Microsoft

GPS Data/AI Strategy FY23

Delivered by CSA Team

Franck Gaillard
Cloud Solution Architect
Data AI
frgail@microsoft.com

Narjes Majdoub
Cloud Solution Architect
Data AI
nmajdoub@microsoft.com

Ali Bouhaddou
Cloud Solution Architect
Data Analytics
albouhad@microsoft.com

Frederic Gisbert
Cloud Solution Architect
Data Analytics
frgisber@microsoft.com

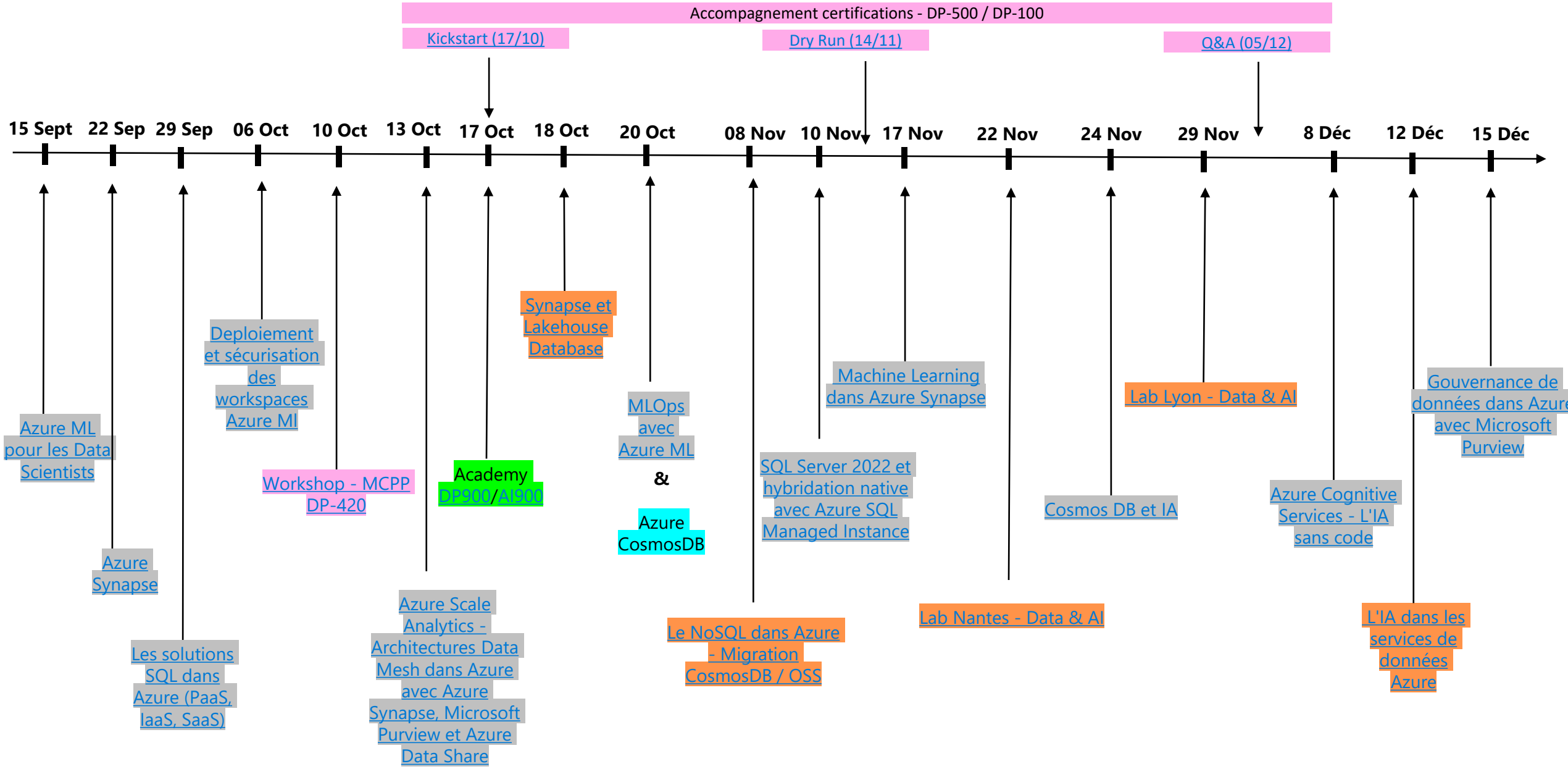
Azure Data & AI technical intensity plan

- From June 2022 to June 2023
- Focus on "**Azure Data & AI**" tech intensity

- Many content, from **L100** Beginner to **L400** Expert level:
  - **Academy** L100
  - **Webinar** L200/L300
  - **Workshop** L300/L400
  - **Certification kickstart** L300/L400
  - **Openhack / Microhack** L400

