Azure Cosmos DB

Lessons learnt from building a globally distributed database from the ground up

Dharma Shukla, @dharmashukla, Distinguished Engineer, Microsoft

Outline

- Background
- Requirements
- Overview of Capabilities
- System Design
- Q & A

Cosmos DB Evolution

- Originally started to address the problems faced by large scale apps inside Microsoft
- Built from the ground up for the cloud
- Used extensively inside Microsoft
- One of the fastest growing services on Azure

2010 — Project Florence

2014 — DocumentDB

2015

2017 — Cosmos DB

Requirements

Turnkey global distribution

Guaranteed low latency at the 99th percentile, worldwide

Guaranteed high availability within region and globally

Guaranteed consistency

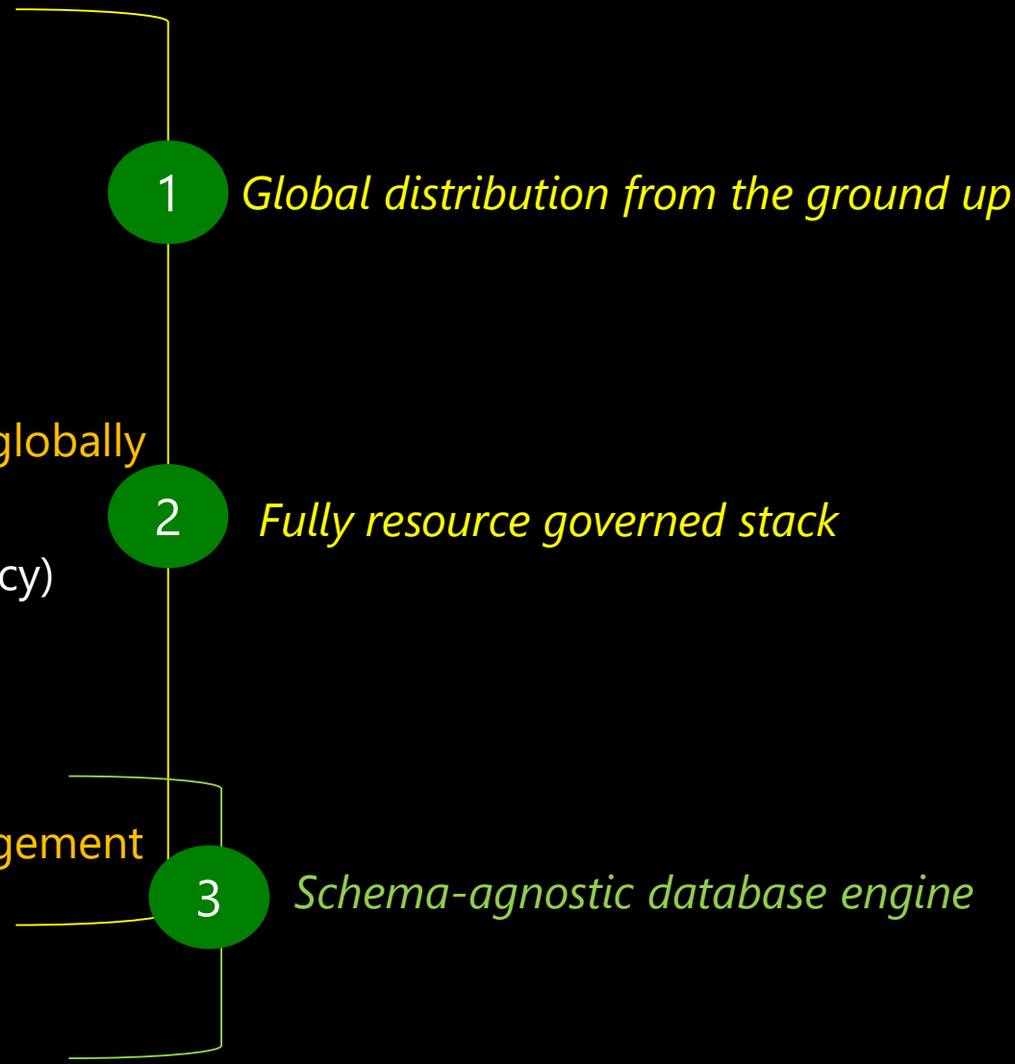
Elastically scale throughput and storage, any time, on-demand, globally

Comprehensive SLAs (availability, latency, throughput, consistency)

Operate at low cost

Iterate & query without worrying about schemas & index management

Provide a variety of data model and API choices



Capabilities

Global distribution from the ground-up

- Cosmos DB as a foundational Azure service
  - Available in all Azure regions by default, including sovereign/government clouds
- Automatic multi-region replication
  - Associate **any** number of regions with your database account
  - Policy based geo-fencing
- Multi-homing APIs
  - Apps don’t need to be redeployed during regional failover
- Allows for dynamically setting priorities to regions
  - Simulate regional disaster via API
  - Test the end to end availability for the entire app (beyond just the database)
- First to offer comprehensive SLA for latency, throughput, availability and consistency

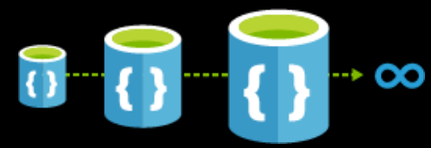
Guaranteed low latency @ P99

	Reads (1KB)	Indexed writes (1KB)
50th	<2ms	<6ms
99th	<10ms	<15ms

- Globally distributed with reads and writes served from local region
- Write optimized, latch-free database engine designed for SSDs and low latency access
- Synchronous and automatic indexing at sustained ingestion rates

Elastically scalable storage

- System designed to independently scale storage and throughput
- Transparent server side partition management and routing
- Automatically indexed SSD storage
- Automatic global distribution of data across any number of Azure regions
- Optionally evict old data using built-in support for TTL



