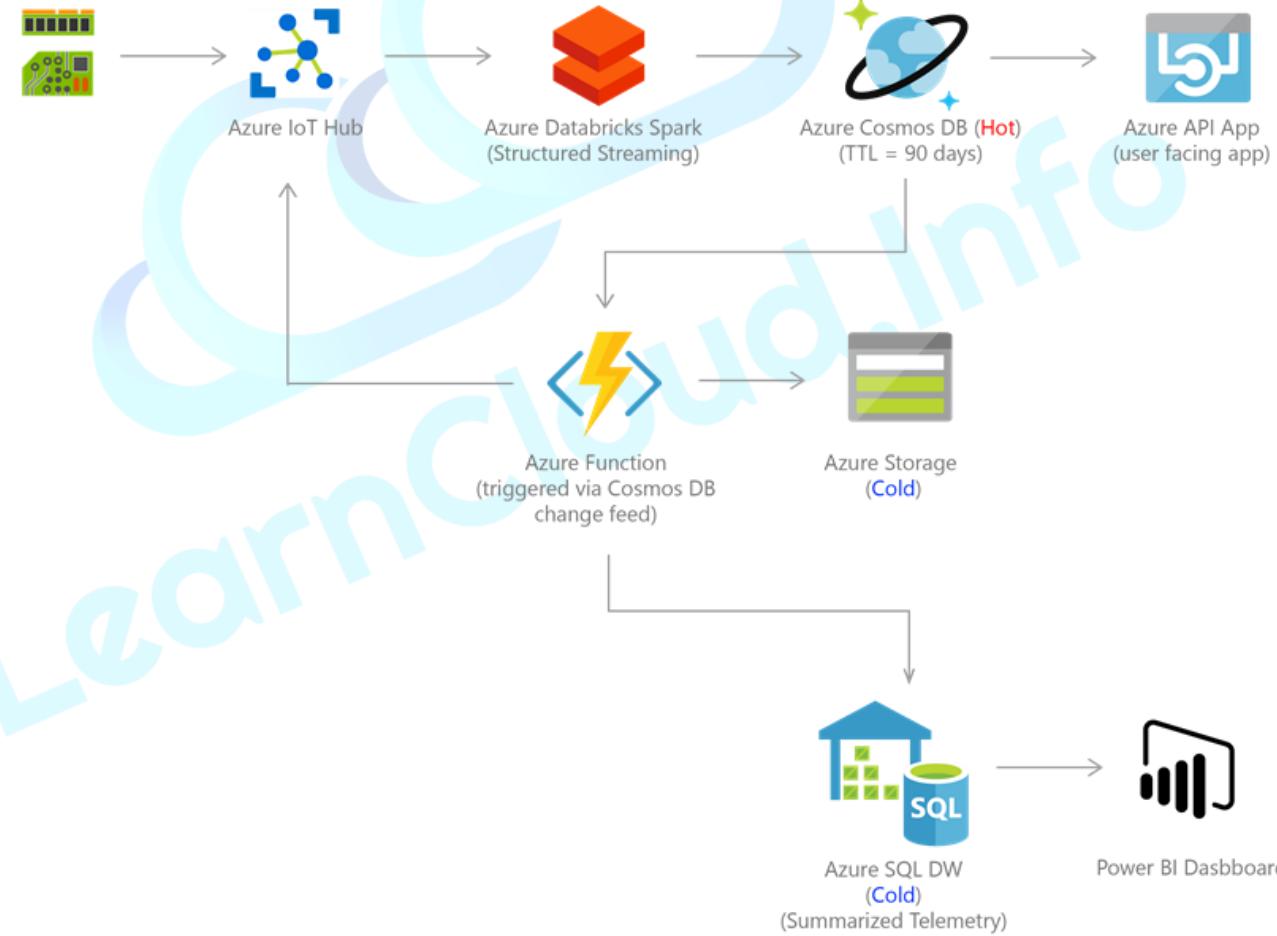
### SCALABLE

- Unlimited scale for both storage and throughput.