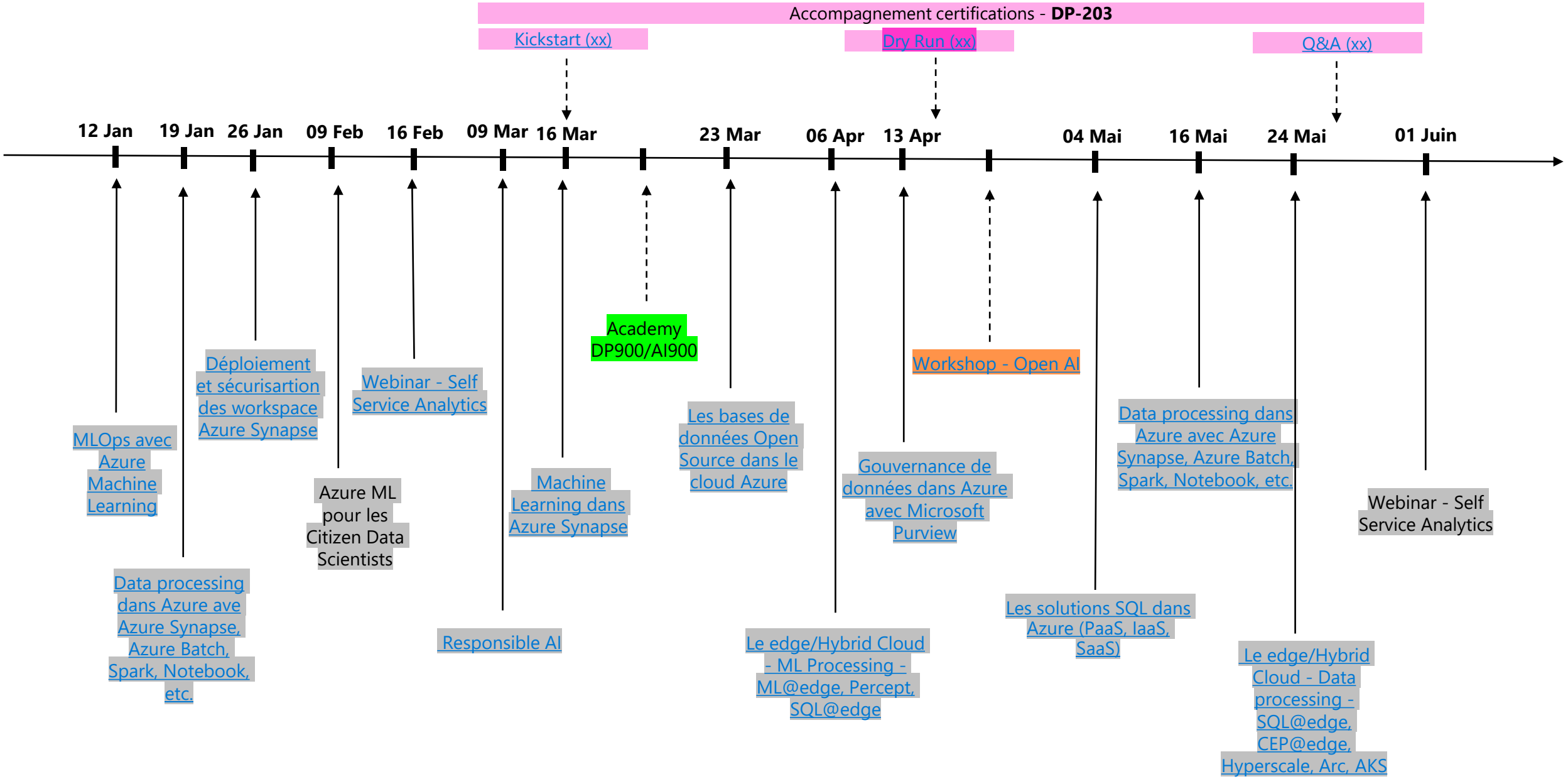
Data & AI events timeline – H1

Webinar/Academy- L 200/300
Workshop/ Openhack/ Certifications – L 300/400



Data & AI events timeline – H2

Webinar/Academy- L 200/300
Workshop/ Openhack/ Certifications – L 300/400



Liste des évènements de type Webinar 2H

Event Webinar (Les jeudis de la Data & AI) - L200/300	Date	Duration (min)	Link
Azure Machine Learning pour les Data Scientists	15/09/2022	120	https://msevents.microsoft.com/event?id=2454281594
Azure Synapse	22/09/2022	120	https://msevents.microsoft.com/event?id=857781749
Les solutions SQL dans Azure (PaaS, IaaS, SaaS)	29/09/2022	120	https://msevents.microsoft.com/event?id=502366997
Déploiement et sécurisation des workspaces Azure Machine learning	06/10/2022	120	https://msevents.microsoft.com/event?id=1505714138
Azure Scale Analytics - Architectures Data Mesh dans Azure avec Azure Synapse, Microsoft Purview et Azure Data Share	13/10/2022	120	https://msevents.microsoft.com/event?id=139685175
MLOps avec Azure Machine Learning	20/10/2022	120	https://msevents.microsoft.com/event?id=1245885767
SQL Server 2022 et hybridation native avec Azure SQL Managed Instance	10/11/2022	120	https://msevents.microsoft.com/event?id=145826476
Machine Learning dans Azure Synapse Analytics	17/11/2022	120	https://msevents.microsoft.com/event?id=3637723312
Azure Cosmos DB et IA	24/11/2022	120	https://msevents.microsoft.com/event?id=2646013445
Azure et les Services Cognitifs	08/12/2022	120	https://msevents.microsoft.com/event?id=3772037220
La gouvernance de données dans Azure avec Microsoft Purview	15/12/2022	120	https://msevents.microsoft.com/event?id=1499560981
MLOps avec Azure Machine Learning	12/01/2023	120	https://msevents.microsoft.com/event?id=4115194515
Data processing dans Azure ave Azure Synapse, Azure Batch, Spark, Notebook, etc.	19/01/2023	120	https://msevents.microsoft.com/event?id=1537241181
Déploiement et sécurisation des workspace Azure Synapse	26/01/2023	120	https://msevents.microsoft.com/event?id=1806467748
Azure Machine Learning pour les Citizen Data Scientists	09/02/2023	120	En cours
PowerBI - Self Service Analytics	16/02/2023	120	https://msevents.microsoft.com/event?id=1401519679
L'IA responsable avec Azure machine learning	09/03/2023	120	https://msevents.microsoft.com/event?id=2072953112
Machine Learning dans Azure Synapse Analytics	16/03/2023	120	https://msevents.microsoft.com/event?id=3413014857
Les bases de données Open Source dans le cloud Azure	23/03/2023	120	https://msevents.microsoft.com/event?id=2727487131
Hybridation des services de Machine Learning Azure	06/04/2023	120	https://msevents.microsoft.com/event?id=1624914222
La gouvernance de données dans Azure avec Microsoft Purview	13/04/2023	120	https://msevents.microsoft.com/event?id=3909342839
Les solutions SQL dans Azure (PaaS, IaaS, SaaS)	04/05/2023	120	https://msevents.microsoft.com/event?id=1162207895
Data processing dans Azure ave Azure Synapse, Azure Batch, Spark, Notebook, etc.	16/05/2023	120	https://msevents.microsoft.com/event?id=3517068442
Hybridation des services de données Azure	24/05/2023	120	https://msevents.microsoft.com/event?id=2996507398
Self Service Analytics	01/06/2023	120	En cours

Liste des évènements de type Workshop/Prepa Cert/Academy

Event Workshop L300/400	Date	Duration (min)	Link
Synapse et Lakehouse Database	18/10/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdURE1RMVgwTDNISTE1TDFYSDVLR0cyS1kwWS4u
Le NoSQL dans Azure - Migration CosmosDB / OSS	08/11/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdURE1RMVgwTDNISTE1TDFYSDVLR0cyS1kwWS4u
Lab Lyon - Data & AI	22/11/2022	240	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUMlZZOURET0RSWjcyTERYRkJGTlFFUjJaUi4u
Lab Nantes - Data & AI	29/11/2022	240	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUMlZZOURET0RSWjcyTERYRkJGTlFFUjJaUi4u
L'IA dans les services de données Azure	12/12/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdURE1RMVgwTDNISTE1TDFYSDVLR0cyS1kwWS4u
Open AI	H2	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdURE1RMVgwTDNISTE1TDFYSDVLR0cyS1kwWS4u

Event Academy, kickstart certifications, workshop certifications	Date	Duration (min)	Link
MCPP - DP-420	10/10/2022	420	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUMkJISlRKSU1RRFA0OVgzSFdTSTY0RE9WQy4u
Micro Hack CosmosDB	20/10/2022	420	H1 - Inscriptions PTA
Academy DP900	17-21/10/2022	300	https://msevents.microsoft.com/event?id=3250818161
Academy AI900	17-21/10/2022	300	https://msevents.microsoft.com/event?id=2717528090
Kickstart DP-500	17/10/2022	60	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNEk3WFQ1TEdNNTQ2Uk85V0cxQzM3TE9ZRS4u
Dry Run DP-500	14/11/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNEk3WFQ1TEdNNTQ2Uk85V0cxQzM3TE9ZRS4u
Q&A DP-500	05/12/2022	90	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNEk3WFQ1TEdNNTQ2Uk85V0cxQzM3TE9ZRS4u
Kickstart DP-100	17/10/2022	60	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNDAxV0hSN0FHM1YzUzI3OUNMUFYxSkRIMi4u
Dry Run DP-100	14/11/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNDAxV0hSN0FHM1YzUzI3OUNMUFYxSkRIMi4u
Q&A DP-100	05/12/2022	90	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUNDAxV0hSN0FHM1YzUzI3OUNMUFYxSkRIMi4u
Kickstart DP-203	17/10/2022	60	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUOVFWOUVCNFcyQk5SVjFBUFczNktCUFpLMi4u
Dry Run DP-203	14/11/2022	120	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUOVFWOUVCNFcyQk5SVjFBUFczNktCUFpLMi4u
Q&A DP-203	05/12/2022	90	https://forms.office.com/Pages/ResponsePage.aspx?id=v4j5cvGGr0GRqy180BHbR3zwJTO3s11AuaqpNnBbrwdUOVFWOUVCNFcyQk5SVjFBUFczNktCUFpLMi4u

Microsoft

Workshop - Azure Cosmos DB API for MongoDB

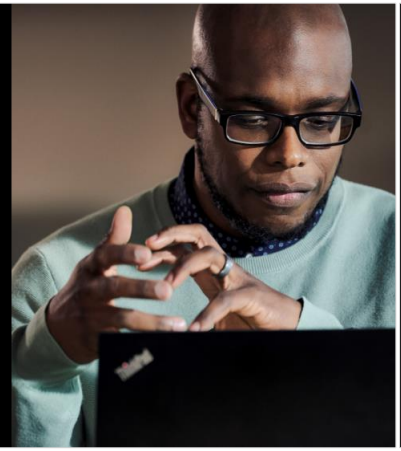
Ali Bouhaddou
Cloud Solution Architect
Data Analytics
albouhad@microsoft.com

Frederic Gisbert
Cloud Solution Architect
Data Analytics
frgisber@microsoft.com

Agenda

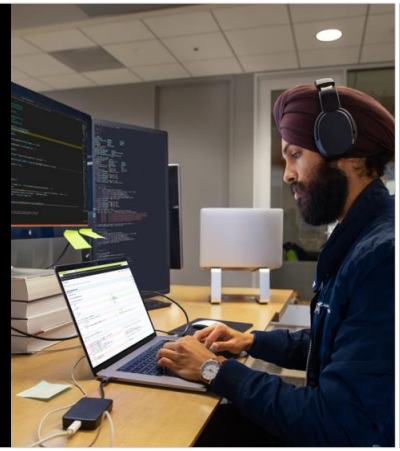
Azure Cosmos DB API for MongoDB overview

Introduction to Azure Cosmos DB
API for MongoDB value proposition
Use cases



Migration

Pre & post migration stages
Data migration methods

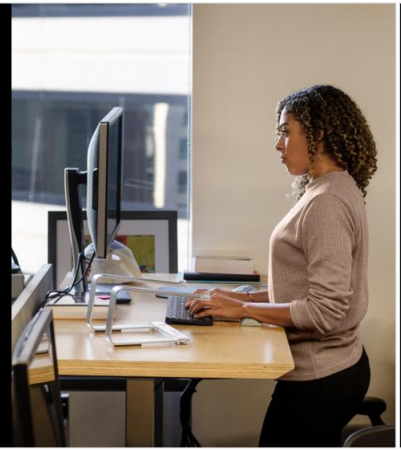


Planning & architecture

Resource model
Sharding
Data modelling
Geo-distribution & failover
Consistency models
Capacity calculations & billing

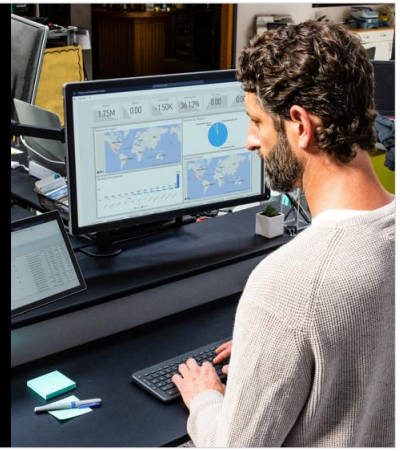
Implementation & development

API for MongoDB & features
Indexing
Query tuning
CRUD operations



Administration

Troubleshooting & monitoring
Log analytics




Microsoft Azure
Azure Cosmos DB
NoSQL migration

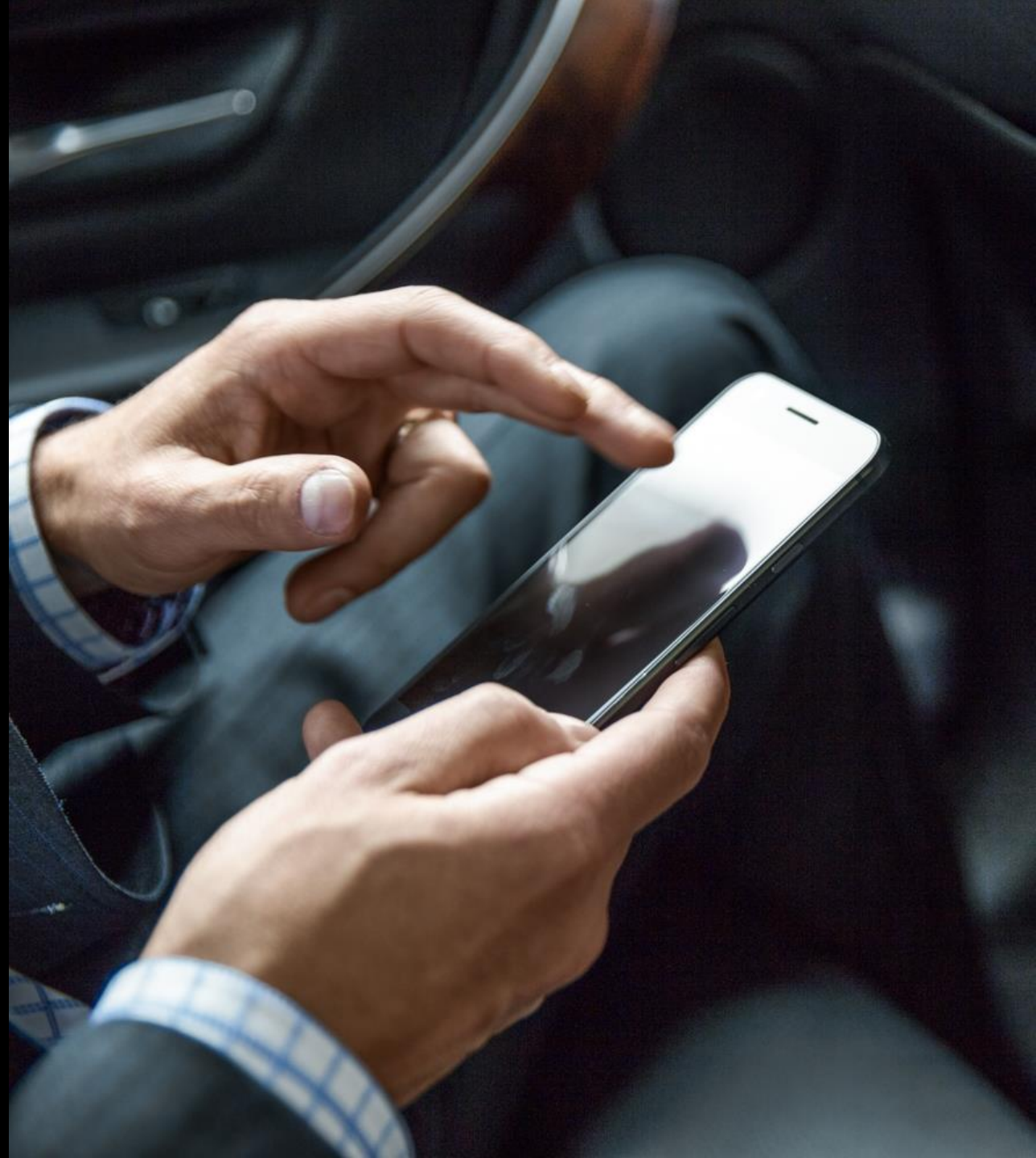
Today's applications are expected to do more than ever