Scaling throughput worldwide

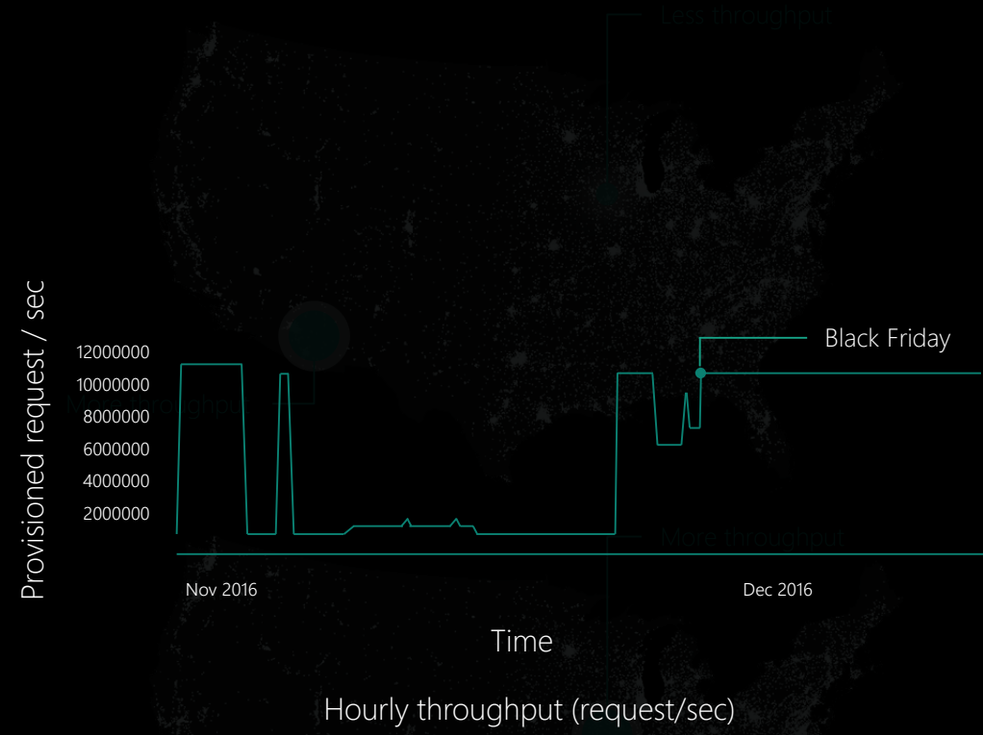
Elastically scalable throughput, globally

Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions

Customers pay by the hour for the provisioned throughput

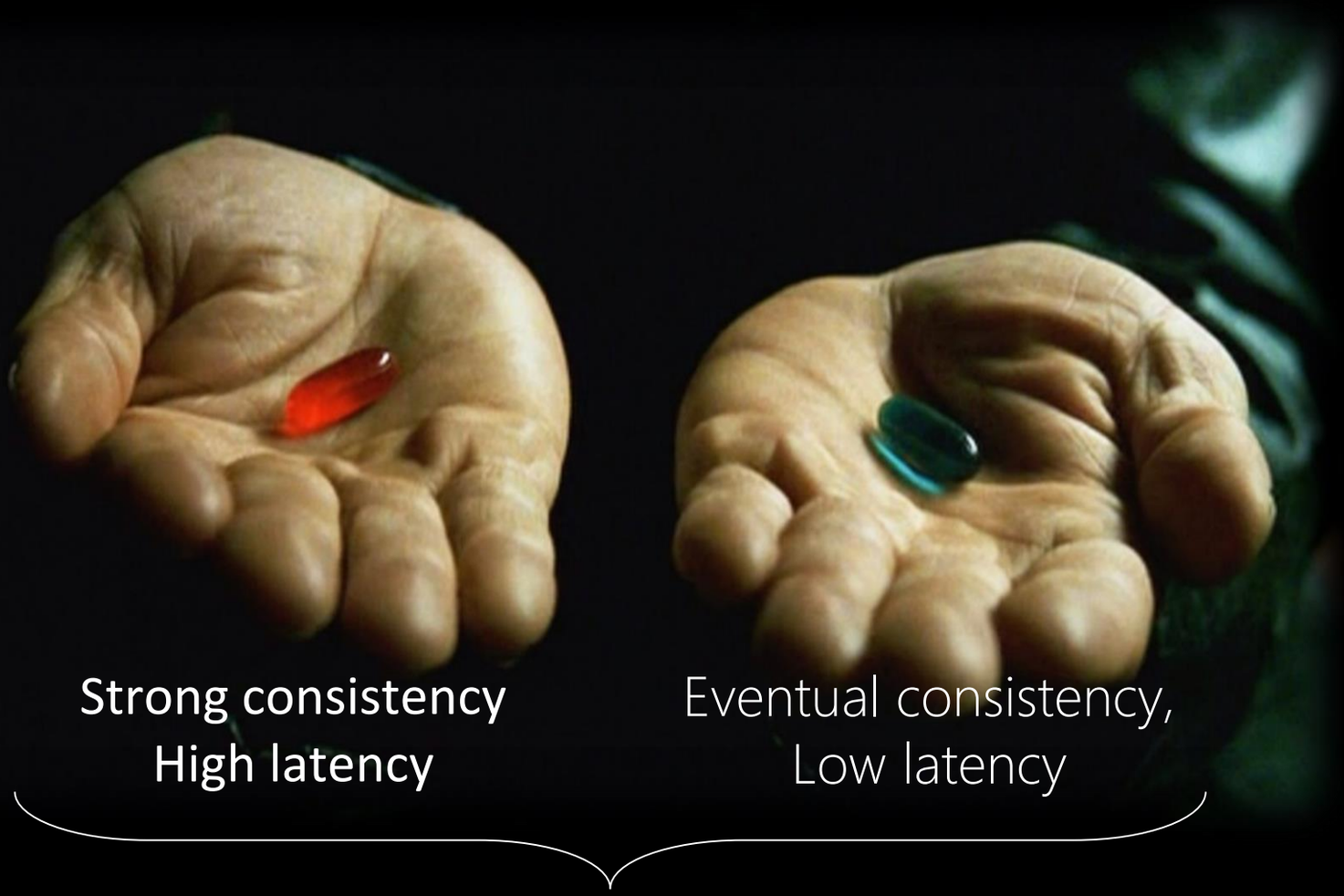
Transparent server side partition management and routing