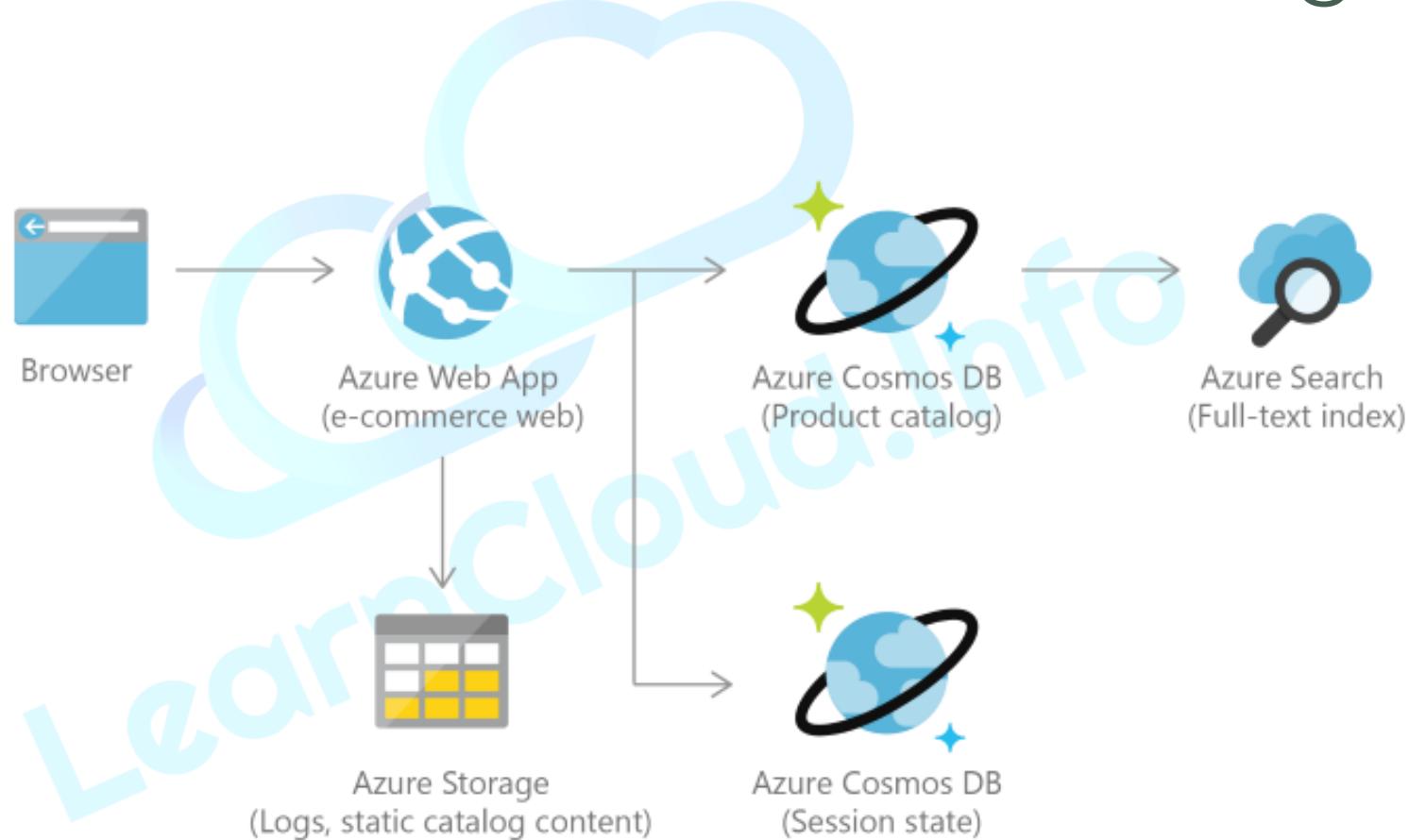
### HIGHLY AVAILABLE, RELIABLE & SECURE

- Always on
- 99.999% SLA
- < 10ms latency

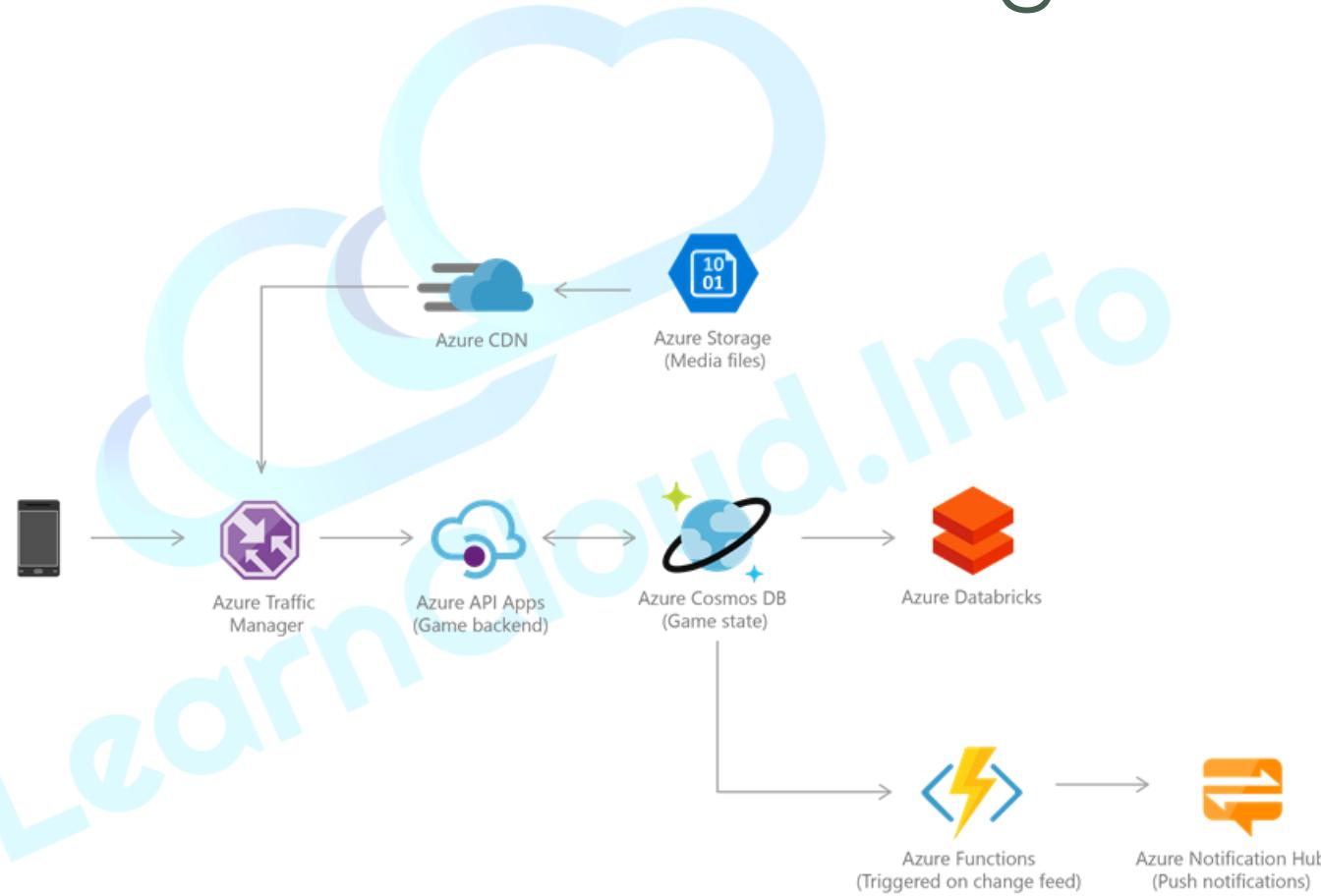
# Use case - IoT



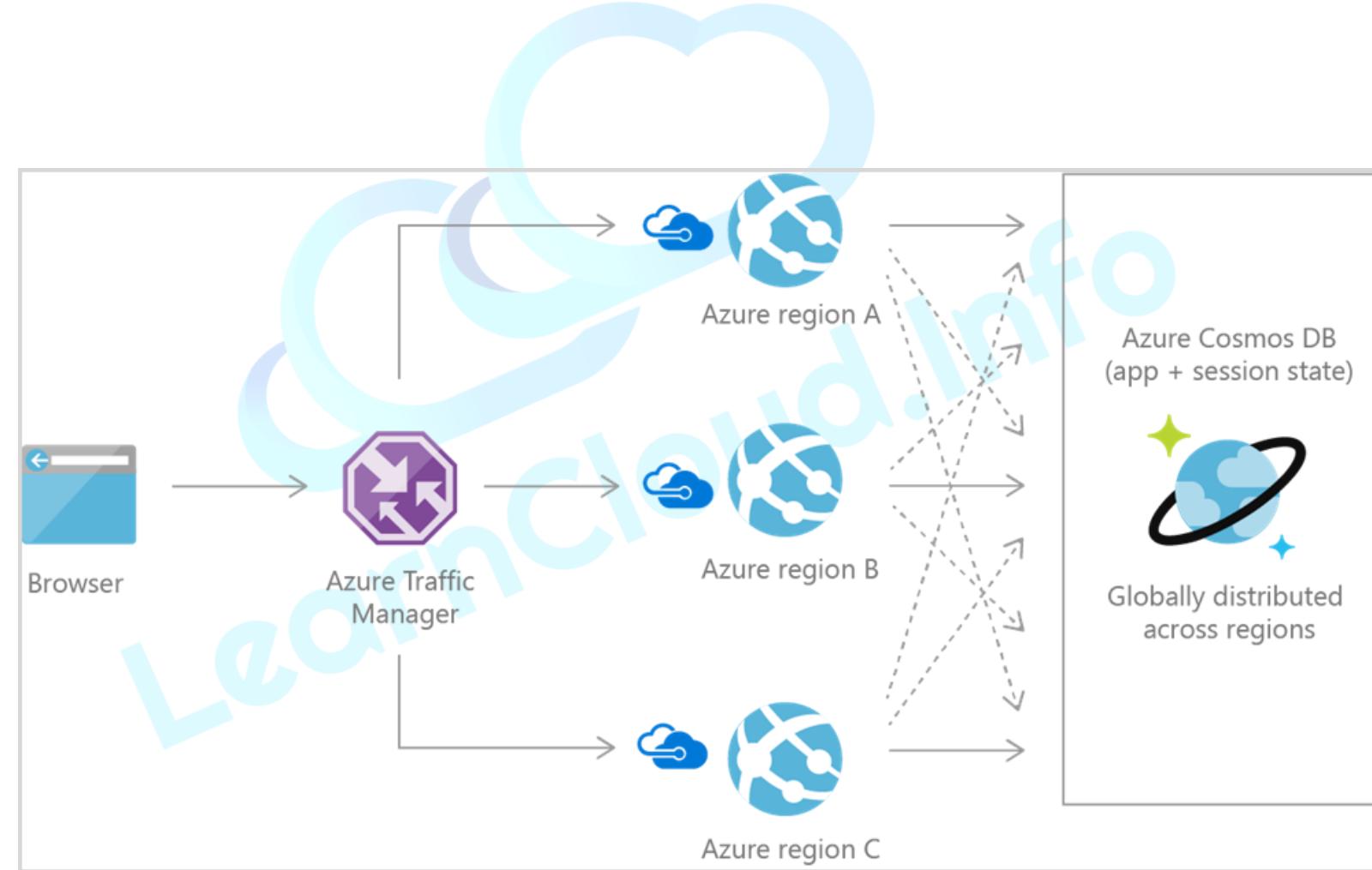
# Use case – Retail and Marketing



# Use case – Gaming

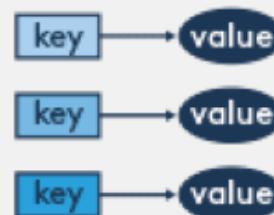


# Use case – Web and mobile

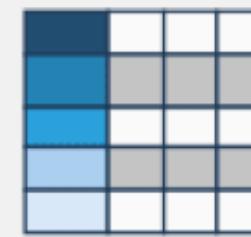


# NoSQL

## Key-Value



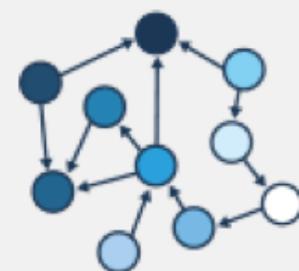
## Column-Family



Table



## Graph



## Document

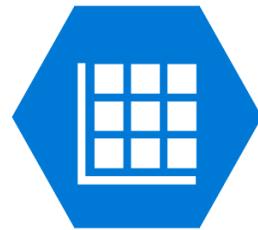
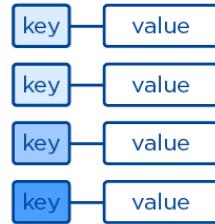


# SQL API vs MongoDB API

| SQL(CORE) API  | MongoDB API   |
|--|---|
| JSON Documents   | BSON Documents  |
| Microsoft original Document DB platform                | Implement Wire protocol                                 |
| Supports server side programming model                 | Fully compatible with Mongo DB application code         |
| You can use SQL like language to query JSON documents. | Migrate existing Cosmos DB without much change of logic |
|  | Use SQL(CORE) API for new development                   |

# Cosmos DB Table API

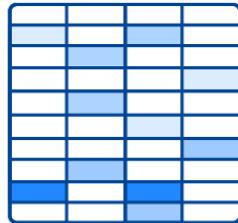
## Key-Value



- Key-Value store
- Premium offering for Azure Table Storage
- Existing Table Storage customers will migrate to Cosmos DB Table API
- Row value can be simple like number or string
- Row cannot store object

# Cosmos DB Cassandra API

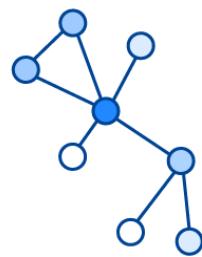
Wide-column



- Wide column No SQL Database
- Name and format of column can vary from row to row.
- Simple migrate your Cassandra application to Cosmos Cassandra API and change connection string.
- Interact
  - Cassandra based tools
  - Data Explorer
  - Programmatically, using SDK (CassandraCSharpdriver)

# Cosmos DB Gremlin API

**Graph**

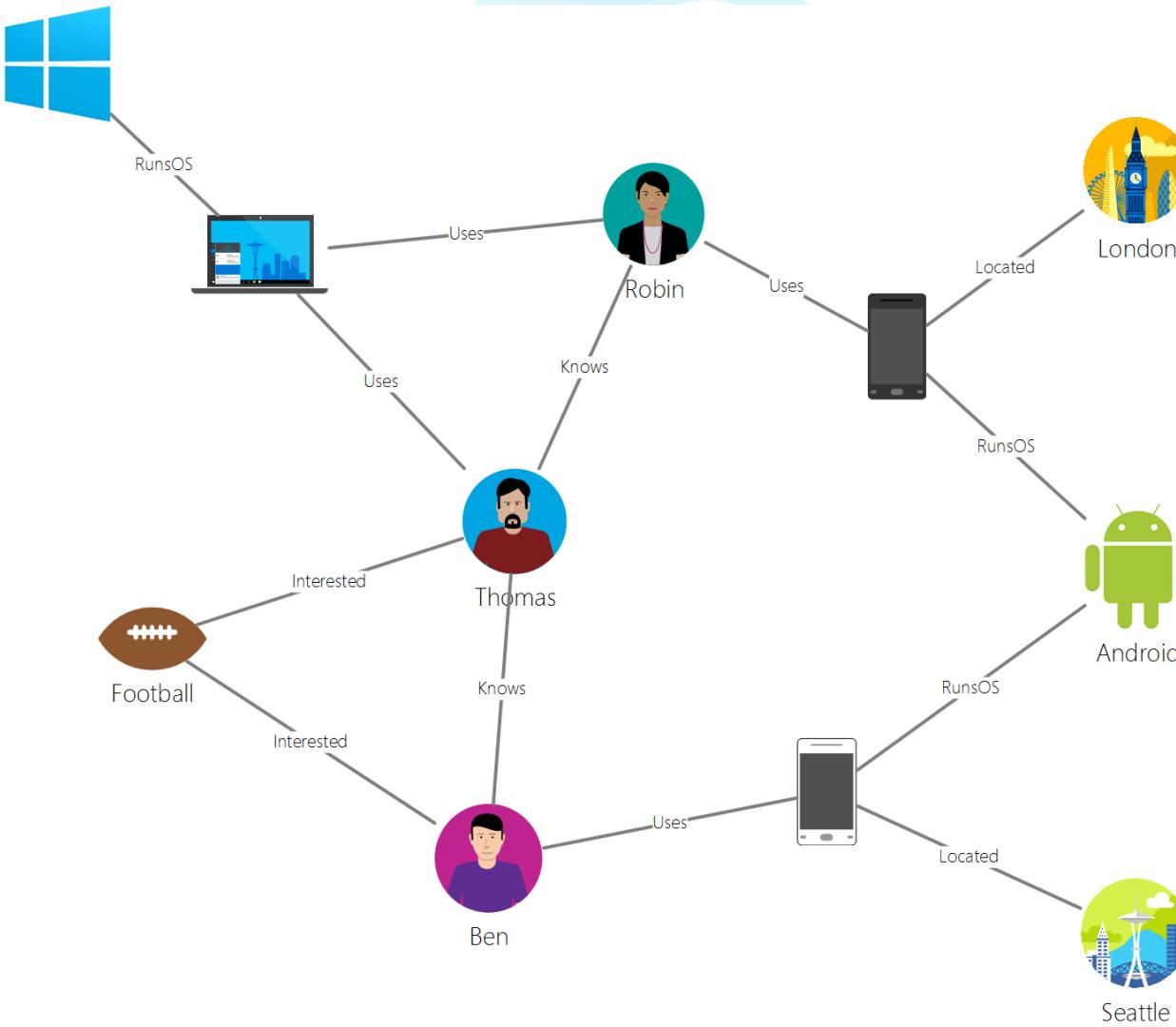


- Graph Data Model
- Real world data connected with each other
- Graph database can persist relationships in the storage layer



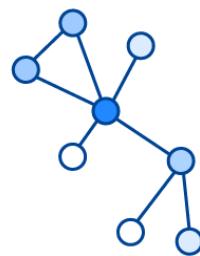
LearnCloud.info

# Graph Model



# Cosmos DB Gremlin API

Graph



- Graph Data Model
- Real world data connected with each other
- Graph database can persist relationships in the storage layer
- Use cases
  - Social networks
  - Recommendation engines
  - Geospatial
  - Internet of things
- Migrate existing apps to Cosmos DB Gremlin API
- Graph traverse a language

# Analyze the decision criteria



|   | Core<br>(SQL) | MongoDB | Cassandra | Azure<br>Table | Gremlin |
|---|---------------|---------|-----------|----------------|---------|
| New projects being created from scratch                   | ✓             |         |           |                |         |
| Existing MongoDB, Cassandra, Azure Table, or Gremlin data |               | ✓       | ✓         | ✓              | ✓       |
| Analysis of the relationships between data                |               |         |           |                | ✓       |
| All other scenarios                                       | ✓             |         |           |                |         |

# Azure Table storage vs Cosmos DB Table API

## Azure Table Storage

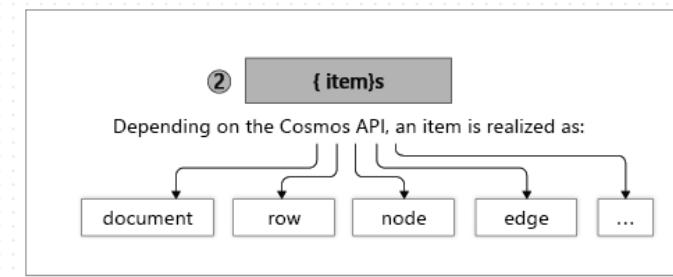
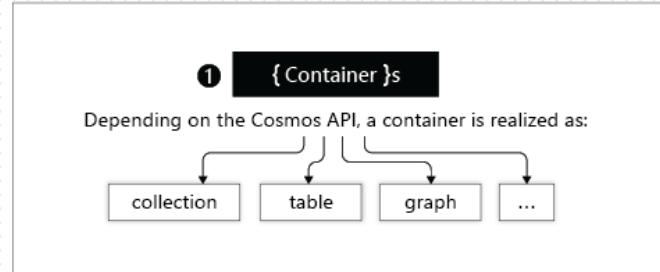
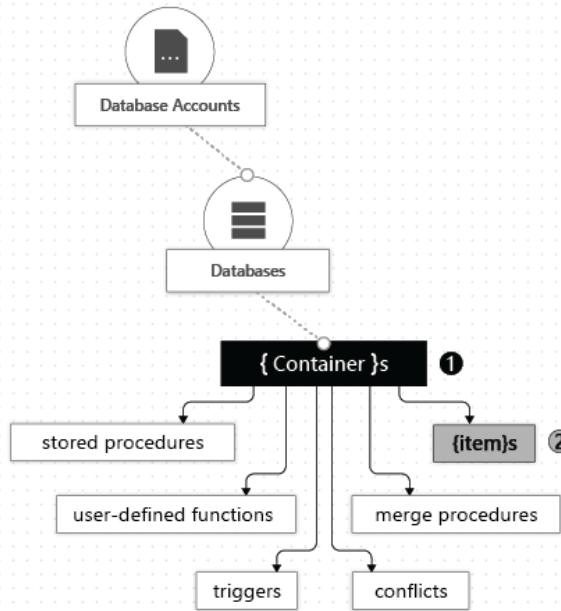
- Geo replication is restricted
  - Only 1 additional pair region
- Support for primary key lookups only
- Price optimized for cold storage
- Lower performance
  - Throughput is capped
  - Latency is higher
- No consistency options