Scale rapidly and elastically

Ensure high performance

Power real-time experience

Deliver business insights



Modernize existing applications

90% of existing applications will be in use, making App Modernization a key customer priority by 2025

Bring superior scale, speed, and availability to your applications by modernizing your apps and the databases that power them

Azure offers fully-managed cloud application services and database services to simplify managing your apps and handle all your data

1. Gartner, Application Modernization Should Be Business-Centric, Continuous and Multiplatform, Thomas Klinect, Stefan Van Der Zijden, 15 August 2019

Modernize existing applications

Lift and shift with virtual machines

Fully managed PaaS

Modernize using containers

Modernize LoB Apps with low code

Control ⟷ Productivity

Data modernization

DevOps

NoSQL IaaS and on-premises offerings don’t meet the moment

On-premises and IaaS NoSQL options struggle with:

Poor performance and lack of data integrity

Tedious set-up and maintenance

Expensive and complex availability solutions

High costs from over-provisioning to achieve scale and high-maintenance databases

Azure Cosmos DB raises the NoSQL bar

On-premises and IaaS NoSQL options struggle with:		Azure Cosmos DB delivers:
Poor performance and lack of data integrity	→	**Guaranteed speed at any scale**
Tedious set-up and maintenance	→	**Simplified app development**
Expensive and complex availability solutions	→	**Mission-critical ready**
High costs from over-provisioning to achieve scale and high-maintenance databases	→	**Fully managed and cost-effective**

Migrate to cloud NoSQL with Azure Cosmos DB

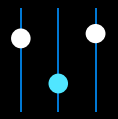
Make data modernization easy with seamless migration of NoSQL workloads to cloud

Get all the benefits of Azure Cosmos DB ⟶ using familiar database engines and tools

- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput
- <10ms latency on Core (SQL) API

- Near-real time insights with Azure Synapse Link on API for MongoDB and Core (SQL) API
- Languages of your choice with SDKs for .NET, Java, and Python

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide SLA
- Continuous backup and point-in-time restore



- Fully managed database service automatically handles all maintenance and updates
- Autoscaling removes the need for provisioning capacity
- Cost-effective serverless database operations model responds to app patterns is ideally suited for small, spiky workloads
- Auto-indexing capabilities on Core (SQL) API

Core (SQL) API

Supports all claims and SLAs

MongoDB

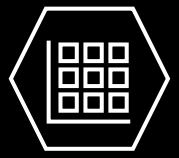


Table API

Azure Cosmos DB Core (SQL) API

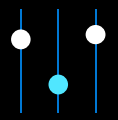
The native API for Azure Cosmos DB offers a view of your data that resembles a traditional NoSQL document store with capabilities like auto-indexing and <10ms latency. The Core (SQL) API can be used to migrate Couchbase, HBase, DynamoDB, and other NoSQL data to Azure Cosmos DB.

Premium capabilities

- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput
- <10ms latency SLA

- **Near-real time insights with Azure Synapse Link**
- Languages of your choice with SDKs for .NET, Java, and Python
- Document data model
- Migration capability from Couchbase, HBase, and Amazon DynamoDB

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide SLA



- Fully managed database service automatically handles all maintenance and updates
- Dynamic and limitless autoscaling for unpredictable workloads with high performance needs
- Cost-effective serverless model for spiky workloads with moderate needs
- Auto-indexing capabilities

Core (SQL) API

Azure Cosmos DB API for MongoDB

Support your MongoDB workloads with industry leading, financially backed SLAs that free you of database management tasks and lower TCO

Premium capabilities → delivered with migration made easy

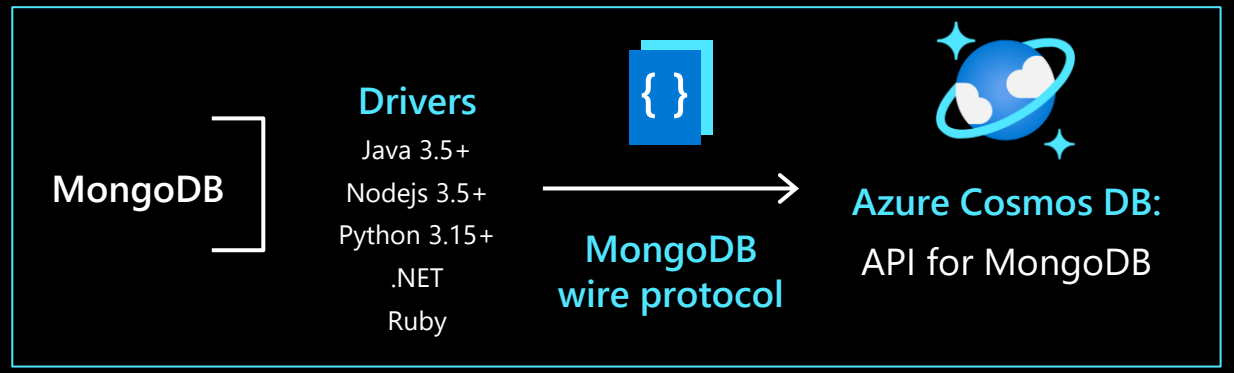
- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput

- **Near-real time insights with Azure Synapse Link**
- Languages of your choice with SDKs for .NET, Java, and Python
- Document data model

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide SLA
- Continuous backup and point-in-time restore

- Fully managed database service automatically handles all maintenance and updates
- Dynamic and limitless autoscaling for unpredictable workloads with high performance needs
- Cost-effective serverless model for spiky workloads with moderate needs

- Compatible with v4.0, v3.6, and v3.2 for wire protocol support*
- ACID-compliant multi-transaction support
- Good pricing option for low-storage, high-throughput workloads
- Migration tools to guide your journey:
  - Azure Database Migration Service online migration
  - Azure Database Migration Service offline migration
  - Striim continuous, real-time data flows



*MongoDB v4.0 supported as of March 2021

Azure Cosmos DB API for Cassandra

Support your Cassandra workloads with industry leading, financially backed SLAs that free you of database management tasks and lower TCO

Premium capabilities

- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput

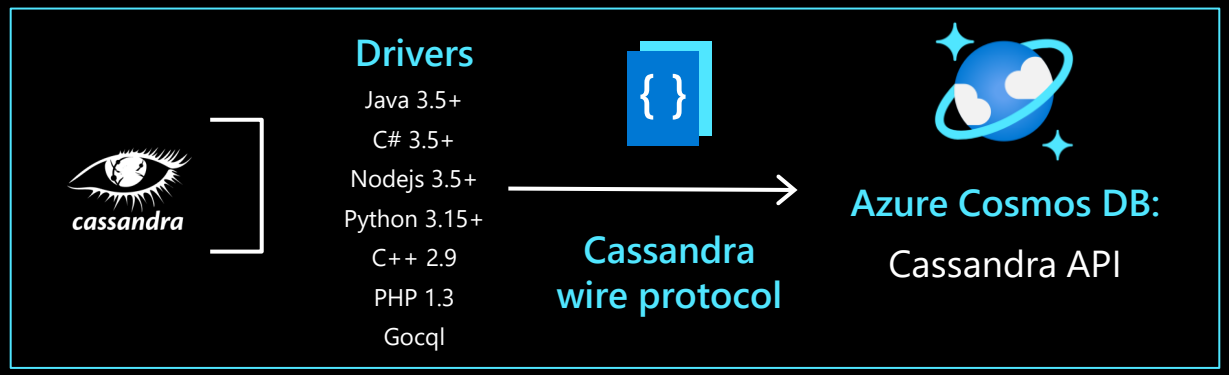
- Languages of your choice with SDKs for .NET, Java, and Python
- Columnar data model

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide SLA

- Fully managed database service automatically handles all maintenance and updates
- Dynamic and limitless autoscaling for unpredictable workloads with high performance needs
- Cost-effective serverless model for spiky workloads with moderate needs

Delivered with migration made easy

- Wire protocol support for Cassandra Query Language (CQL) v3.11
- Migration tools to guide your journey*:
  - Use cqlsh COPY command
  - Use Spark



*Migration from Azure Managed Instance for Apache Cassandra coming soon

Azure Cosmos DB API for Gremlin

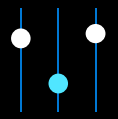
Store massive graphs with billions of vertices and edges with the API based on Apache TinkerPop using Gremlin query language

Premium capabilities

- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput

- Languages of your choice with SDKs for .NET, Java, and Python
- Graph data model

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide



- Fully managed database service automatically handles all maintenance and updates
- Dynamic and limitless autoscaling for unpredictable workloads with high performance needs
- Cost-effective serverless model for spiky workloads with moderate needs

Azure Cosmos DB Table API

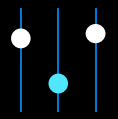
Applications written for Azure Table storage can migrate to Azure Cosmos DB by using the Table API with no code changes and take advantage of premium capabilities

Premium capabilities

- Instant, elastic, automatic, and limitless scale
- Data replication across any Azure region
- Guaranteed data consistency and throughput

- Languages of your choice with SDKs for .NET, Java, and Python
- Key-value data model

- Turnkey multi-master writes enabled by a click of a button
- Industry-leading 99.999% availability worldwide SLA



- Fully managed database service automatically handles all maintenance and updates
- Dynamic and limitless autoscaling for unpredictable workloads with high performance needs
- Cost-effective serverless model for spiky workloads with moderate needs

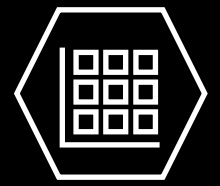


Table API

Migration

Pre & post migration stages

Data migration methods

Migration steps

Pre-migration

Inventory the data estate

Build an execution plan

Migration

Execute data migration & monitor progress

Post-migration

Reconcile application(s) to use Azure Cosmos DB connection string

Optimize indexing, global distribution, consistency, etc.