Support for requests/sec and requests/min for different workloads



Programmable Data Consistency

Choice for most distributed apps



Well-defined consistency models

Intuitive programming model

5 Well-defined, consistency models

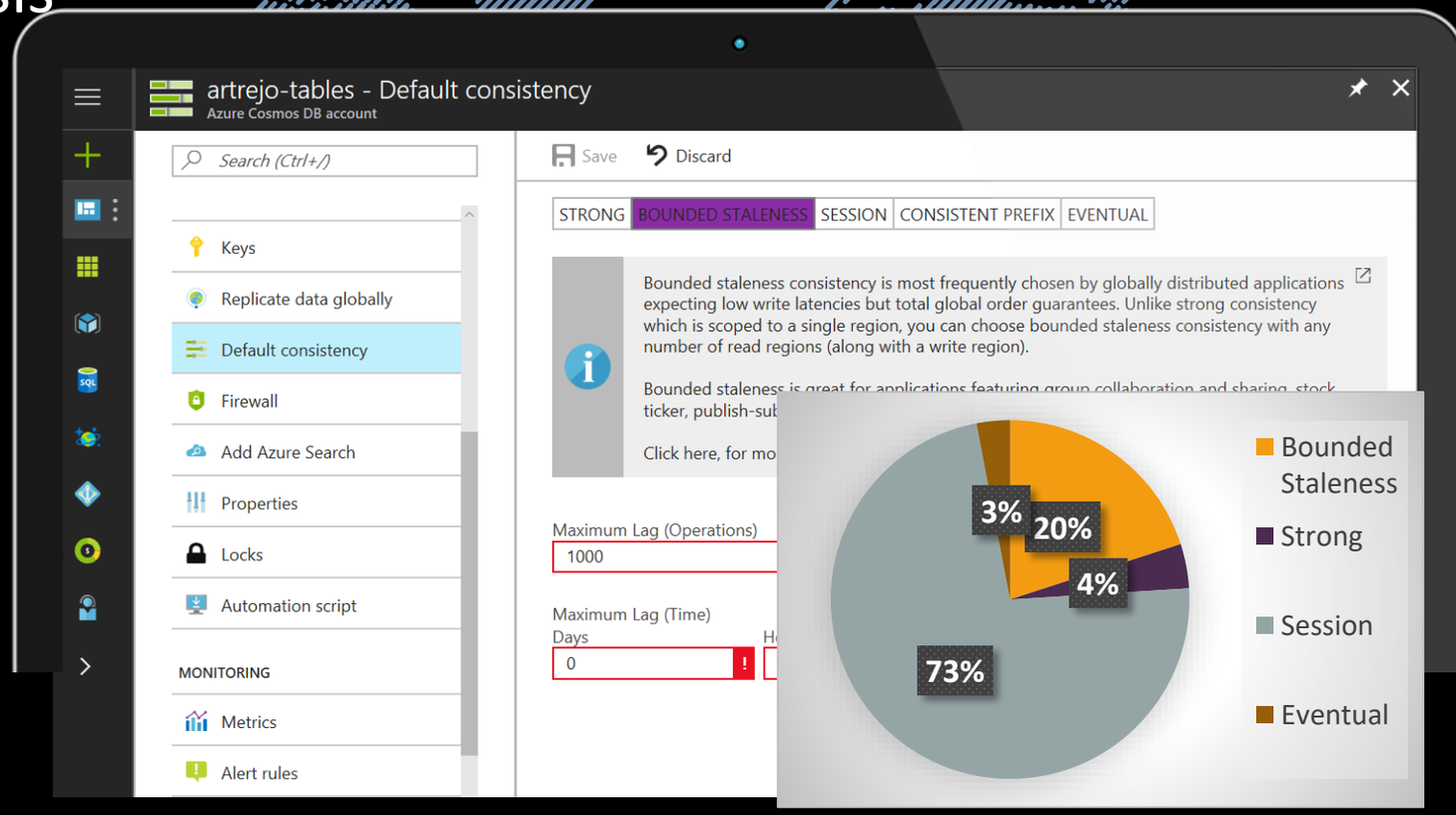
Overridable on a per-request basis

Clear tradeoffs

Latency

Availability

Throughput



SLA for Azure Cosmos DB

Last updated: May 2017

Azure Cosmos DB is Microsoft's globally distributed multi-model database. It offers turnkey global distribution across any number of Azure regions by transparently scaling and replicating your data wherever your users are. The service offers 99.99% guarantees for availability, throughput, latency, and consistency.