## Cosmos DB Table API

- Geo replication across your choice of any number of regions
- Secondary index support for lookups across multiple dimensions
- Better performance
  - Unlimited and predictable throughput
  - latency is lower
- 5 consistency options

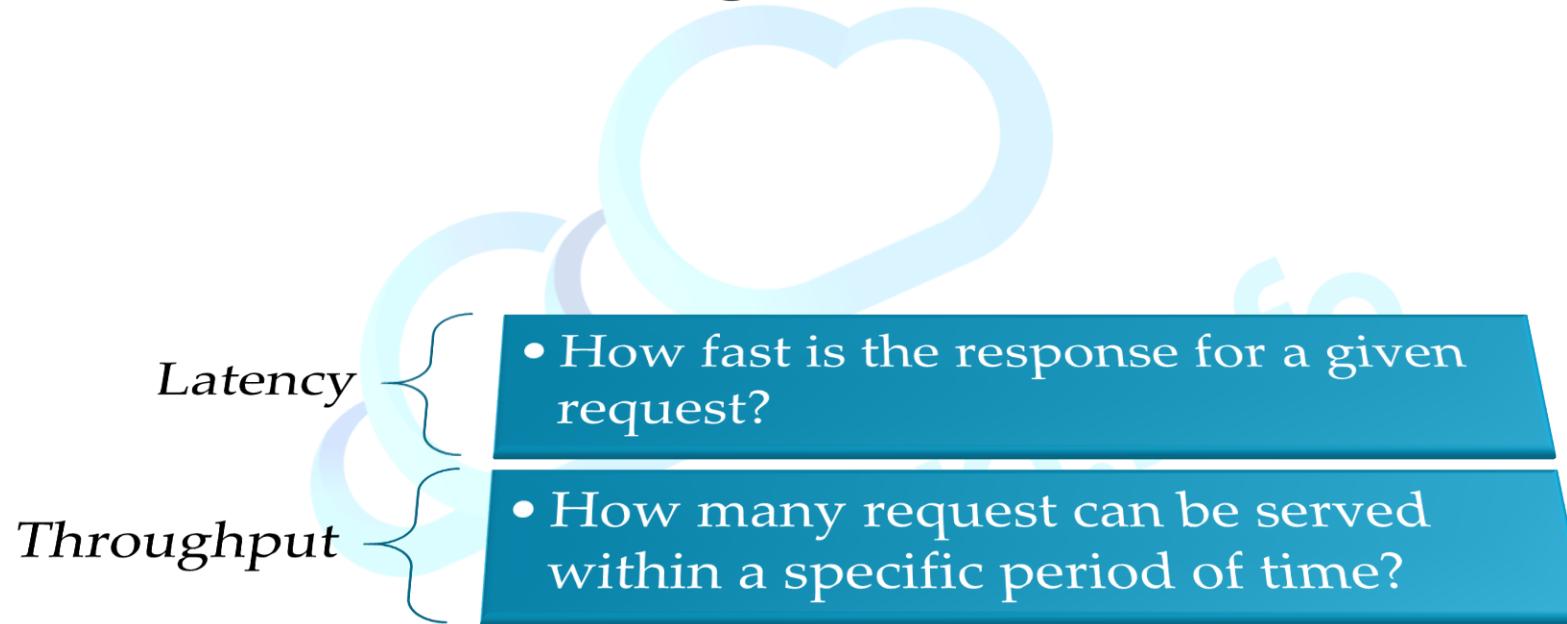
VS

# Database Containers and Items

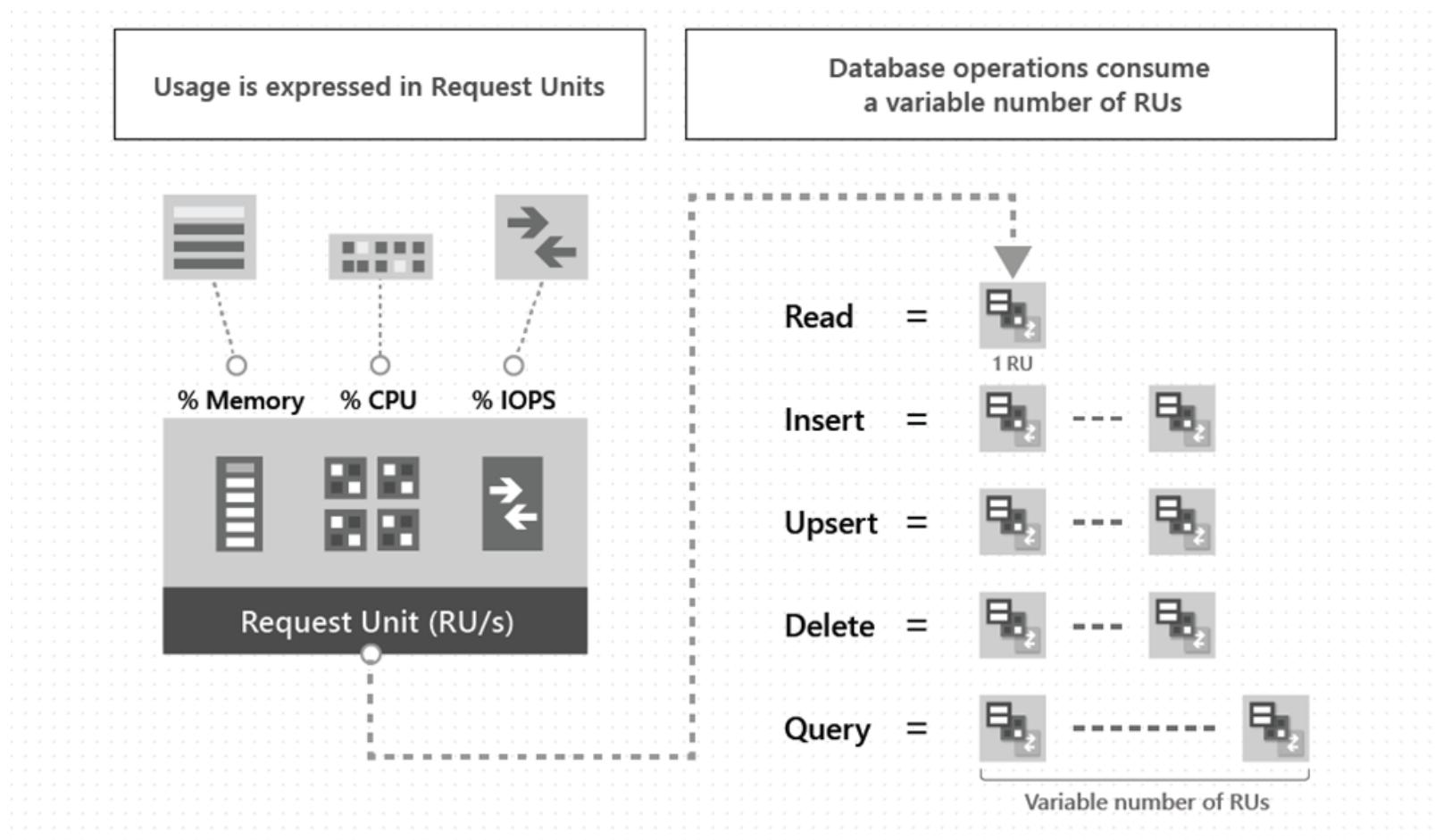


| Azure Cosmos entity    | SQL API   | Cassandra API | MongoDB API | Gremlin API  | Table API |
|------------------------|-----------|---------------|-------------|--------------|-----------|
| Azure Cosmos database  | Database  | Keyspace      | Database    | Database     | NA        |
| Azure Cosmos container | Container | Table         | Collection  | Graph        | Table     |
| Azure Cosmos item      | Document  | Row           | Document    | Node or edge | Item      |

# Measuring Performance



# Introducing Request Units



# Introducing Request Units

The screenshot shows the Azure Cosmos DB SQL API interface. On the left, the database structure is visible with a collection named 'departuredelays' selected. In the center, a query editor window titled 'Query 1' contains a single line of SQL:

```
1  SELECT * FROM d WHERE d.origin = 'ABE'
```

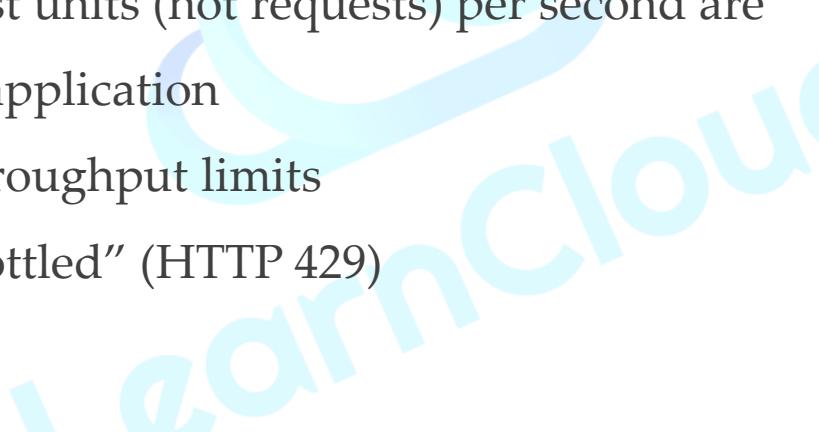
Below the query, there are two tabs: 'Results' and 'Query Stats'. The 'Query Stats' tab is highlighted with a red box. The results table shows the following metrics:

| METRIC          | VALUE                 |
|-----------------|-----------------------|
| Request Charge  | 46.18000000000001 RUs |
| Showing Results | 1 - 100               |
| Round Trips     | 1                     |



# Reserving requests units

- Provision Request units per second (RU/s)
  - How many request units (not requests) per second are available to your application
- Exceeding reserved throughput limits
  - Requests are “throttled” (HTTP 429)



A screenshot of a web form for provisioning Azure Cosmos DB throughput. The form includes fields for Container id (B), Partition key (/id), and a checkbox for My partition key is larger than 100 bytes. A red box highlights the 'Provision dedicated throughput for this container' checkbox and the 'Throughput (400 - 100,000 RU/s)' input field, which contains the value 400. Below the input field are decrease (+) and increase (-) buttons. At the bottom, it shows an estimated spend of \$0.58 hourly / \$13.82 daily (8 regions, 400RU/s, \$0.00016/RU).