Pre-migration checklist

Choose the API for MongoDB version based on the supported features and syntax

- https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb-feature-support-40

Estimate throughput needed for the application

Provisioned RU/s = C*T/R

Provisioned RU/s = (Recommended provisioned RU/s per vCore or vCPU) * (Total vCores and/or vCPUs in your existing database)/(Replication factor of your existing **data-bearing** replica set)

- *C* = 600 RU/s/vCore* for Azure Cosmos DB for NoSQL
- *C* = 1000 RU/s/vCore* for Azure Cosmos DB for MongoDB v4.0

Pick the right shard keys

Decide the best indexing policy for the data

Keep the destination Azure Cosmo DB account ready with the right API version, region

Choose the migration method

Common migration methods

- Database Migration Service
- Native MongoDB Tools
- Azure Data Factory

Demo & overview of the migration methods

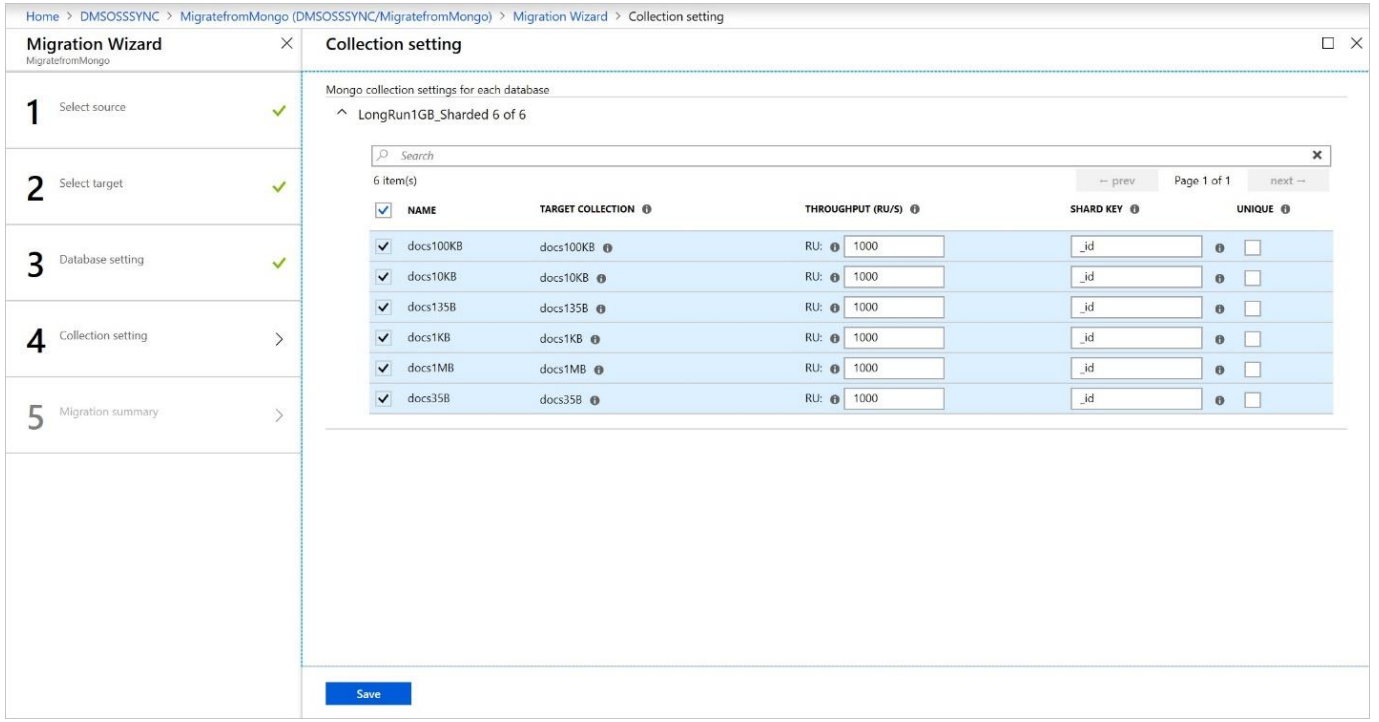
Database Migration Service

Makes use of the Azure Cosmos DB bulk executor library

Online and Offline mode available. Takes care of replicating live changes in Online mode.

Online migration is supported in Premium tier. Free for first 6 months.

Suitable for large datasets



Native MongoDB Tools

- Familiar, easy to set up and integrate
- *mongodump* & *mongorestore* – BSON-based
- *mongoexport* & *mongoimport* – JSON or csv-based, Can be used to migrate subset of data
- Can be used with clients such as mongo shell, Studio 3T, Robomongo and others
- Considerations: Needs custom handling for throttles

Azure Data Factory

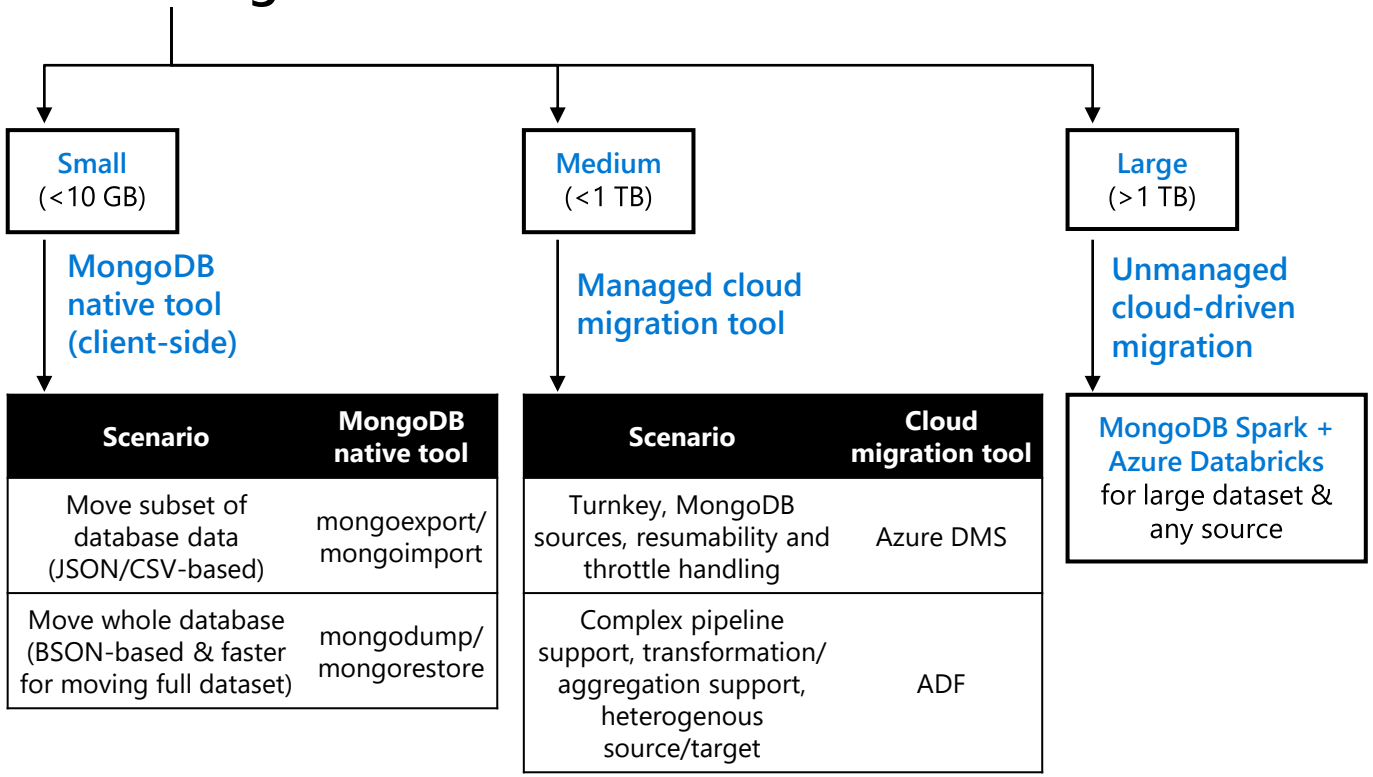
- A fully managed, serverless data integration service
- Makes use of the Azure Cosmos DB bulk executor library
- Suitable for large datasets

Considerations: Azure Data Factory

- Pipeline needs to created for each collection
- Lack of checkpointing
- Lack of a dead letter queue
- Indexes cannot be migrated

Which migration method to choose?