⊕ Introduction

⊕ General Terms

⊖ SLA details

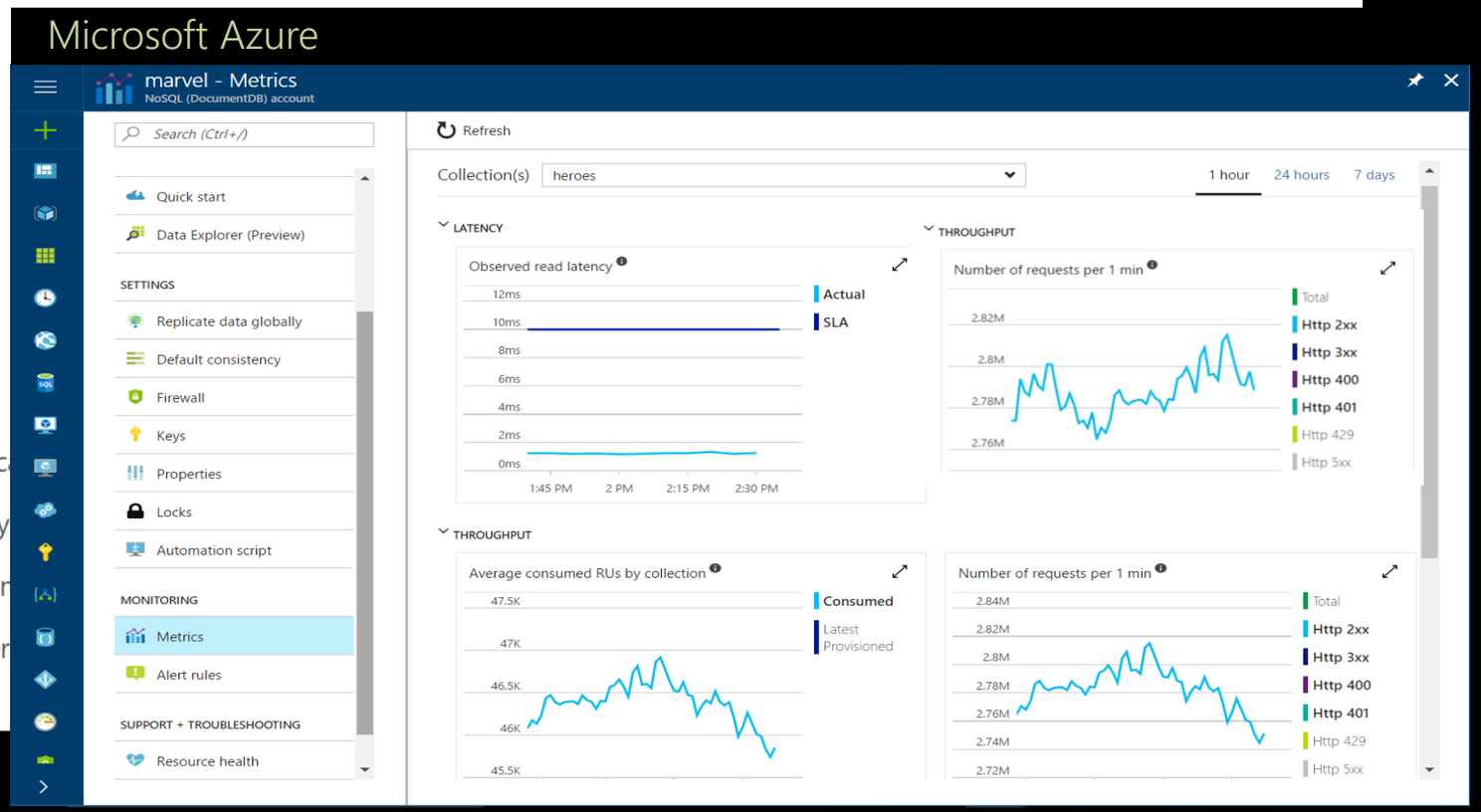
Additional Definitions

"**Collection**" is a container of JSON documents, and a unit of sc

"**Consumed RUs**" is the sum of the Request Units consumed by

"**Database Account**" is the top-level resource of the Azure Cosm

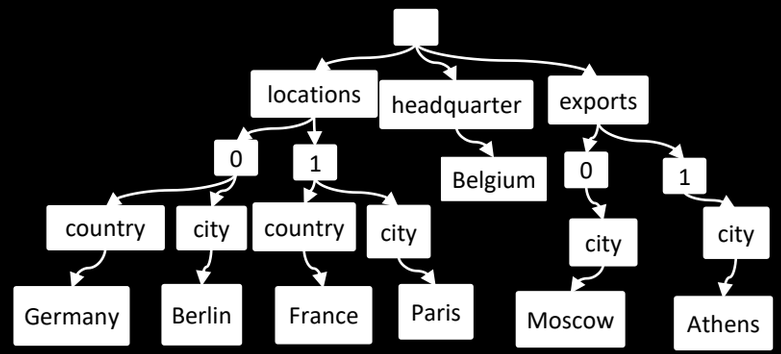
"**Failed Requests**" are requests within Total Requests that either
the table below.



Schema agnostic indexing

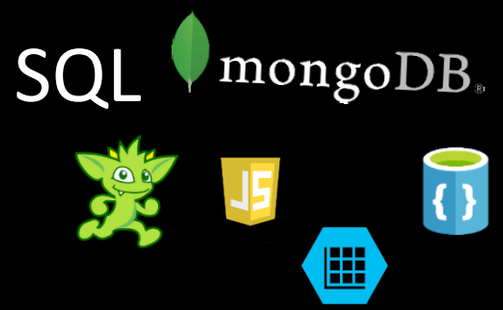
- At global scale, schema/index management is hard
- Automatic and synchronous indexing of all ingested content - hash, range, geo-spatial, and columnar
  - No schemas or secondary indices ever needed
- Resource governed, write optimized database engine with latch free and log structured techniques
- Online and in-situ index transformations

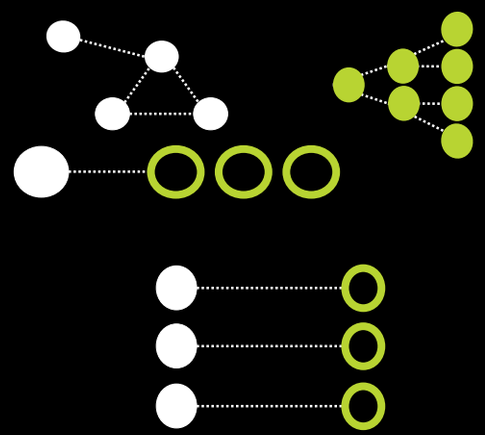
```
{
    "locations":
    [
      { "country": "Germany", "city": "Berlin" },
      { "country": "France", "city": "Paris" }
    ],
    "headquarter": "Belgium",
     "exports":[{ "city": "Moscow" },{ "city": "Athens"}]
};
```



Native support for multiple data models

- Database engine operates on atom-record-sequence (ARS) based type system
  - All data models are translated to ARS
- API and wire protocols are supported via extensible modules
- Instance of a given data model can be materialized as trees
- Graph, documents, key-value, column-family, … more to come