\* Container id

\* Partition key

My partition key is larger than 100 bytes

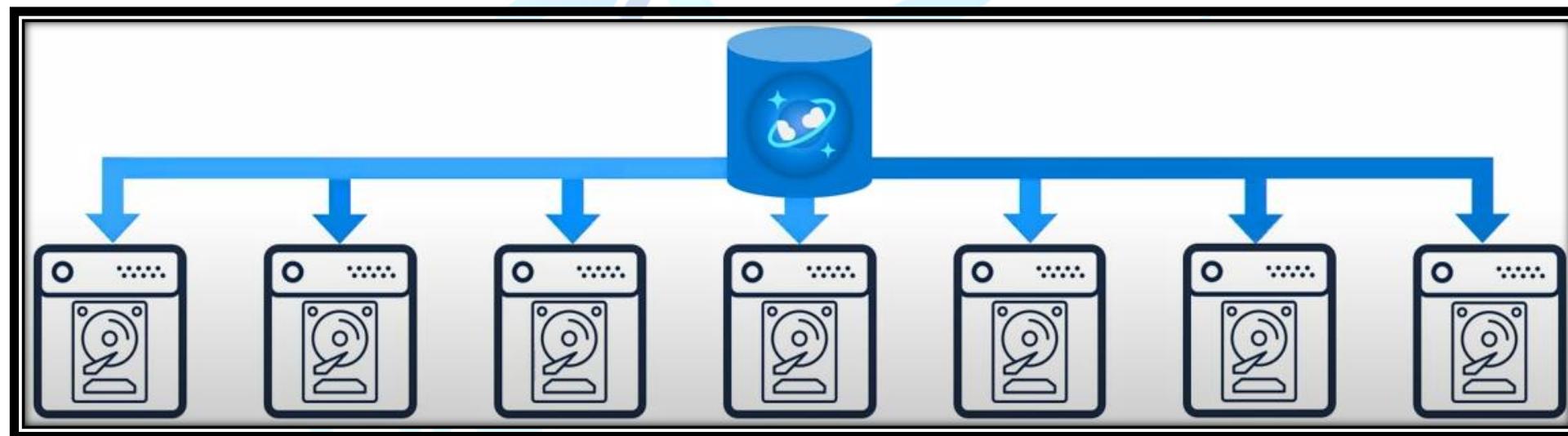
Provision dedicated throughput for this container

\* Throughput (400 - 100,000 RU/s)  - +

Estimated spend (USD): **\$0.58 hourly / \$13.82 daily** (8 regions, 400RU/s, \$0.00016/RU)



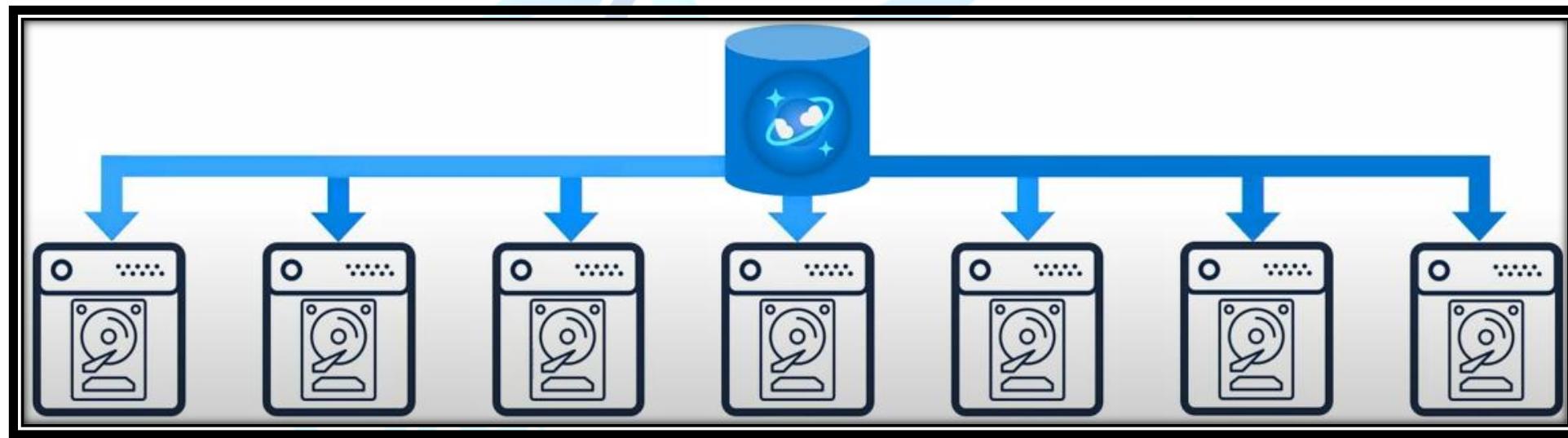
# Horizontally Scalable



Unlimited Storage

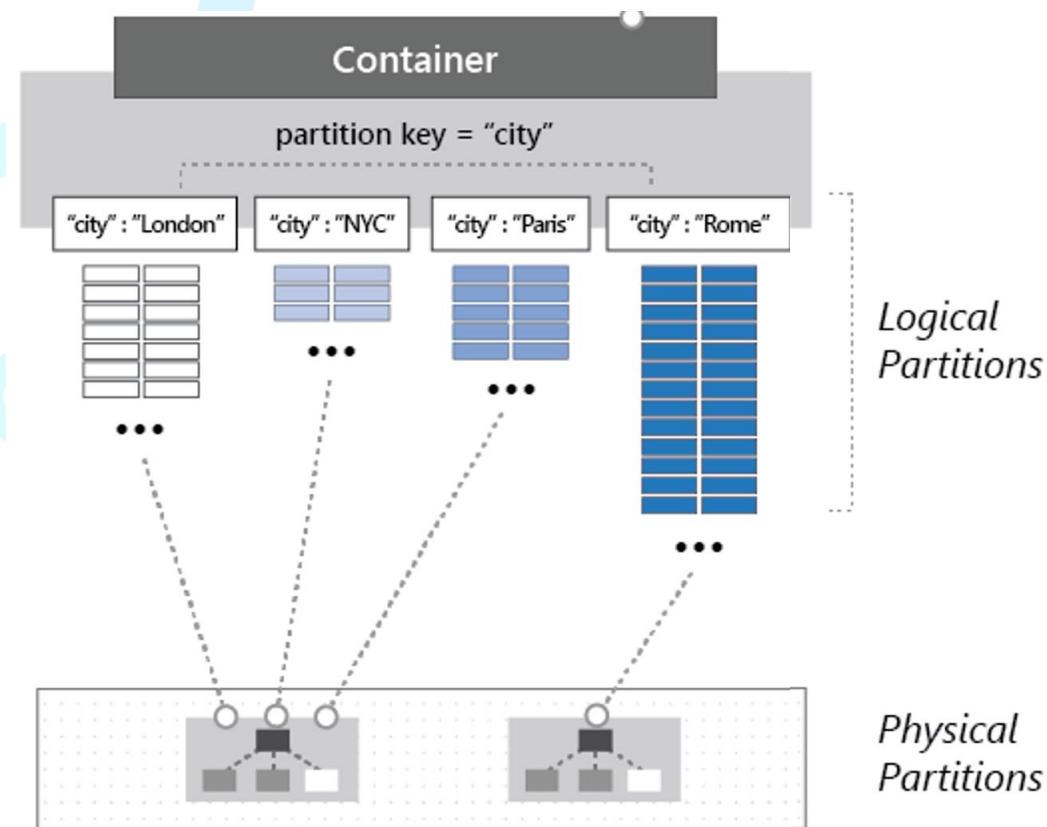
Unlimited Throughput

# Partitioning

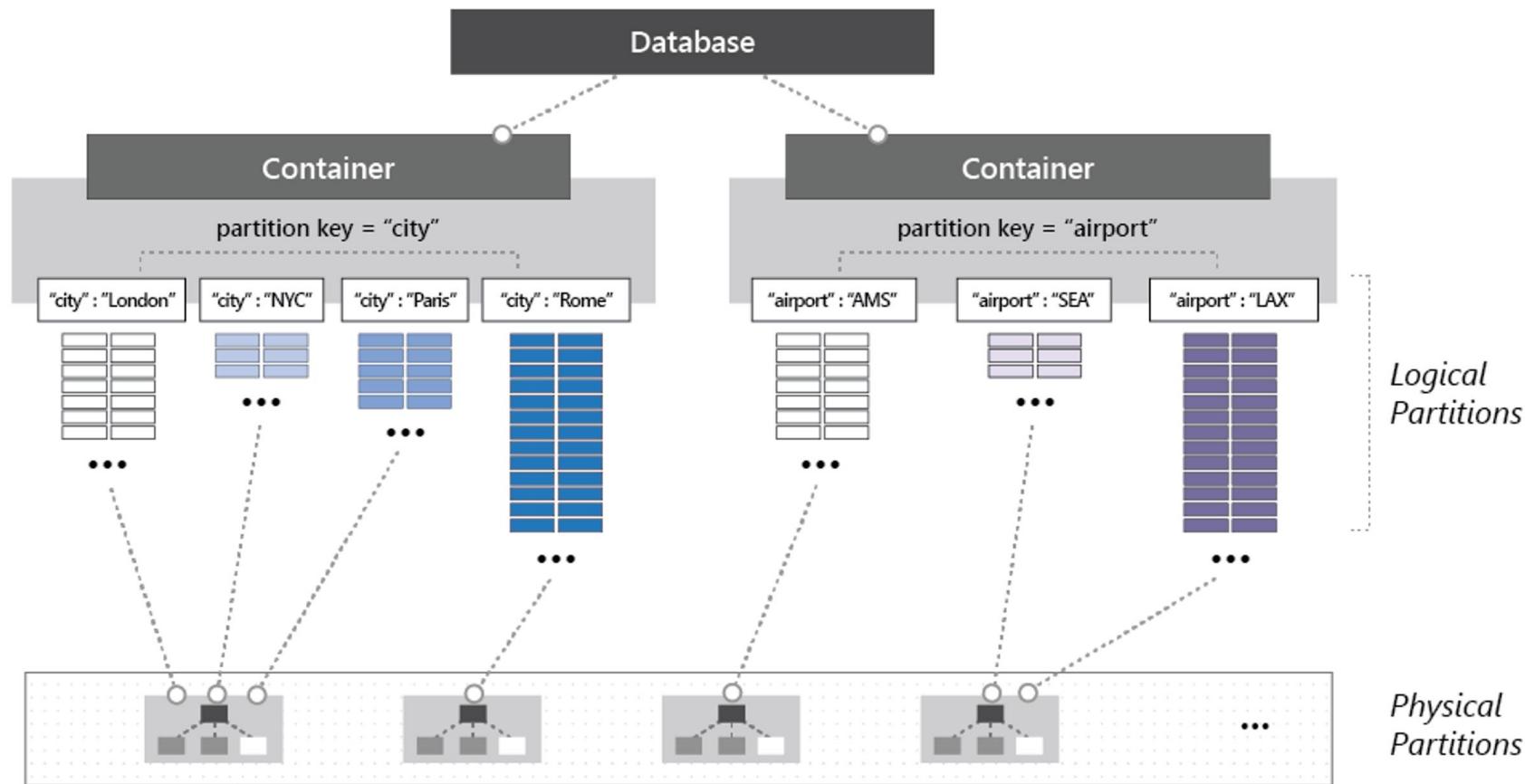


# Partitioning

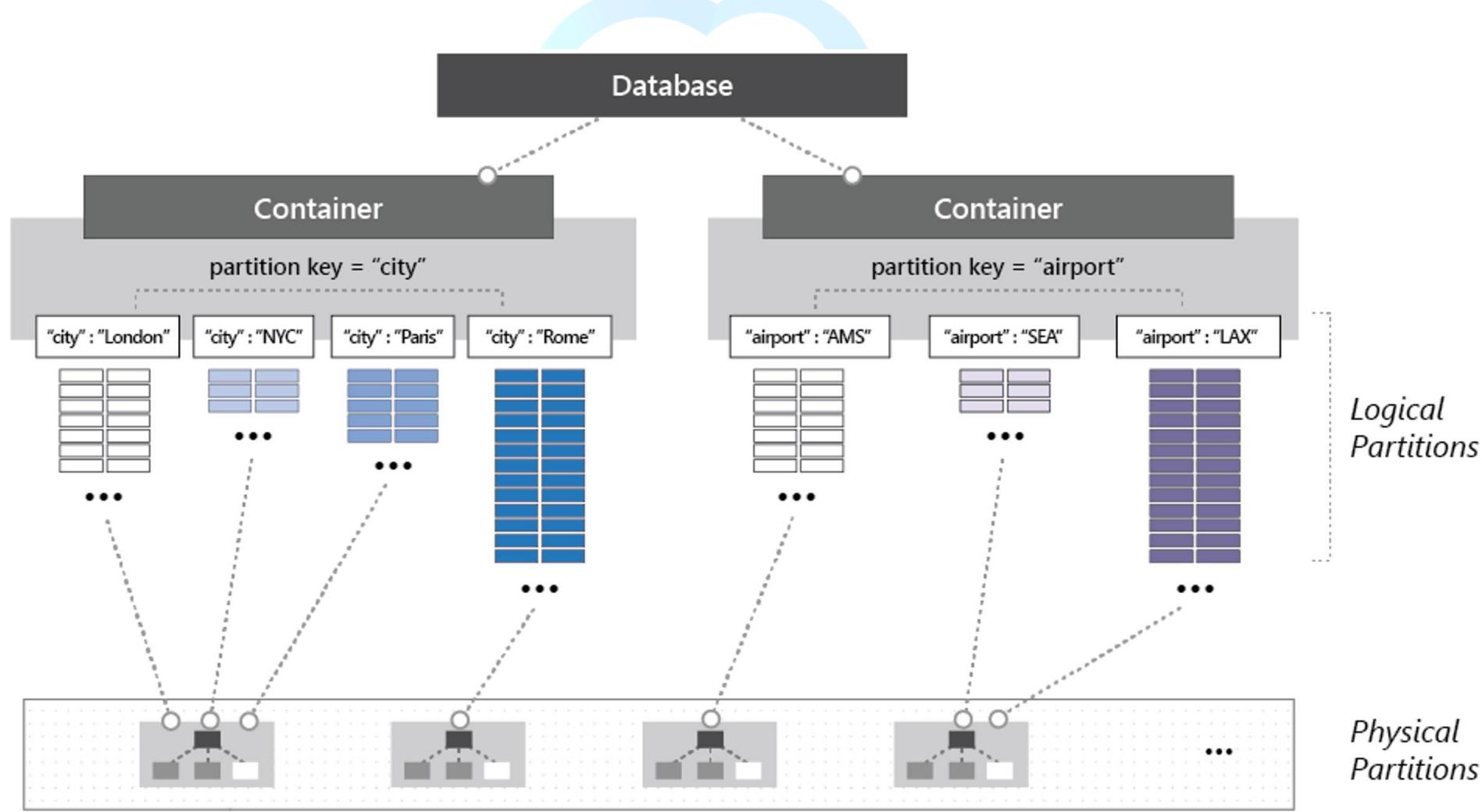
- **Partitioning:** the items in a container are divided into distinct subsets called logical partitions.
- **Partition key** is the value by which Azure organizes your data into logical divisions.
- **Logical partitions** are formed based on the value of a partition key that is associated with each item in a container.
- **Physical partitions:** Internally, one or more logical partitions are mapped to a single physical partition.