Offline migration

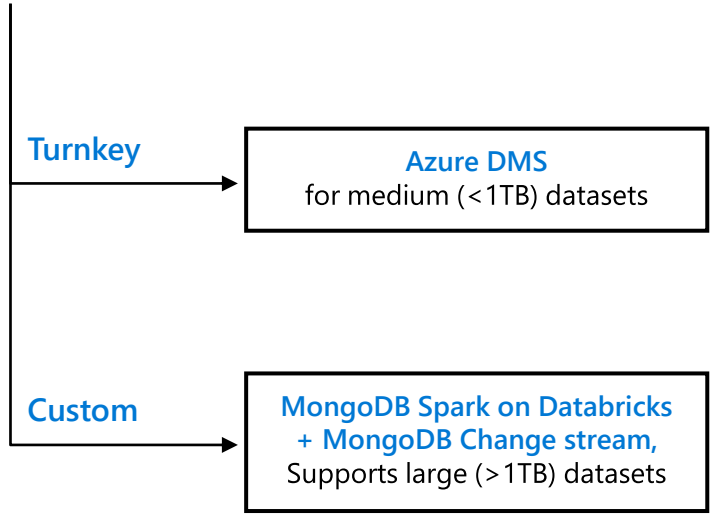


Scenario	MongoDB native tool
Move subset of database data (JSON/CSV-based)	mongoexport/ mongoimport
Move whole database (BSON-based & faster for moving full dataset)	mongodump/ mongorestore

Scenario	Cloud migration tool
Turnkey, MongoDB sources, resumability and throttle handling	Azure DMS
Complex pipeline support, transformation/ aggregation support, heterogenous source/target	ADF

MongoDB Spark + Azure Databricks
for large dataset & any source

Online migration



Post-migration Steps

- Change connection string in the application
- Optionally:
  - Optimize indexing policy, change consistency levels, configure geo-replication, automatic failover
  - Monitor performance – metrics, log analytics
  - Backups – smart defaults and customizable
  - Networking – firewall and private endpoints
  - Use built-in Jupyter and C# notebooks

Azure Cosmos DB API for MongoDB overview

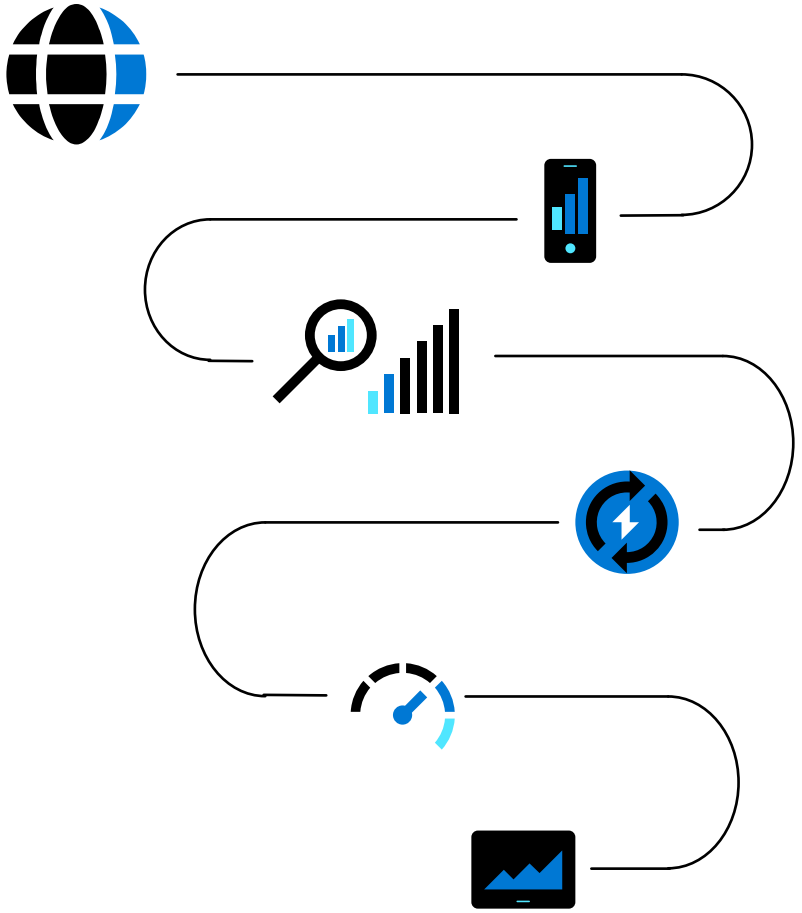
Introduction to Azure Cosmos DB

API for MongoDB value proposition

Use cases

Modern apps face new challenges

- Managing and syncing data distributed around the globe
- Delivering highly-responsive, real-time personalization
- Processing and analyzing large, complex data
- Scaling both throughput and storage based on global demand
- Offering low-latency to global users
- Modernizing existing apps and data



Azure Cosmos DB

A fully managed NoSQL database for modern app development with SLA-backed speed and availability, automatic and instant scalability, and open-source APIs for MongoDB, Cassandra, and other NoSQL engines.

Guaranteed speed at any scale

Gain unparalleled speed and throughput, fast global access, and instant elasticity.

Simplified application development

Build fast with open source APIs, multiple SDKs, schemaless data, and no-ETL analytics over operational data.

Mission-critical ready

Guarantee business continuity, 99.999% availability, and enterprise-level security for every application.

Fully managed and cost-effective

End-to-end database management with serverless and automatic scaling matching your application and TCO needs.

Simplified Application development

Start working with a DB using familiar MongoDB syntax within minutes

- JSON format document database
- Makes it easy to use Azure Cosmos DB as if it were a MongoDB database without the hassle of provisioning and managing it
- Supports API for MongoDB wire protocol versions
  - 4.2
  - 4.0
  - 3.6
  - 3.2
- Leverage existing MongoDB drivers, libraries and tools

Turnkey global distribution

Put your data where your users are in minutes

Automatically replicate all your data around the world, and across more regions than AWS and Google Cloud Platform combined.

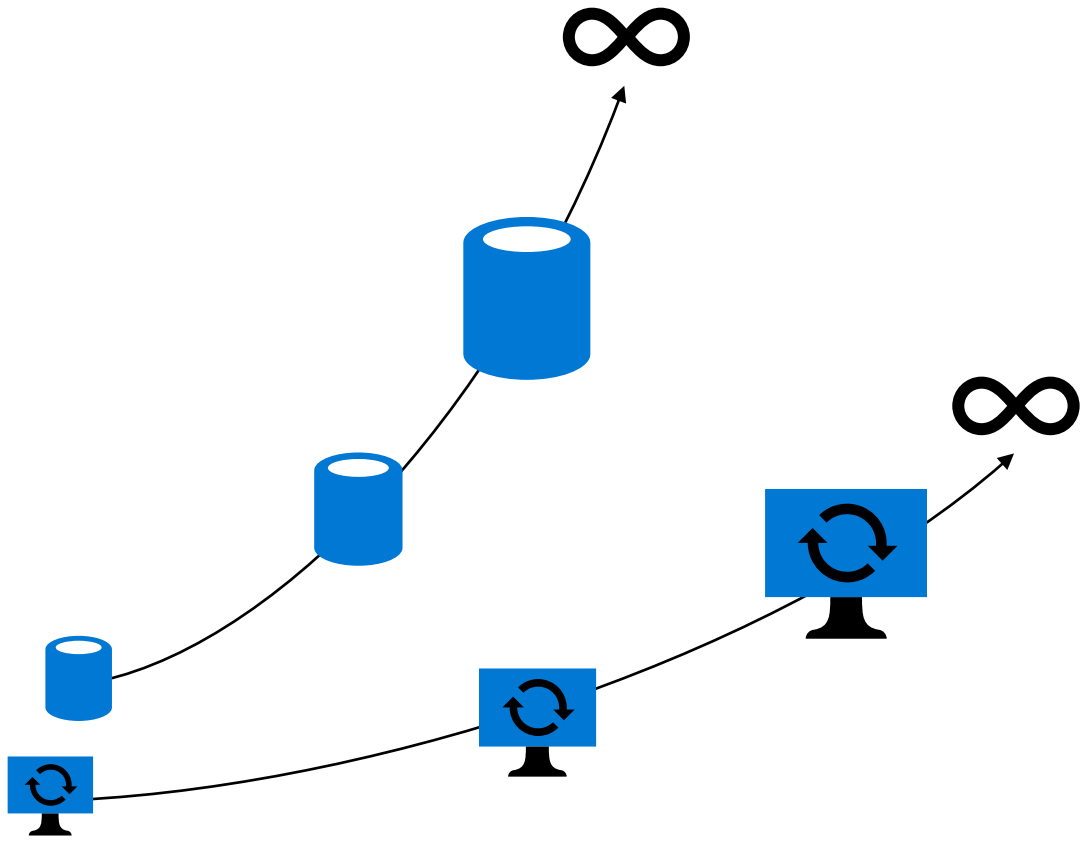
- Available in all Azure regions (60+ regions currently)
- Ring 0 service on Azure
- Manual and automatic failover
- Automatic & synchronous multi-region replication
- Multi-region writes

Elastic & granular scale-out of storage and throughput

Independently and elastically scale storage and throughput across regions

Pay only for the throughput and storage you need for your workload.

- No warm-up period needed for scale-up.
- Can scale in increments as small as 1/100th of a VM unlike MongoDB Atlas that can only increment in multiples of VM sizes.
- Scale down happens immediately, so you only pay for the resources that your workloads need on a per-hour basis.
- The first managed cloud database service to offer serverless MongoDB.