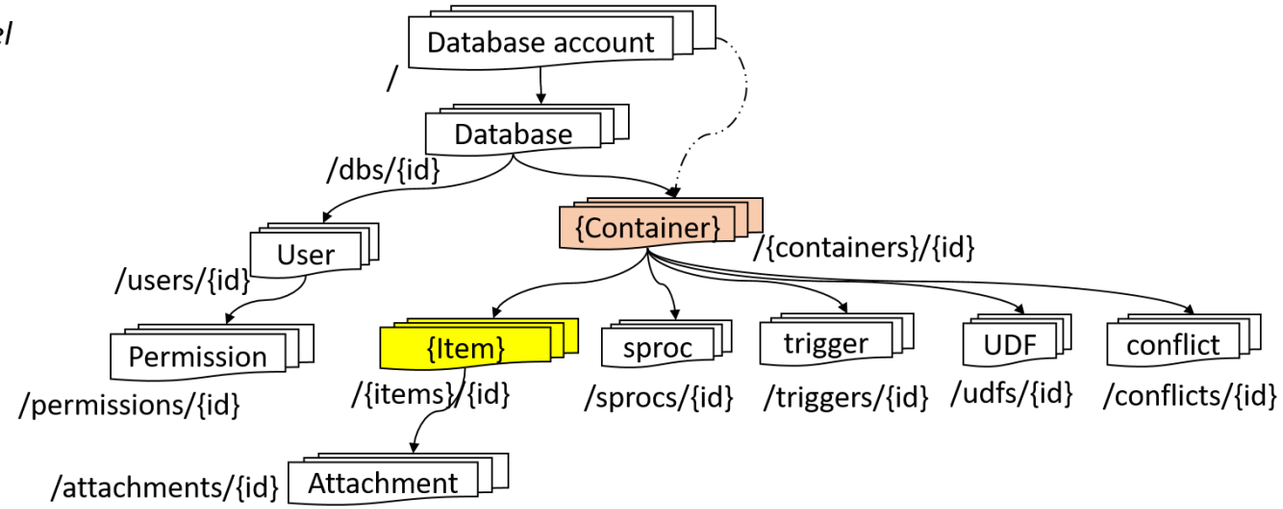




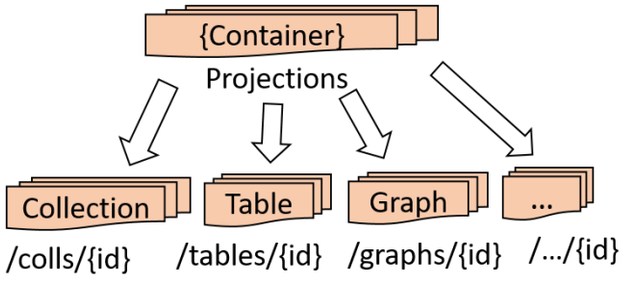
System Design

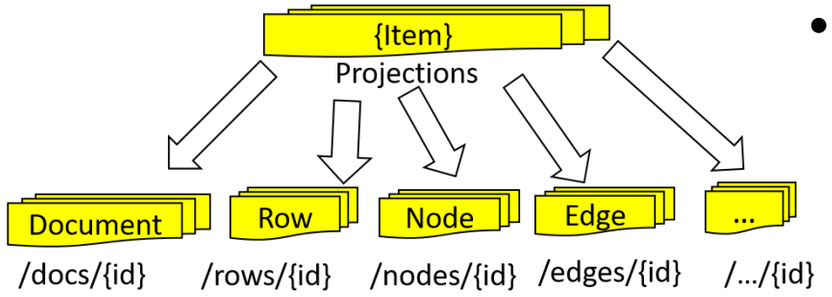
Resource Model

Resource Model



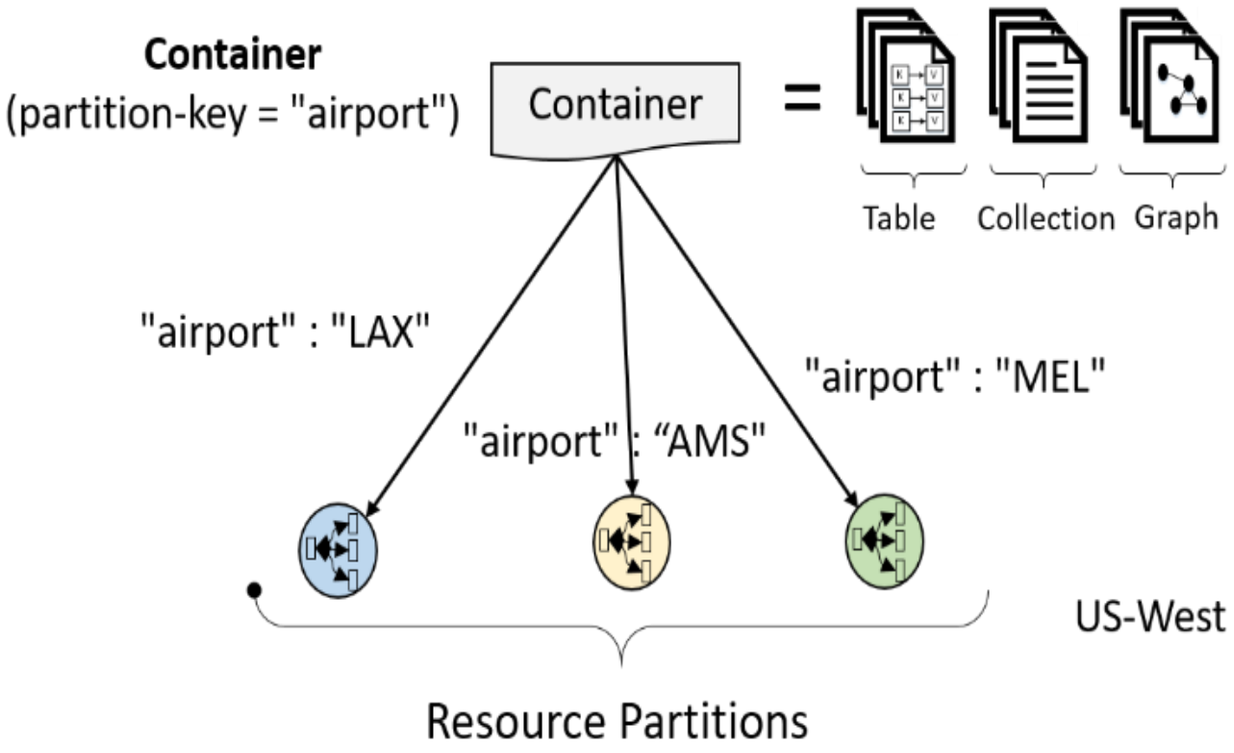
Depending on the API, container and item resources are projected as specialized resource types