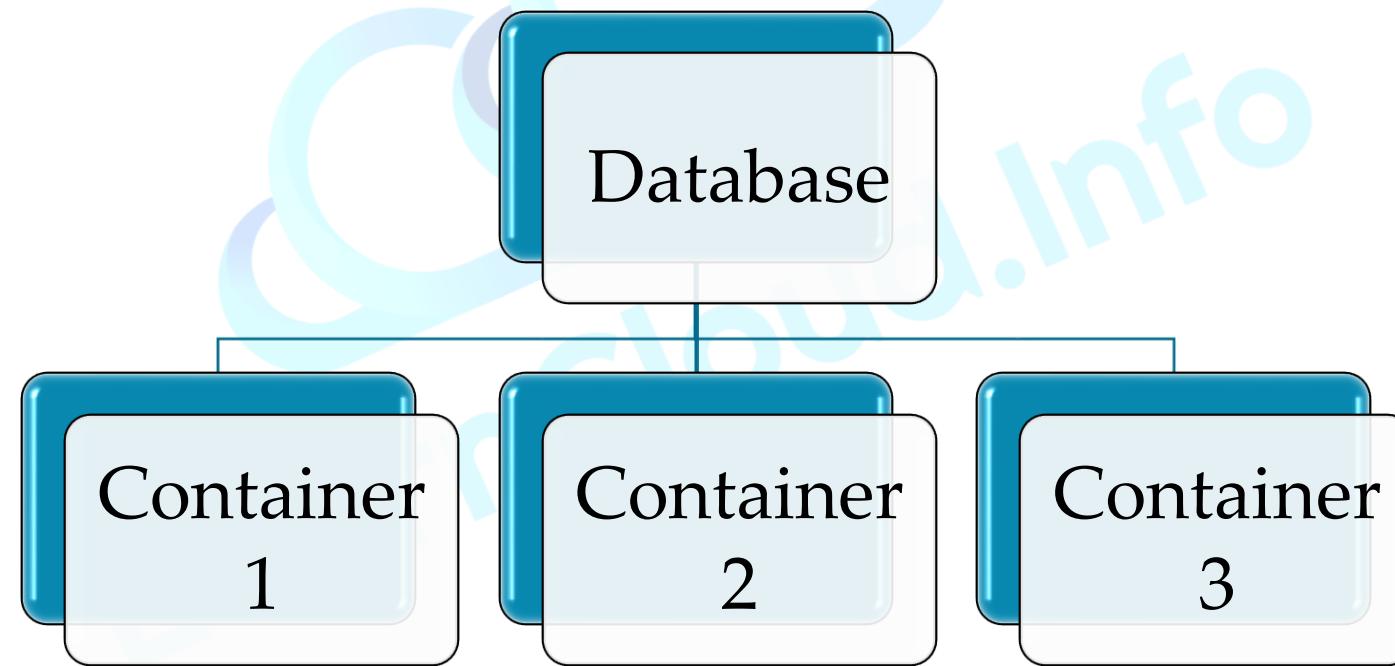
# Partitioning



# Dedicated vs Shared throughput



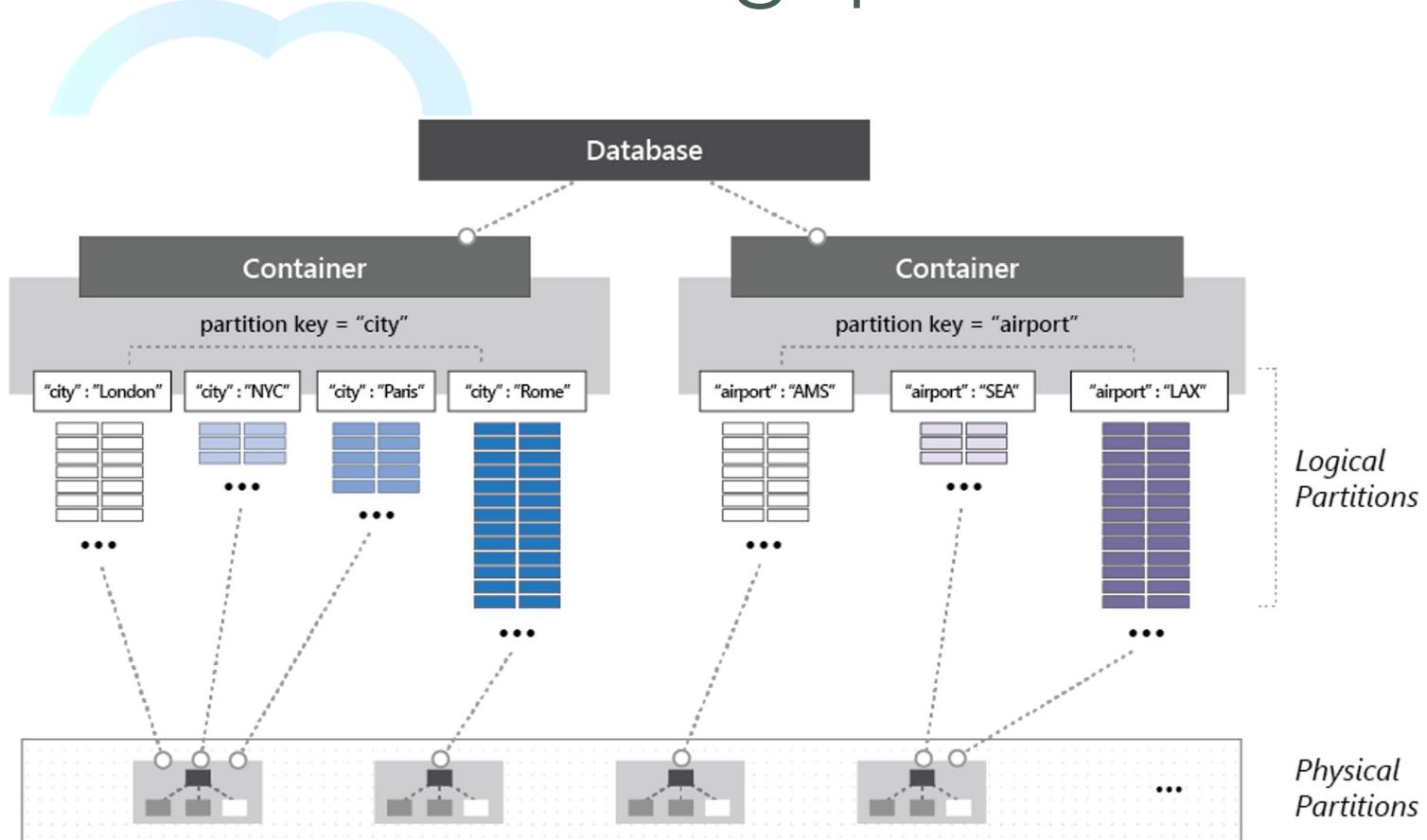
# Dedicated vs Shared throughput



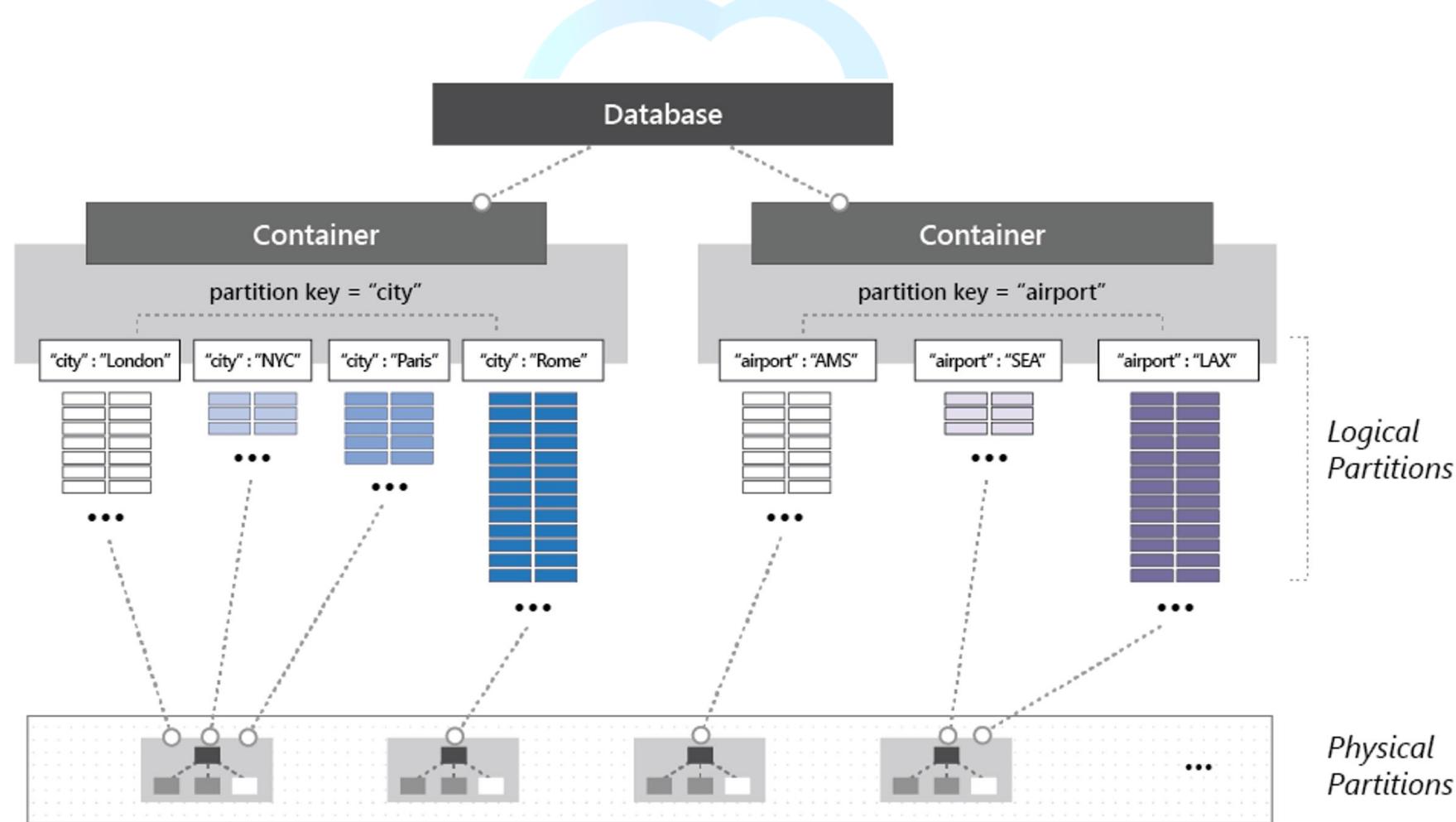
# Dedicated vs Shared throughput

- You can set throughput at:
  - Database level – Shared throughput
  - Container level – Dedicated throughput
  - It is recommended to set throughput at container level.
- Rate-Limited
- Choose at the time of creation