Choose throughput mode as per workload

Pay only for what you need

Provisioned

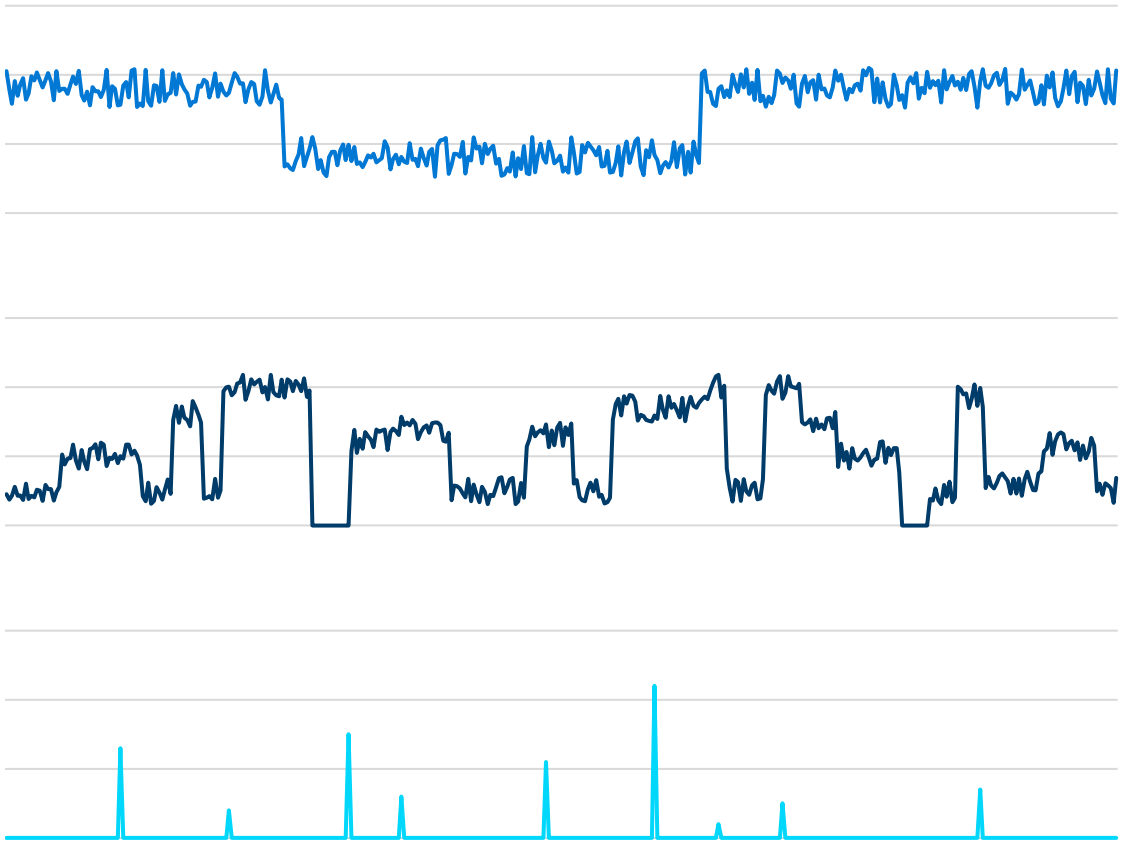
- Suits fixed workload requirements

Autoscale

- Suits workloads with flexible needs
- Accommodates sudden growth in usage.
- Granularly scale from 10 to 100s of millions of requests/sec.

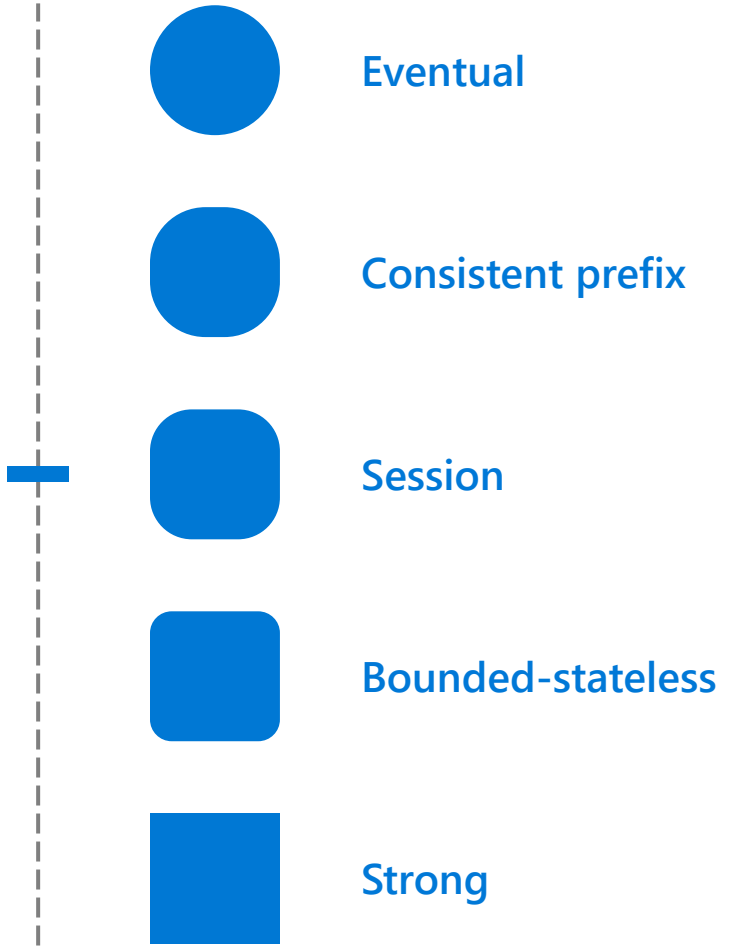
Serverless

- Suits spiky workloads
- Pay only for requests you use



Five well-defined consistency models

- Choose the best consistency model for your app
  - Offers five consistency models
  - 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



Comprehensive SLAs

Run your app on world-class infrastructure

- Financially-backed SLA for 99.999% high availability
- Guaranteed throughput and consistency

High Availability

99.999%

Throughput

Provisioned
Autoscale serverless

High Availability

Strong
Bounded Staleness Session
Consistent Prefix Eventual

Trust your data to industry-leading security & compliance

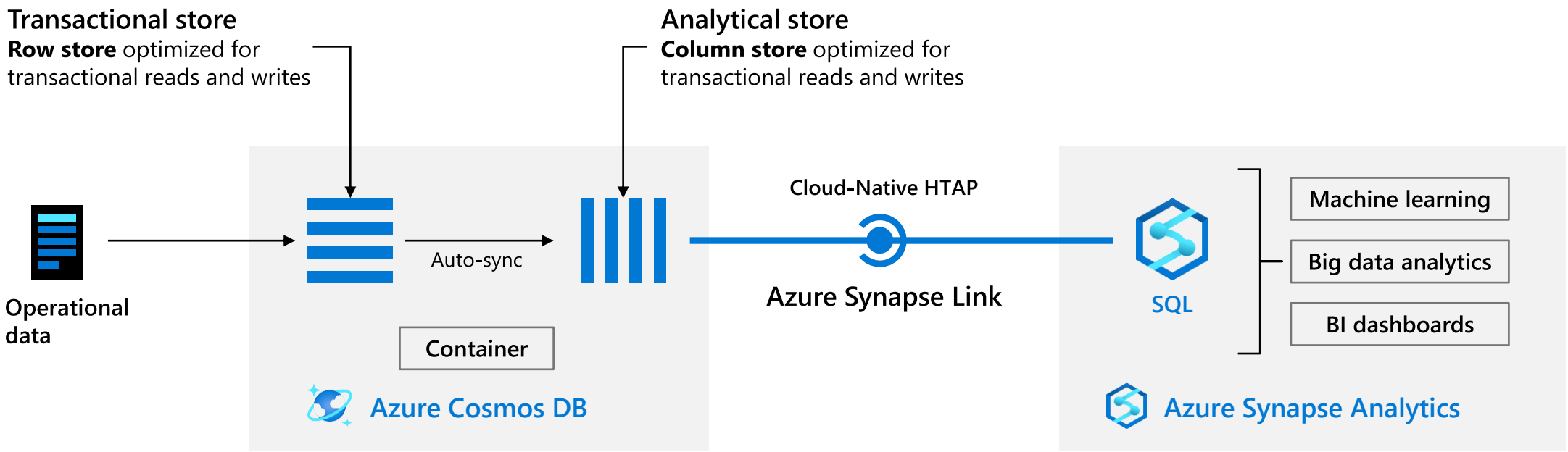
Azure is the world's most trusted cloud, with more certifications than any other cloud provider.

- Enterprise grade security
- Encryption at rest
- Encryption is enabled automatically by default
- Comprehensive Azure compliance certification

Real-time analytics with Synapse link

Use cloud-native HTAP to get the most for your business

- No performance impact on transactional workloads
- No ETL required



Fully managed & cost effective

Set-up your solutions hassle-free and within budget

- Handles maintenance and updates.
- API upgrades done with single click and zero downtime.
- Serverless and autoscale options
  - Keep costs low
  - Match resources to demand
  - Eliminate capacity management
  - Eliminate resource over-provisioning during busy or slow times.
- Free-tier discount: Get first 1000 RU/s & 25 GB storage free monthly for the lifetime of one Azure Cosmos DB account per Azure subscription.