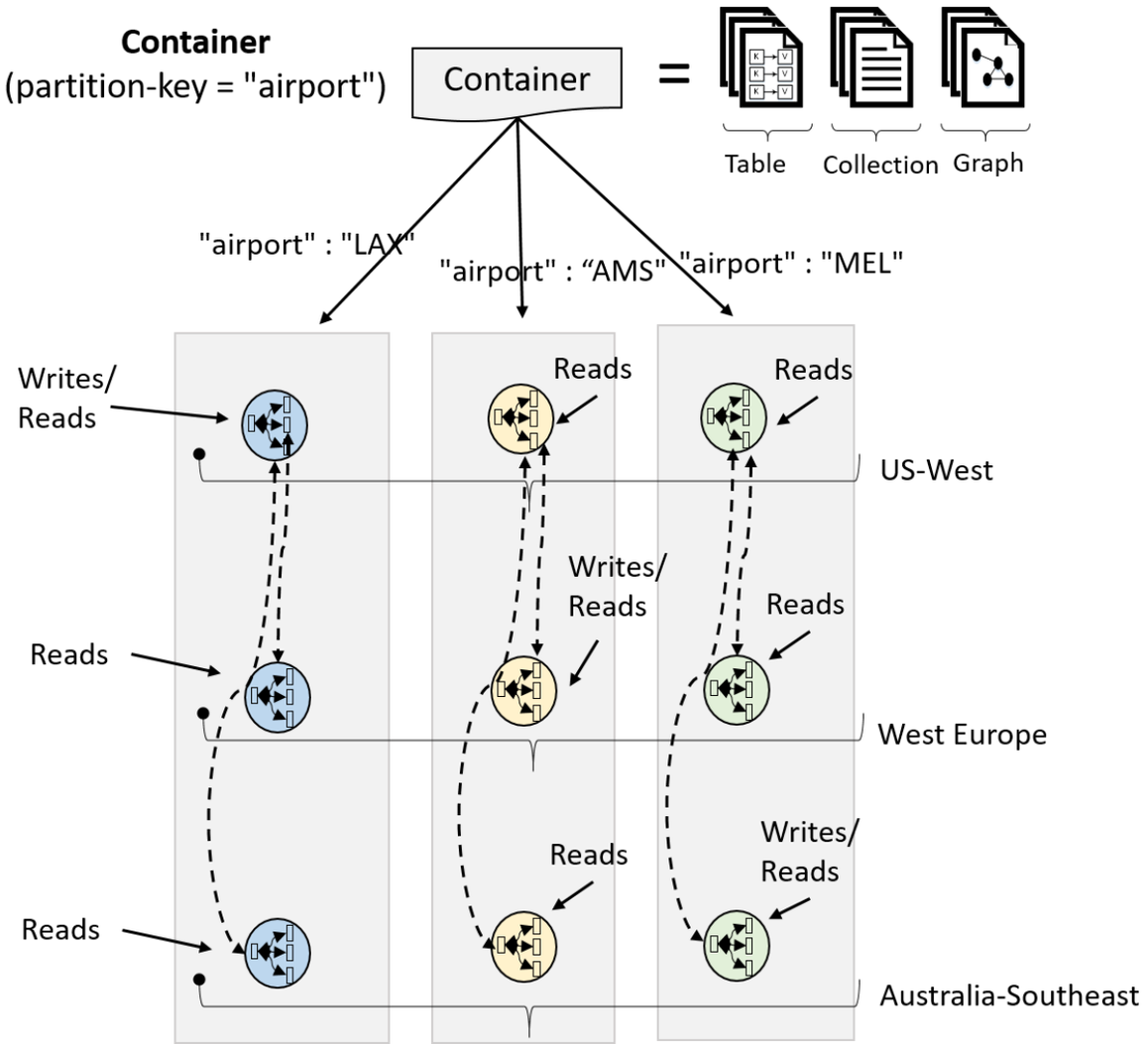
- Single system image of globally distributed, URI addressable logical resources
- Consistent, hierarchical overlay over horizontally partitioned entities
- Extensible custom projections

Horizontal partitioning



- All resources are horizontally partitioned
- Resource Partition
  - Consistent, highly available and resource governed, coordination primitive
  - Uniquely belongs to a tenant
- Partition management is transparent and made highly responsive