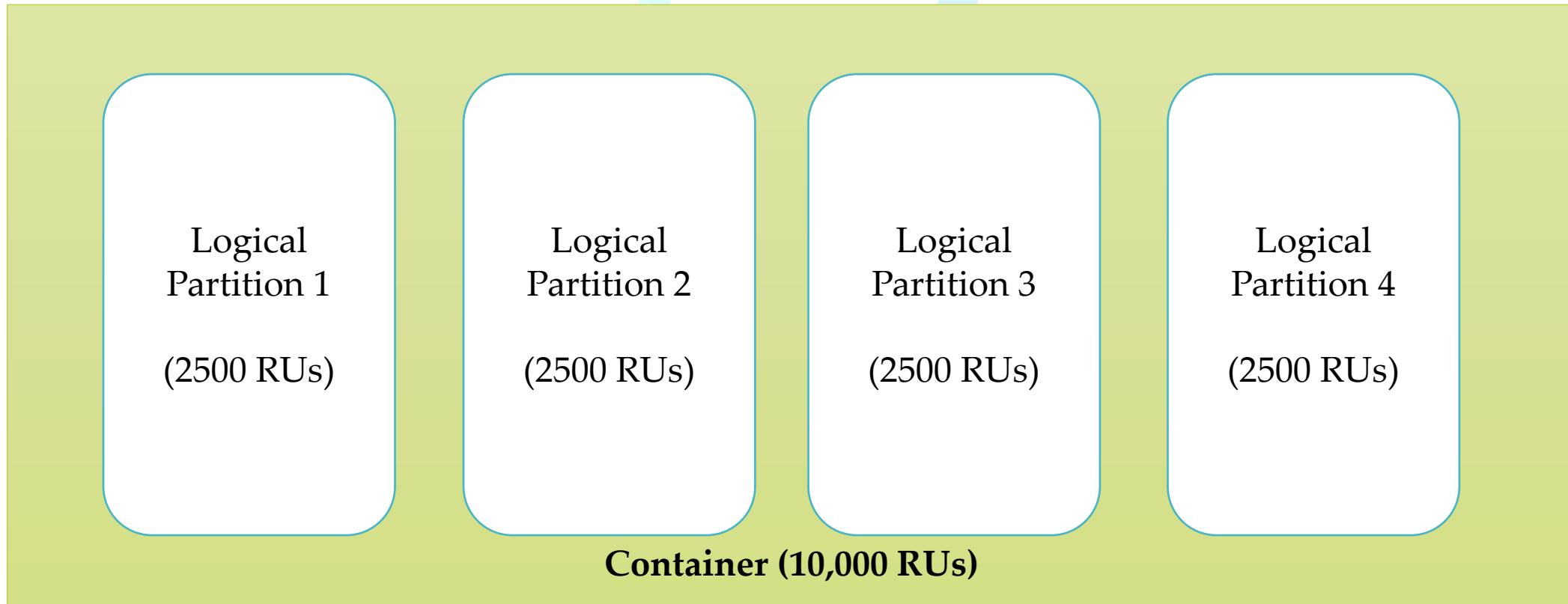
LearnCloud



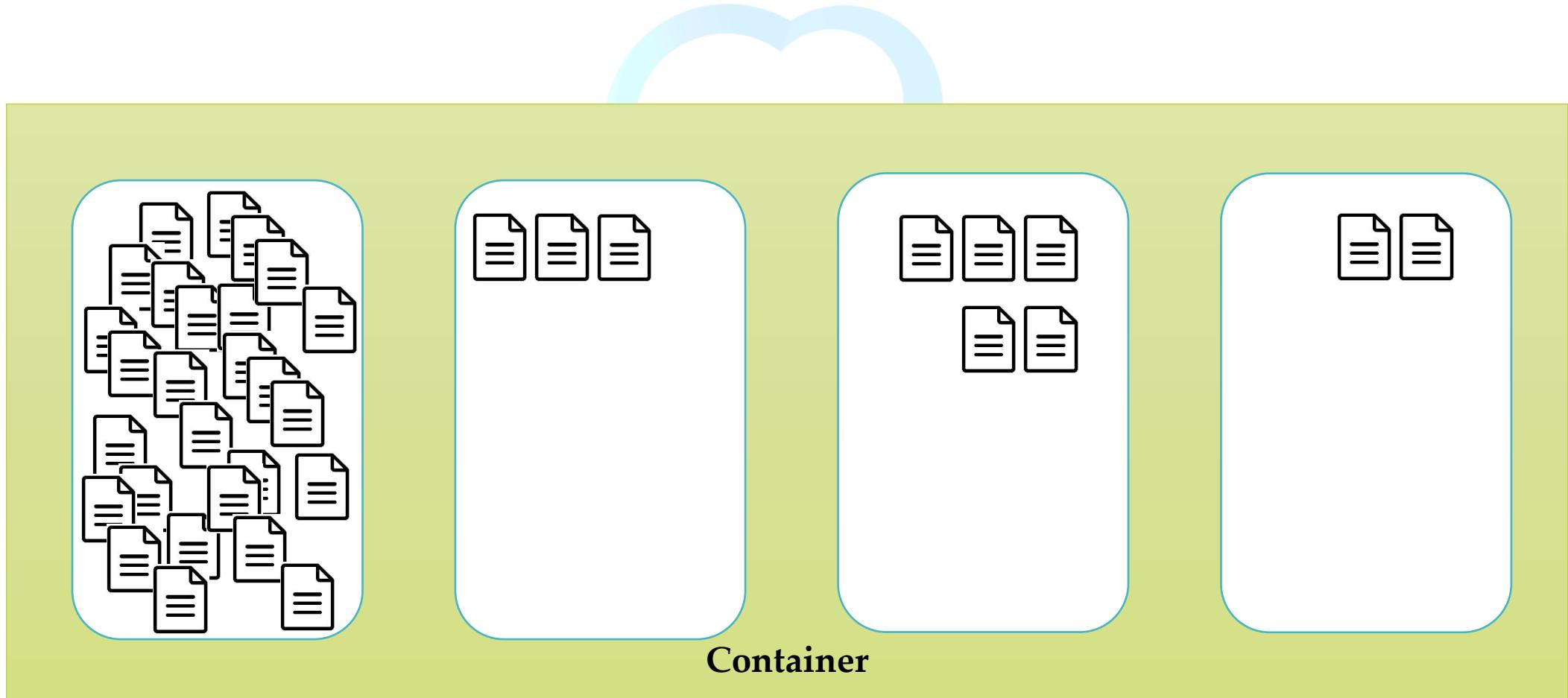
# Avoiding hot partition



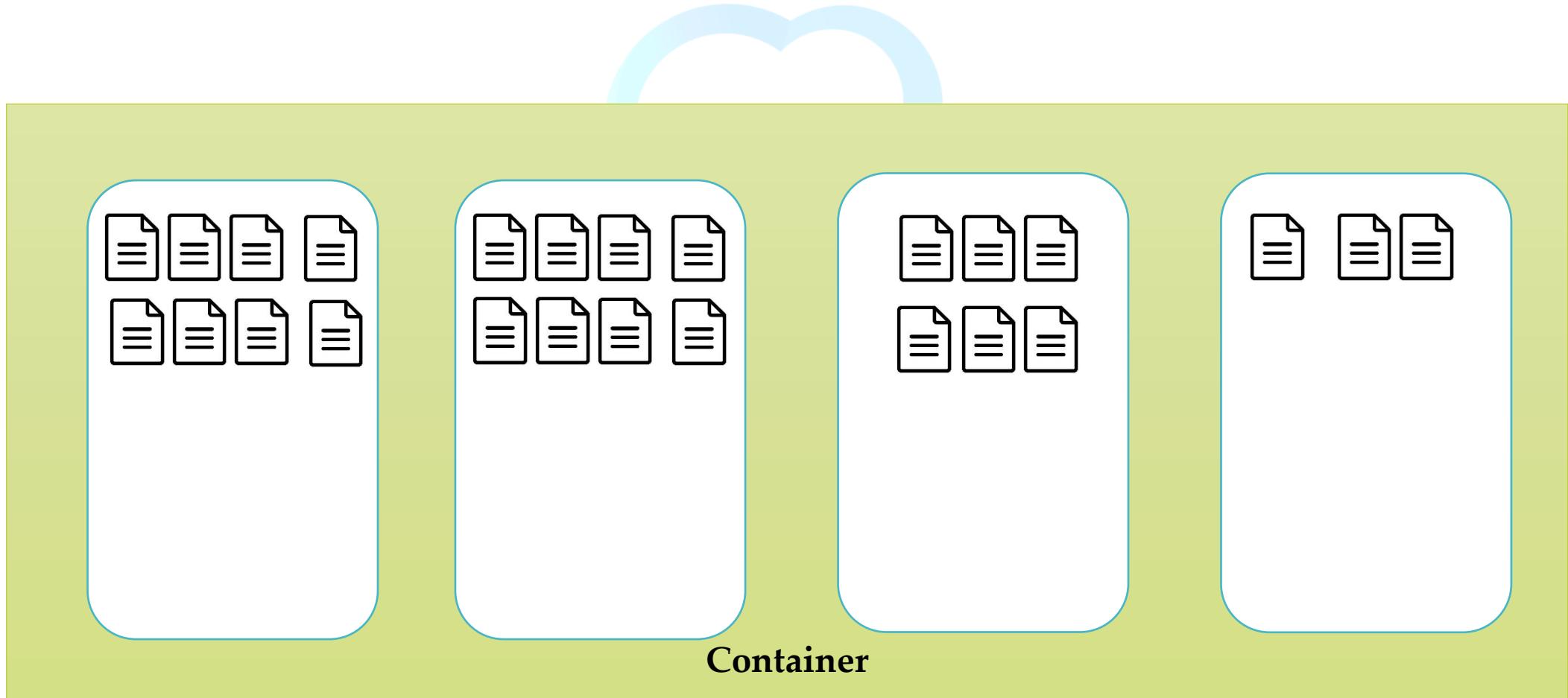
# Avoiding Hot Partitions



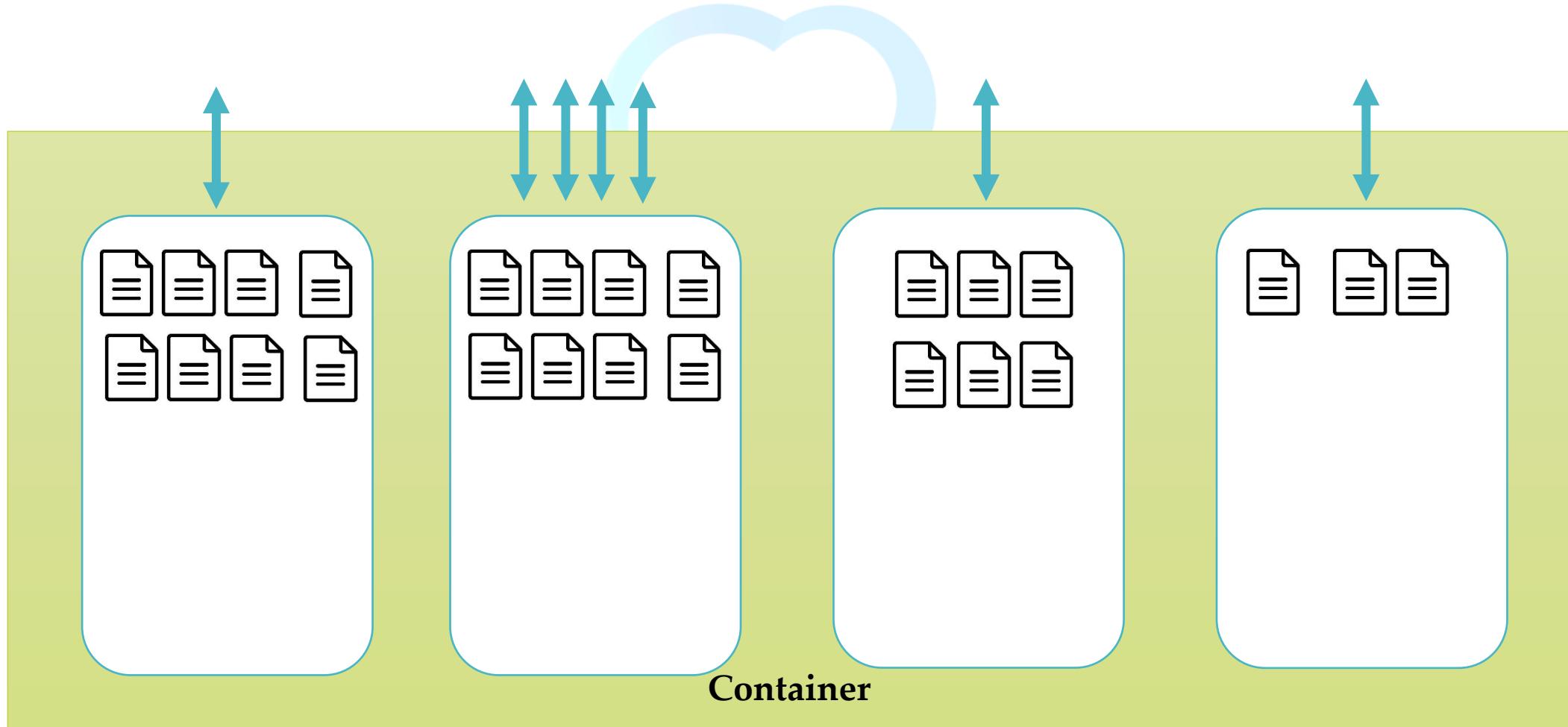
# Avoid Hot partitions on storage



# Avoiding Hot Partitions at store

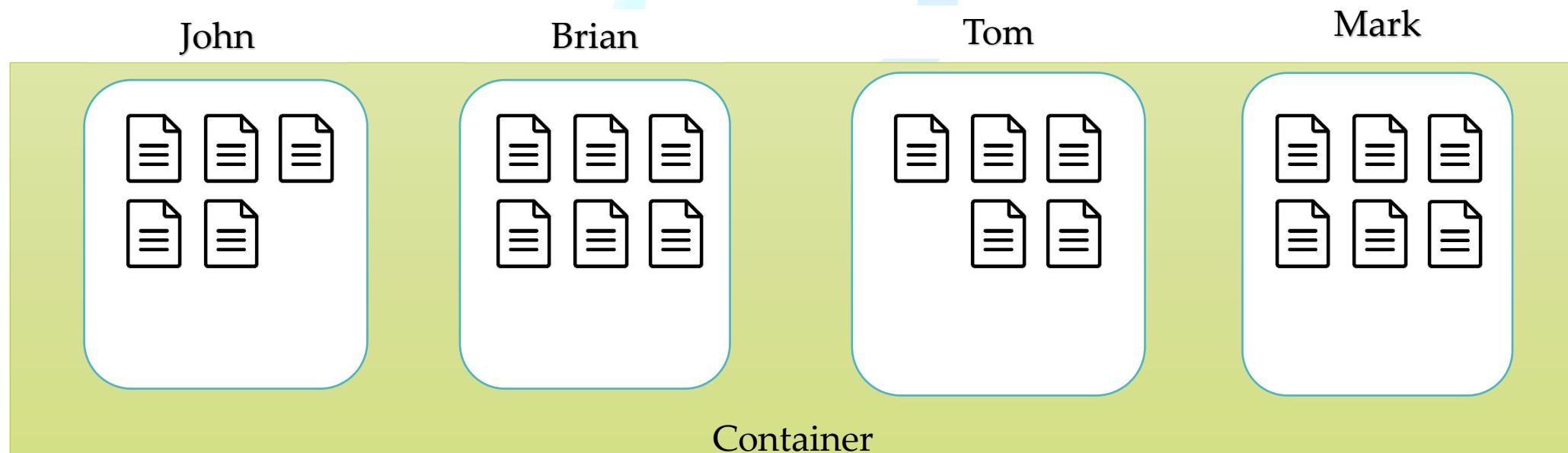


# Avoid Hot partitions on throughput



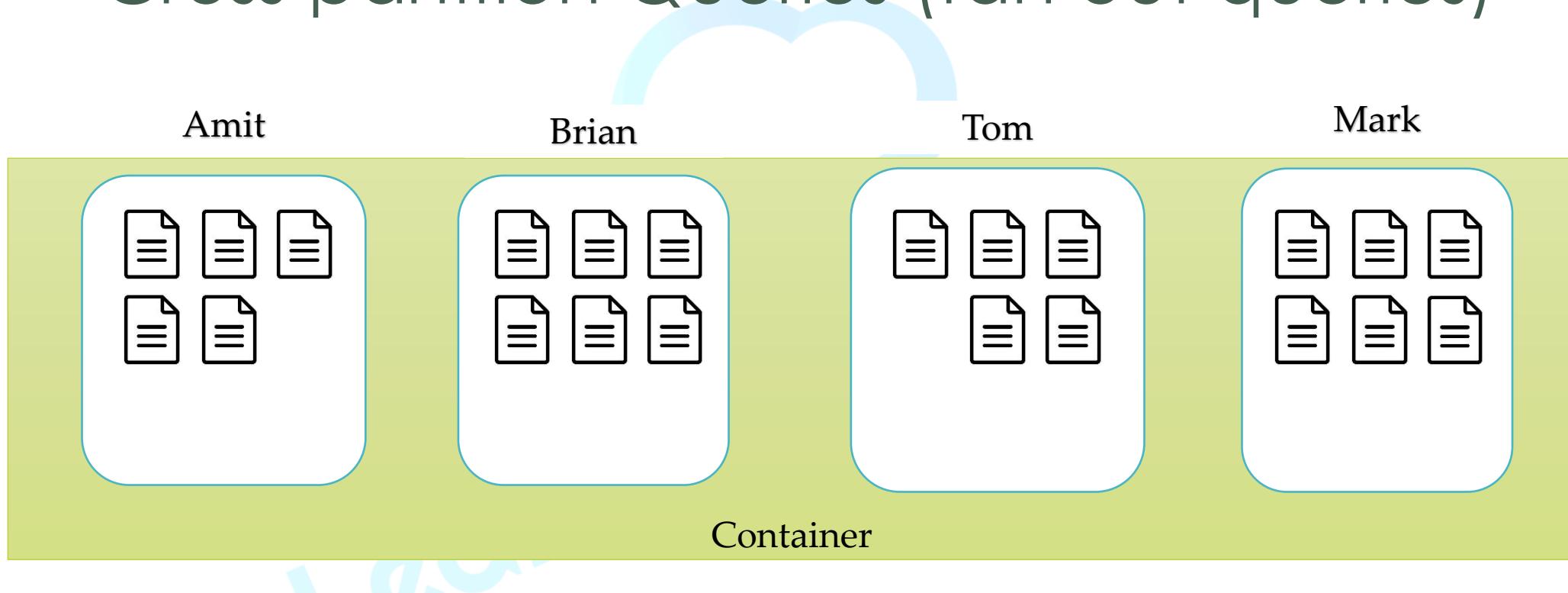
- **Partition key Bad choice:** Current time
- **Partition key Good choices:** User ID, Product ID

# Single partition Query



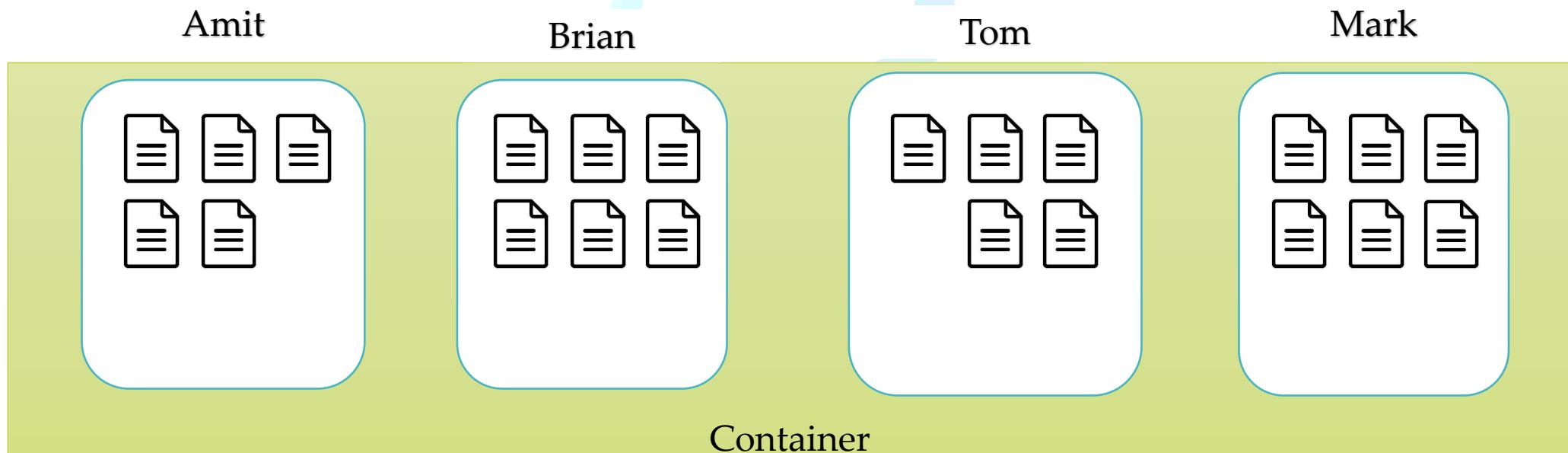
```
SELECT * FROM c WHERE c.username = 'Brian'
```

# Cross partition Queries (fan out queries)



```
SELECT * FROM c WHERE c.favoritecolor= 'Blue'
```

# Composite Key



Composite Key: CustomerName-mmddyyyy

# Choosing a Partition key

- Evenly distribute storage
  - Make sure you pick your partition key that doesn't result in hot spots within your applications
  - Have a high cardinality
  - Don't be afraid of choosing a partition key that has a large number of values
  - Example User Id & Product Id
- Evenly distribute requests.
  - RUs evenly distribute across all partitions.
  - Review where clause of top queries
- Consider document and partition limit while designing partition key.
  - Max document size – 2 MB
  - Max logical partition size – 20 GB

# Choosing a Partition key

Question: Your organization is planning to use Azure Cosmos DB to store vehicle telemetry data generated from millions of vehicles every second. Which of the following options for your Partition Key will optimize storage distribution?

Answer choices:

1. Vehicle model
2. Vehicle Identification Number (VIN) which looks like WDDEJ9EB6DA032037

# Choosing a Partition key

Question: Your organization is planning to use Azure Cosmos DB to store vehicle telemetry data generated from millions of vehicles every second. Which of the following options for your Partition Key will optimize storage distribution?