Planning & architecture

Resource model

Sharding

Data modelling

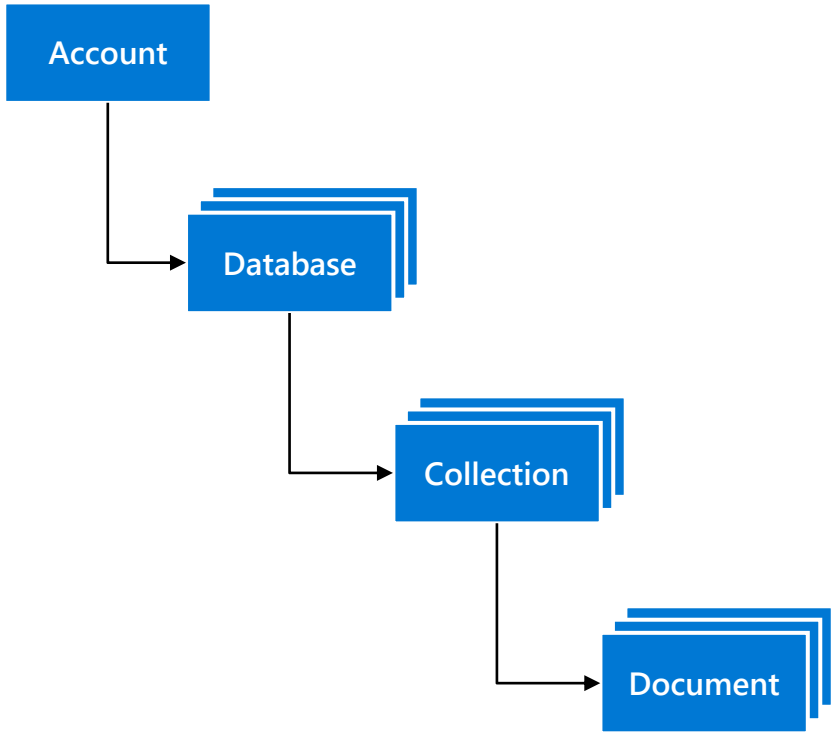
Geo-distribution & failover

Consistency models

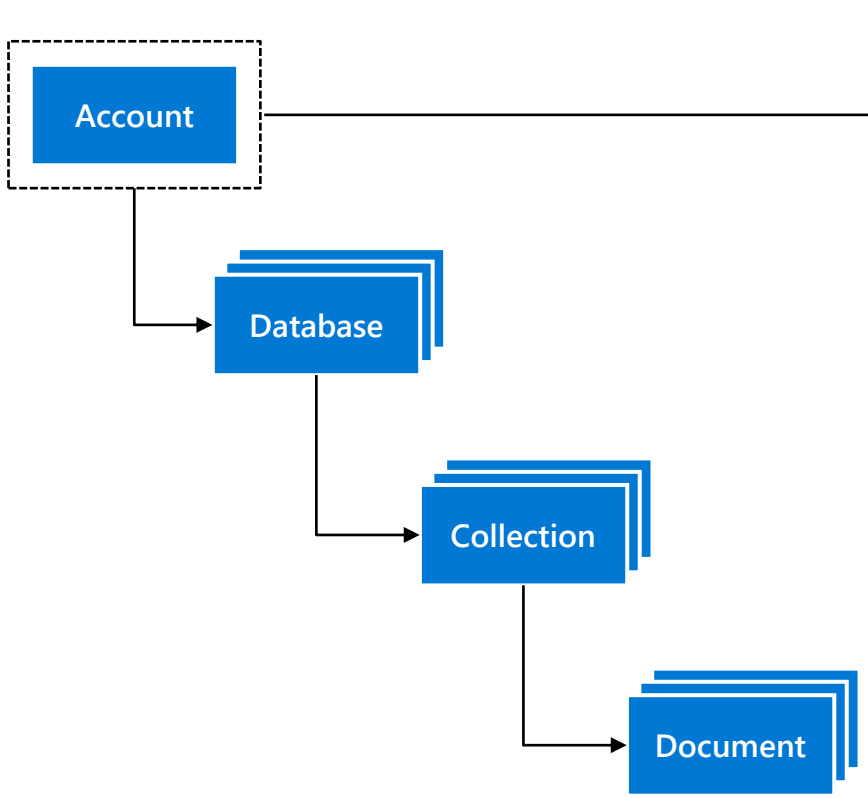
Capacity calculations & billing

Resource model

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



Account URI and credentials



*.mongo.cosmos.azure.com → For API for MongoDB > 3.6

*.documents.azure.com → For API for MongoDB 3.2

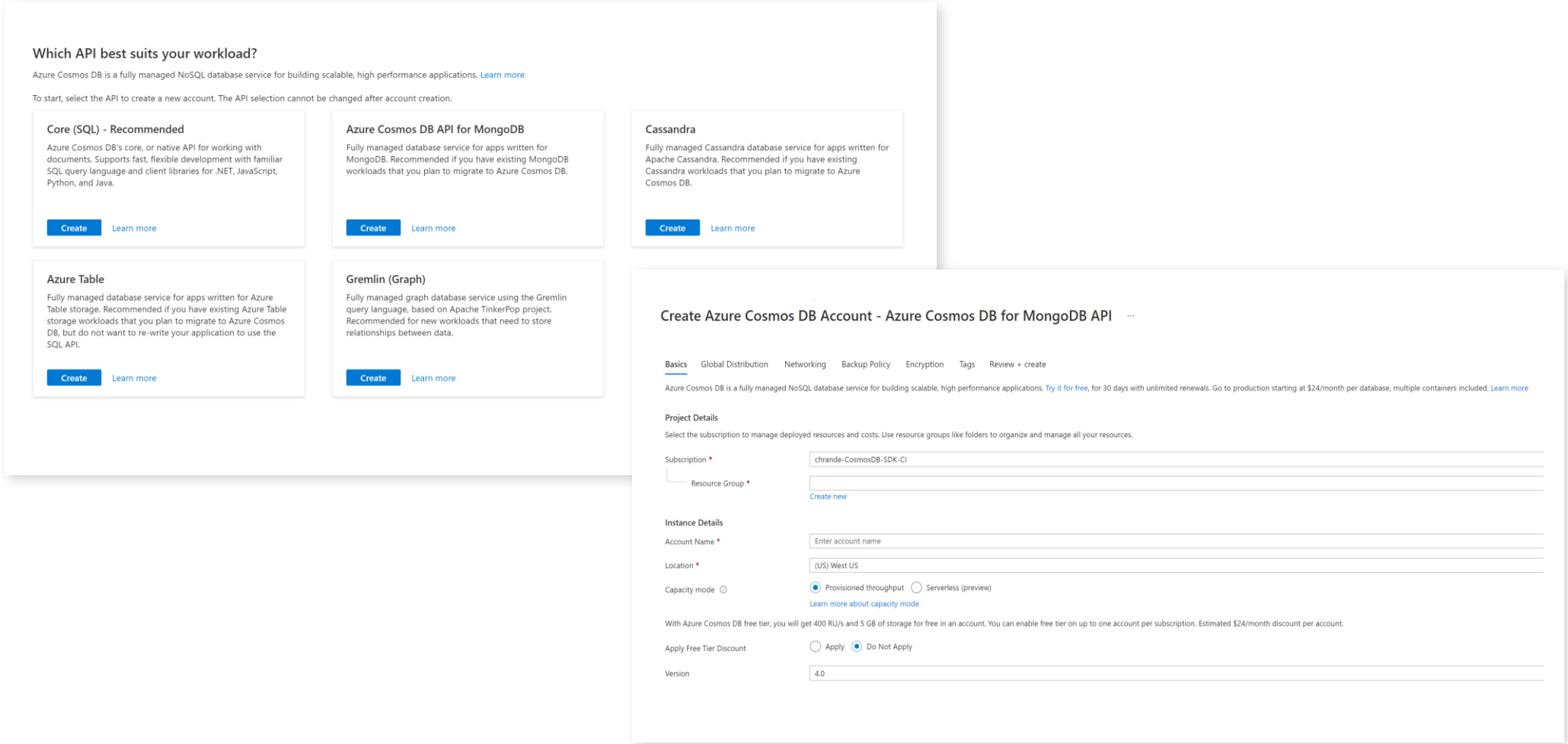
IGeAvVUp …

Primary & Secondary Read-Write Keys
Primary & Secondary Read-Only Keys

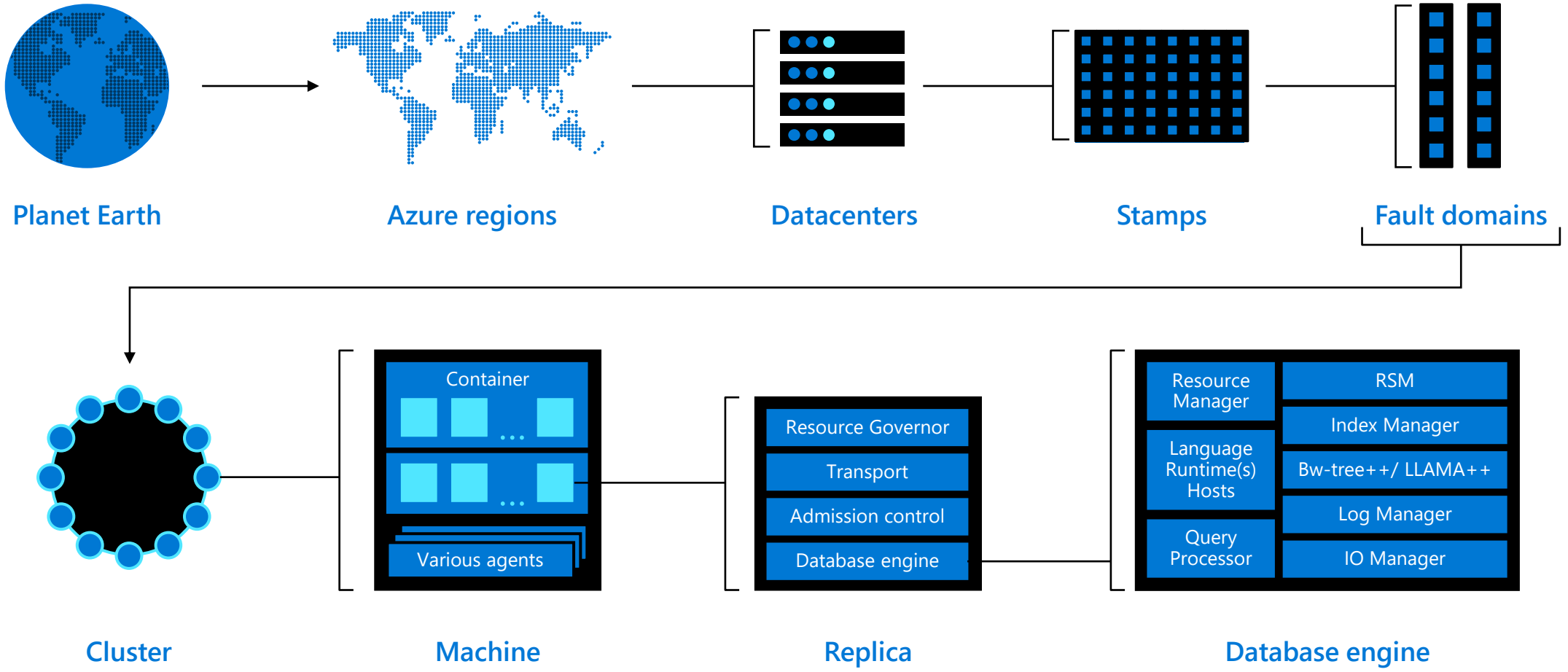
Mapping against Native MongoDB terminology

	Azure Cosmos DB API for MongoDB	Native MongoDB
Resources	**Account**	**Server or cluster**
	NA (abstracted for users, but we maintain 4 replicas per region)	**Replica set**
	Database	Database
	Collection	Collection
	Document	Document
	Partition or **Shard**	**Shard**
Credentials	**Account Name** or **Username**	**Username**
	Key or **Password**	**Password**