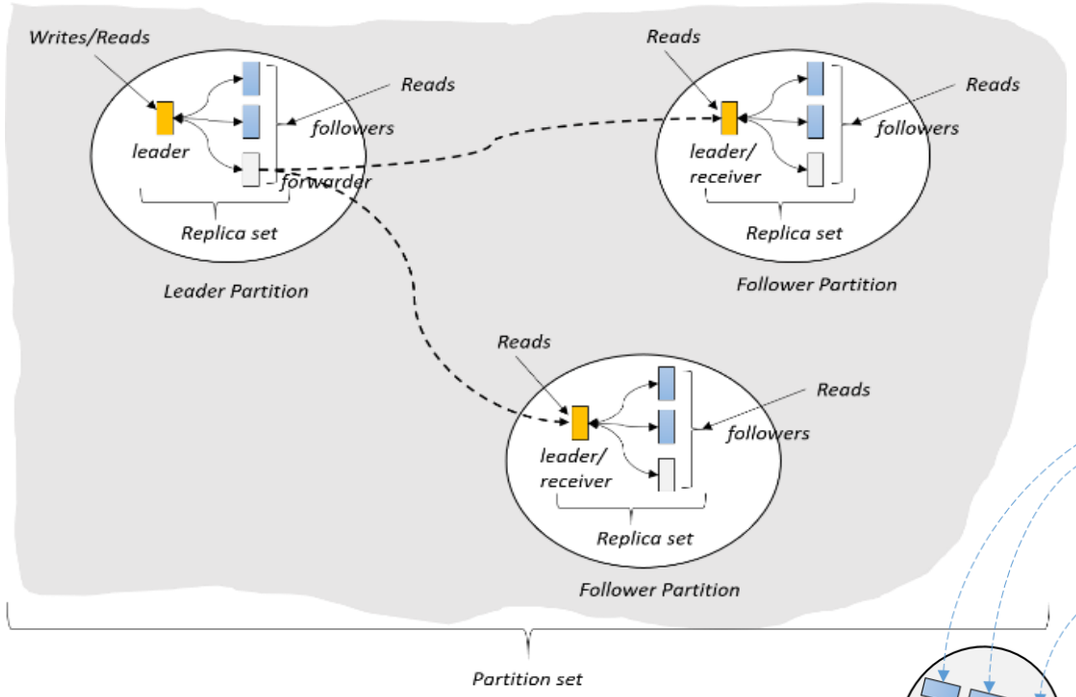
Global distribution

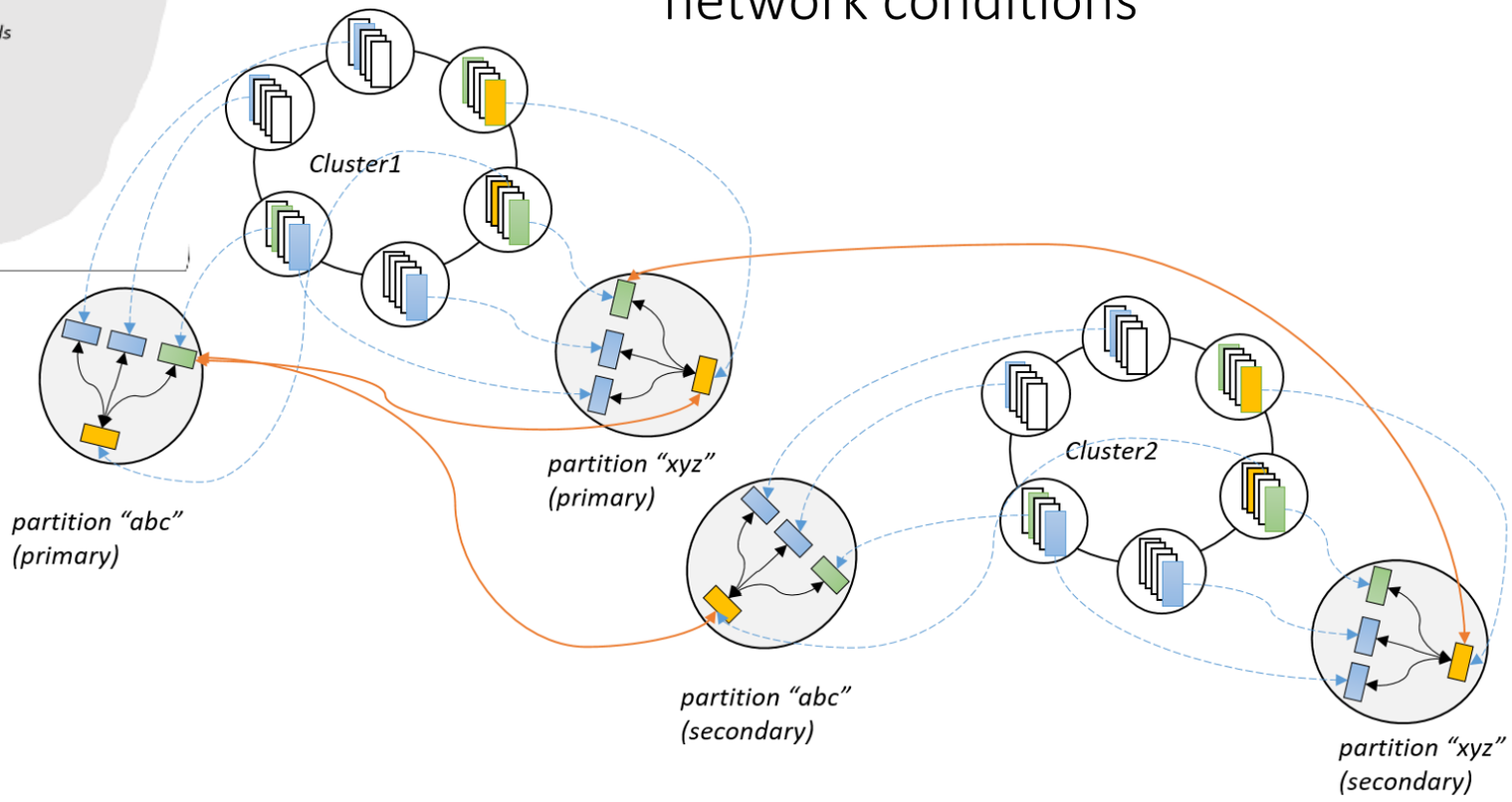


- All resources are horizontally partitioned and vertically distributed
- Nested consensus
- Distribution can be within a cluster, x-cluster, x-DC or x-region

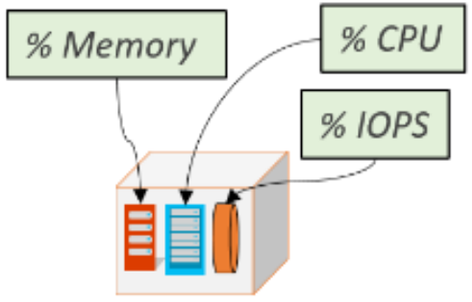
Partition-sets

- Dynamic allocations of system resources
- Dynamic replication topologies (e.g. tree, chain, hub-spoke) based on consistency level and network conditions



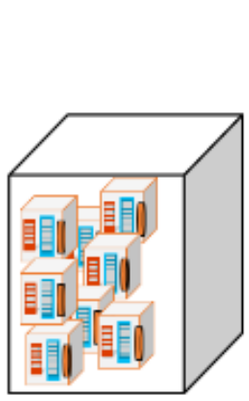


Resource Governed Stack

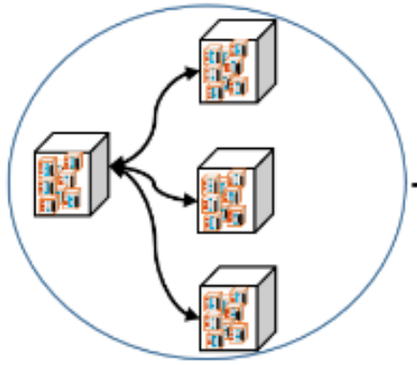


Request Unit (RU)