Answer choices:

- 1. **Vehicle model**

Most auto manufacturers only have a couple dozen models. This option is potentially the least granular, will create a fixed number of logical partitions, and may not distribute data evenly across all physical partitions.

- 2. **Vehicle Identification Number (VIN) which looks like WDDEJ9EB6DA032037**

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



# Automatic Indexing

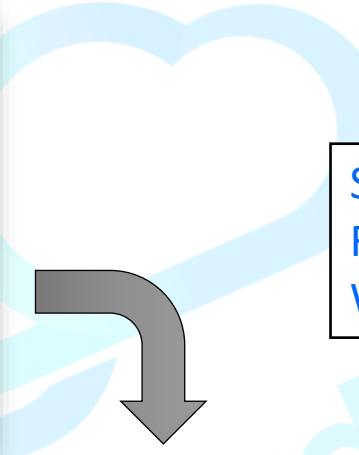
- Index all data without requiring Index management
- Every property of every record automatically index
- Index update synchronously as you create, update or delete items
- Not specific for SQL, but available for all APIs



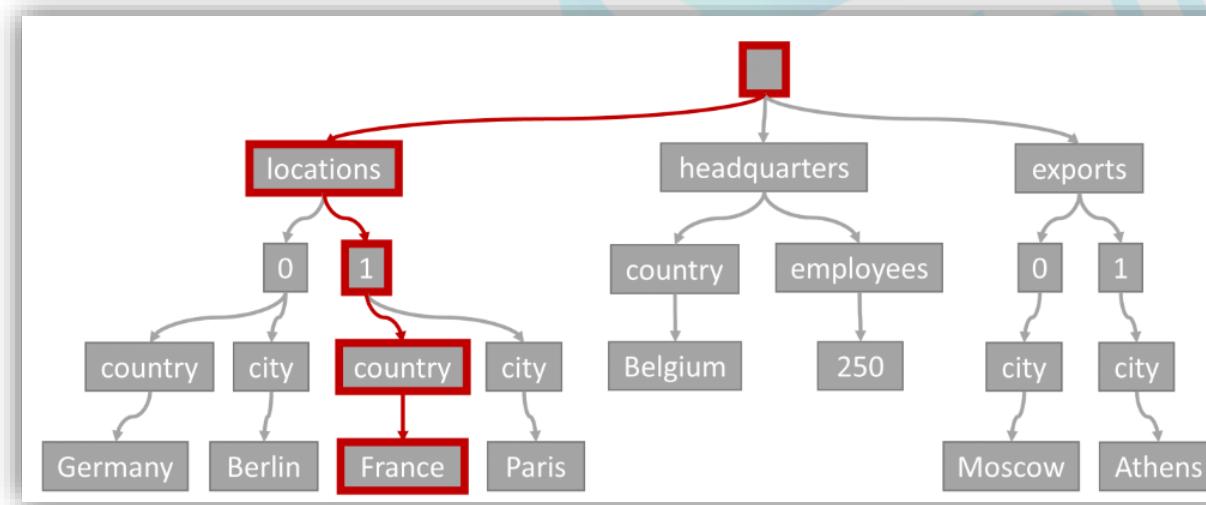
# Automatic Indexing

JSON

```
{  
  "locations": [  
    { "country": "Germany", "city": "Berlin" },  
    { "country": "France", "city": "Paris" }  
  ],  
  "headquarters": { "country": "Belgium", "employees": 250 },  
  "exports": [  
    { "city": "Moscow" },  
    { "city": "Athens" }  
  ]  
}
```



```
SELECT location  
FROM location IN company.locations  
WHERE location.country = 'France'
```



```
/locations/0/country: "Germany"  
/locations/0/city: "Berlin"  
/locations/1/country: "France"  
/locations/1/city: "Paris"  
/headquarters/country: "Belgium"  
/headquarters/employees: 250  
/exports/0/city: "Moscow"  
/exports/1/city: "Athens"
```



# Time-to-Live



## Time to Live (TTL)

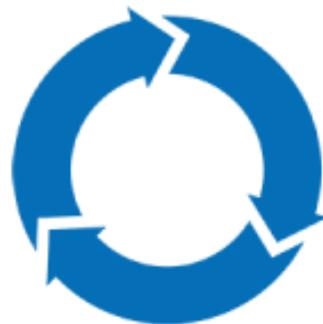
- You can set the expiry time for Cosmos DB data items
- Time to live value is configured in seconds.
- The system will automatically delete the expired items based on the TTL value
- Consume only leftover Request units
- Data deletion delay if not enough Request units
  - Though the data deletion is delayed, data is not returned by any queries (by any API) after the TTL has expired.