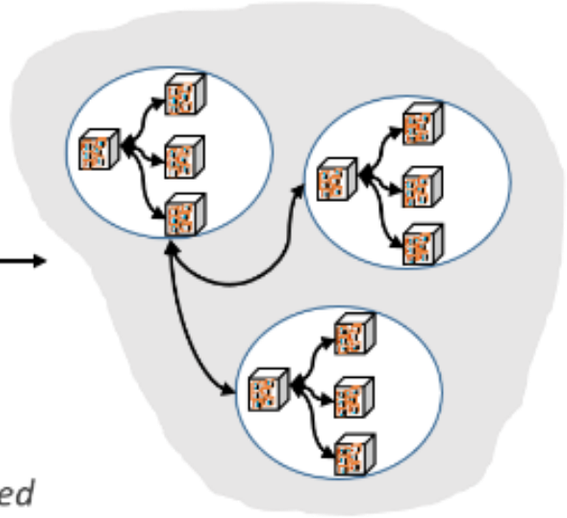
Database operations consume RUs



Replica is allotted a budget of RUs



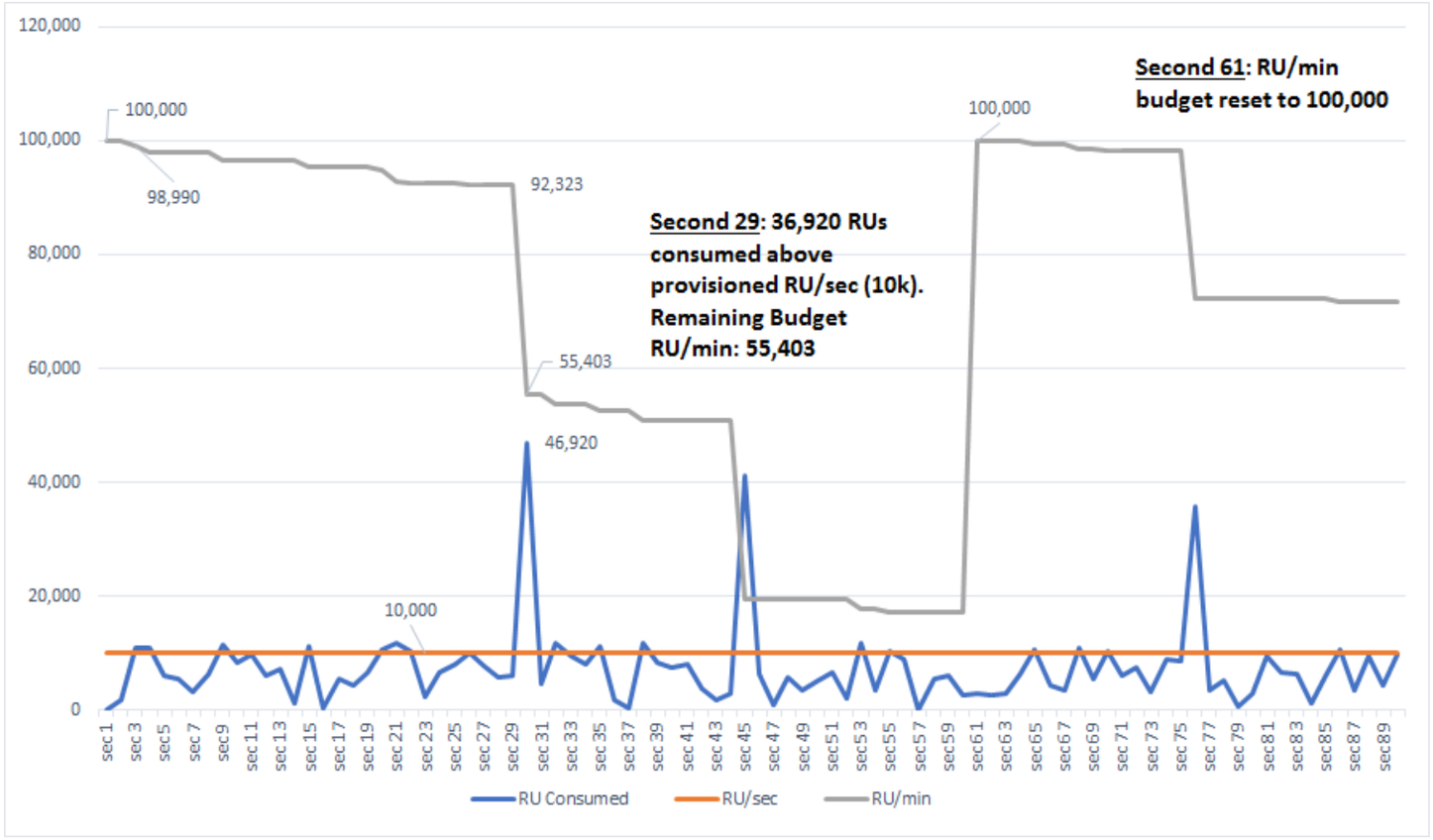
Resource partition is allotted a budget of RUs



Partition set is also allotted a budget of RUs

- Replica density, COGS and SLA, all depend on stringent resource governance across the entire stack
- Request Unit (RU)
  - Rate based currency
  - Normalized across various access methods
  - Available for second (RU/s) and minute (RU/m) granularities
- All engine operations are finely calibrated

Fine-grained Resource Governance



This is how it works:

- At a container level, for every 100 RU/sec provisioned, a customer can provision an additional 10kRU/min
- RUs consumed above provisioned RU/sec is removed from RU/min budget
- Provisioning per minute provides peace of mind and predictable performance to spikes and unexpected throughput needs

Savings (example)

- Customers save 73% in provisioning cost

Next steps & references

- Getting Started
  - cosmosdb.com
  - portal.azure.com
  - aka.ms/cosmosdb
  - Downloadable service emulator (aka.ms/CosmosDB-emulator)

- Technical Overview -> https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/
- Schema Agnostic Indexing, VLDB 2015 -> http://www.vldb.org/pvldb/vol8/p1668-shukla.pdf

- Follow #CosmosDB on Twitter
  - @azurecosmosdb
  - @dharmashukla

Azure Cosmos DB

We are just getting started…

We are Hiring