# Global Distribution benefits



## Performance

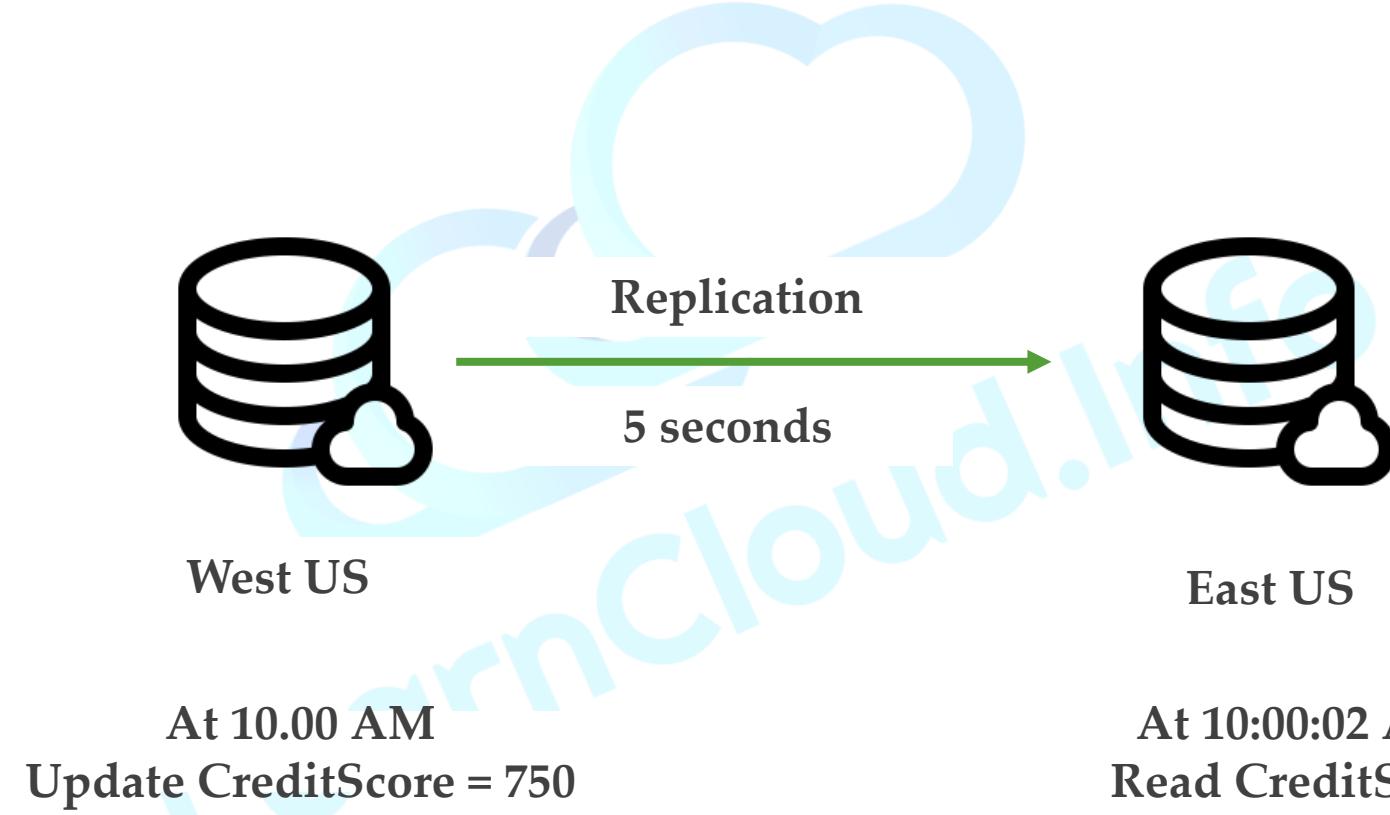
- Ensures high availability within a region
- Across regions, brings data closer to the consumer.



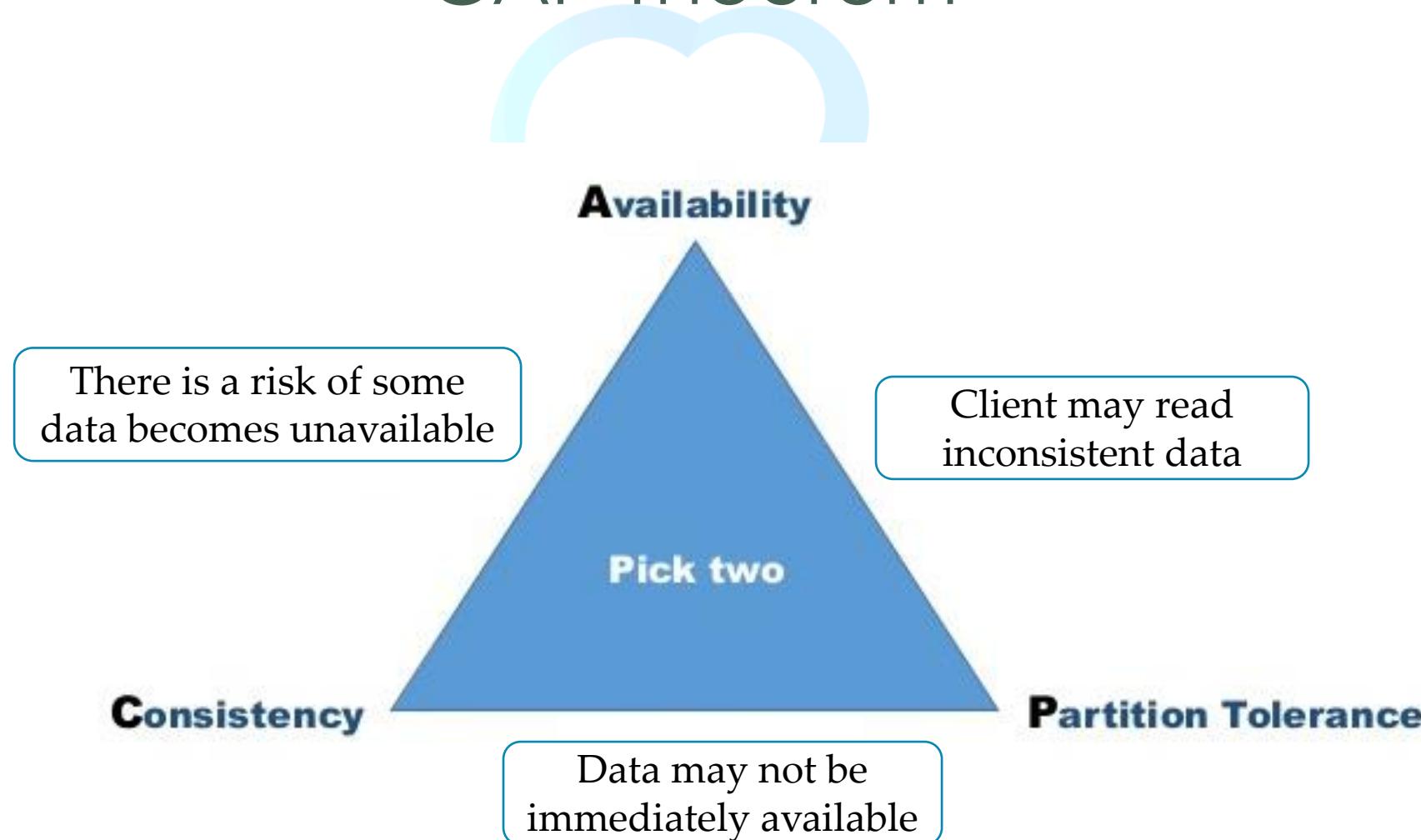
## Business continuity

- In the event of major failure or natural disaster

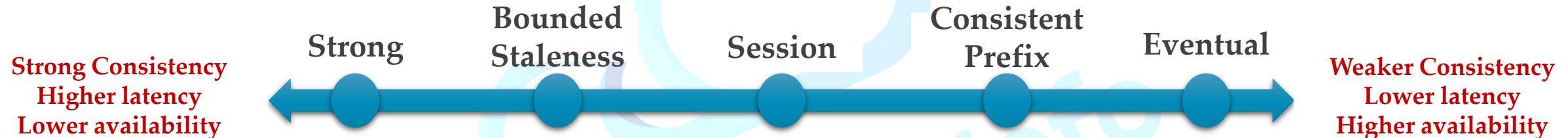
# Data consistency



# CAP Theorem



# Five consistency Levels



**Strong:** No dirty reads, high latency, cost highest, closest to RDBMS

**Bounded staleness:** Dirty reads possible, bounded by time and updates

**Session:** No dirty reads for writers (within same session), dirty read possible for other users

**Consistency prefix:** Dirty reads possible but sequence maintain, reads never see out-of-order writes

**Eventual:** No guaranteed order, but eventually everything gets in order

# Setting the consistency level

**Set default for entire account**

Can be changed any time

**Override at request level**

Any request can weaken the default consistency level

**Strong Consistency**  
Higher latency  
Lower availability



**Weaker Consistency**  
Lower latency  
Higher availability

# Azure CLI



```
az cosmosdb create --name
    --resource-group
    [--capabilities]
    [--default-consistency-level {BoundedStaleness, ConsistentPrefix, Eventual, Session, Strong}]
    [--enable-automatic-failover {false, true}]
    [--enable-multiple-write-locations {false, true}]
    [--enable-virtual-network {false, true}]
    [--ip-range-filter]
    [--kind {GlobalDocumentDB, MongoDB, Parse}]
    [--locations]
    [--max-interval]
    [--max-staleness-prefix]
    [--subscription]
    [--tags]
    [--virtual-network-rules]
```



# Azure CLI – Create Database example

```
Create a SQL API database and container

# Generate a unique 10 character alphanumeric string to ensure unique resource names
uniqueId=$(env LC_CTYPE=C tr -dc 'a-z0-9' < /dev/urandom | fold -w 10 | head -n 1)

# Variables for SQL API resources
resourceGroupName="Group-$uniqueId"
location='westus2'
accountName="cosmos-$uniqueId" #needs to be lower case
databaseName='database1'
containerName='container1'

# Create a resource group
az group create -n $resourceGroupName -l $location

# Create a Cosmos account for SQL API
az cosmosdb create \
    -n $accountName \
    -g $resourceGroupName \
    --default-consistency-level Eventual \
    --locations regionName='West US 2' failoverPriority=0 isZoneRedundant=False \
    --locations regionName='East US 2' failoverPriority=1 isZoneRedundant=False

# Create a SQL API database
az cosmosdb sql database create \
    -a $accountName \
    -g $resourceGroupName \
    -n $databaseName
```



# Azure CLI – Example question

You are a data engineer for your company. You use the following Azure CLI commands to create an Azure Cosmos DB account. You plan to use this account to store sales data.

```
az cosmosdb create --resource-group 'sales-rg' --name 'sales' --kind GlobalDocumentDB \
--locations regionName="South Central US" failoverPriority=0 \
--locations regionName="North Central US" failoverPriority=1 \
--default-consistency-level "Strong" --enable-multiple-write-locations true
```

You need to answer questions regarding sales data queries and updates.

For each of the following statements, select Yes if the statement is true. Otherwise, select No.

| Statement  |
|--|
|  You can query data by using Gremlin API.   |
|  A client can see partial writes of a sales data record by default.                 |
|  A client can set the consistency level to Eventual Consistency at connection time. |
|  A client can set a different consistency level during each request to sales data.  |