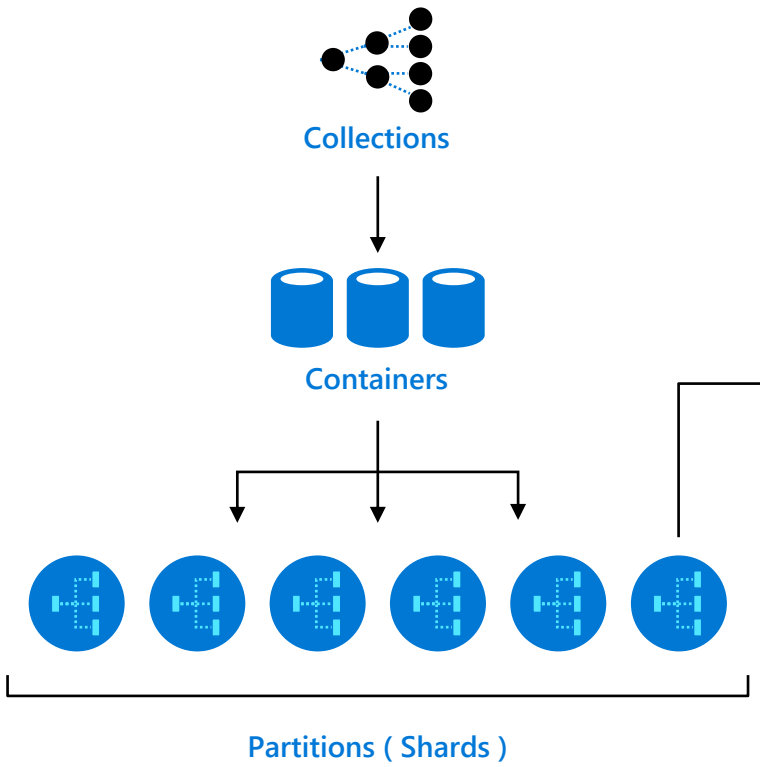
Creating account from portal



System topology (behind the scenes)

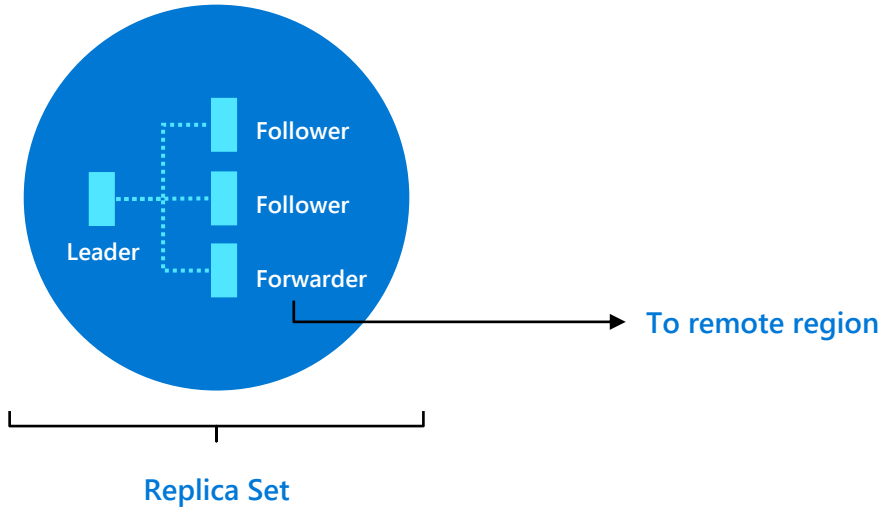


Resource hierarchy



Containers

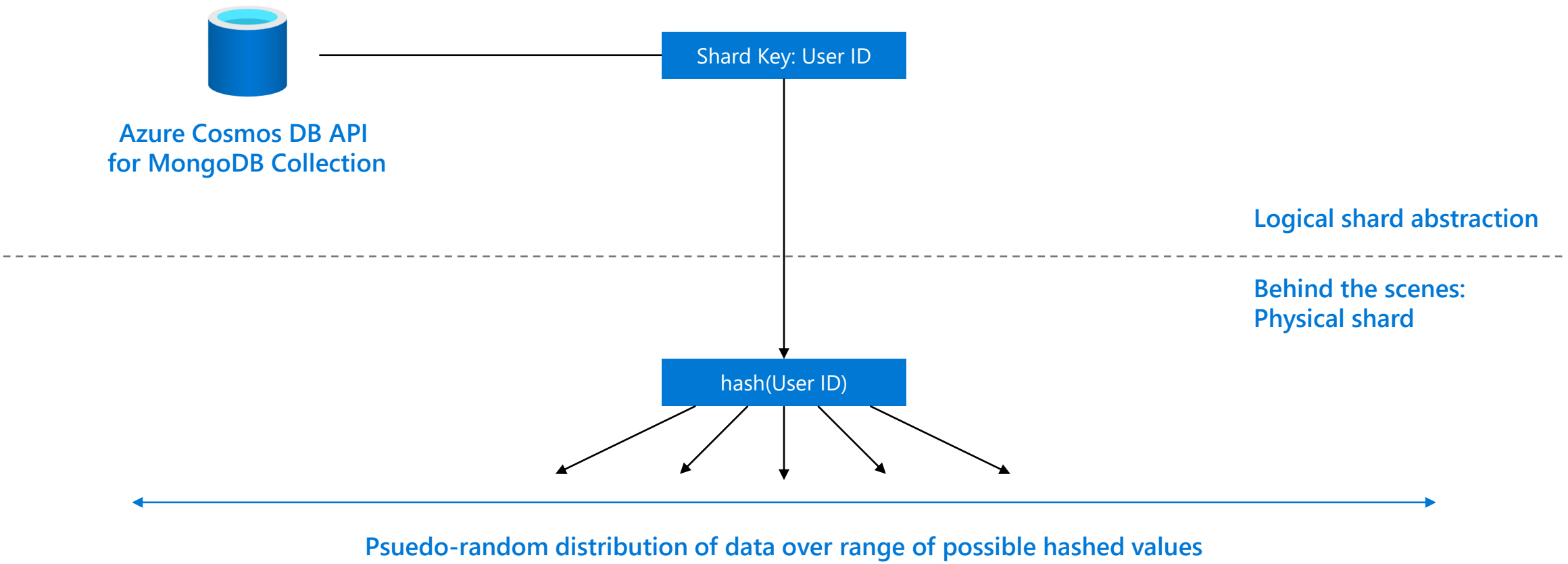
Logical resources “surfaced” to APIs as tables, collections or graphs, which are made up of one or more physical partitions or servers.



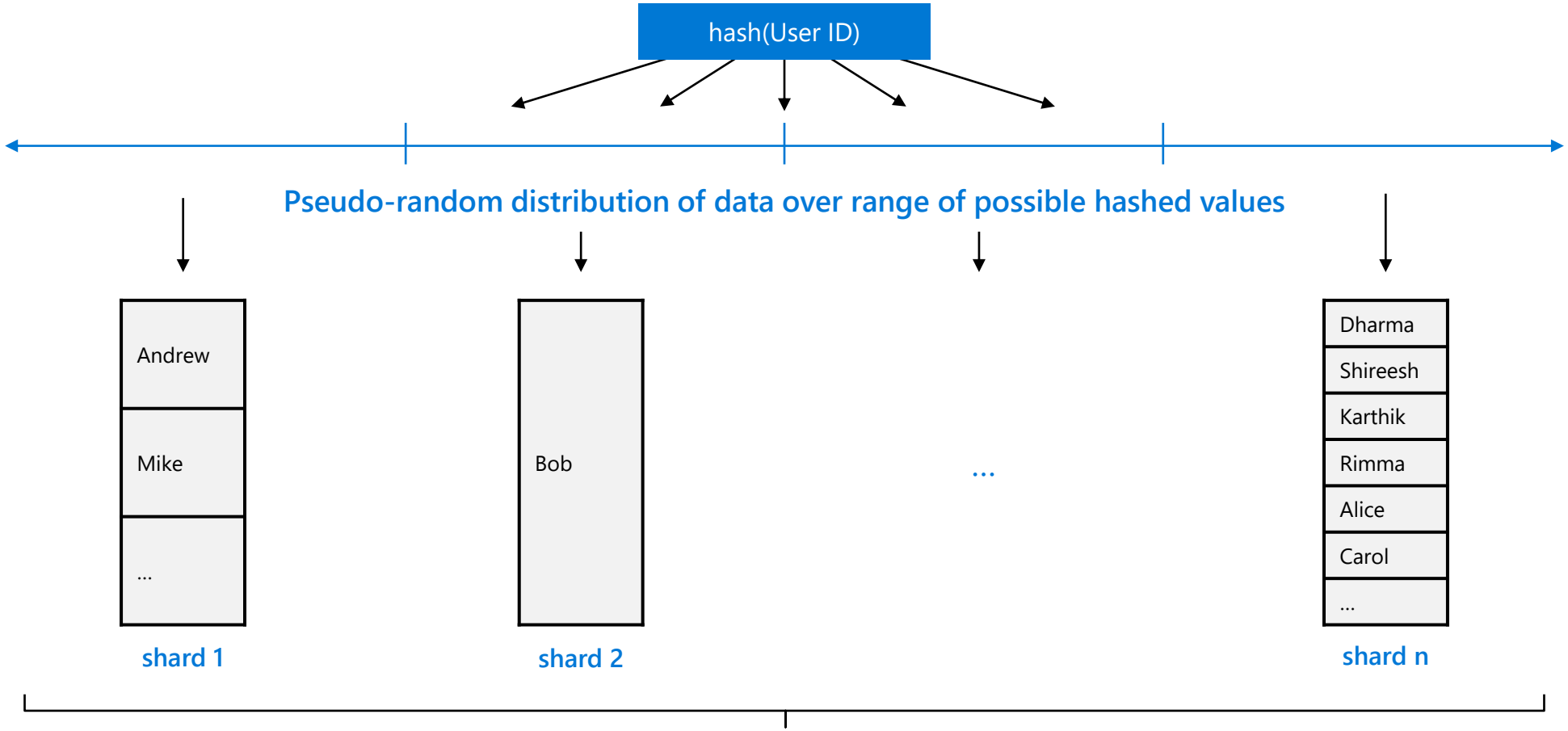
Resource partitions

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

Shards

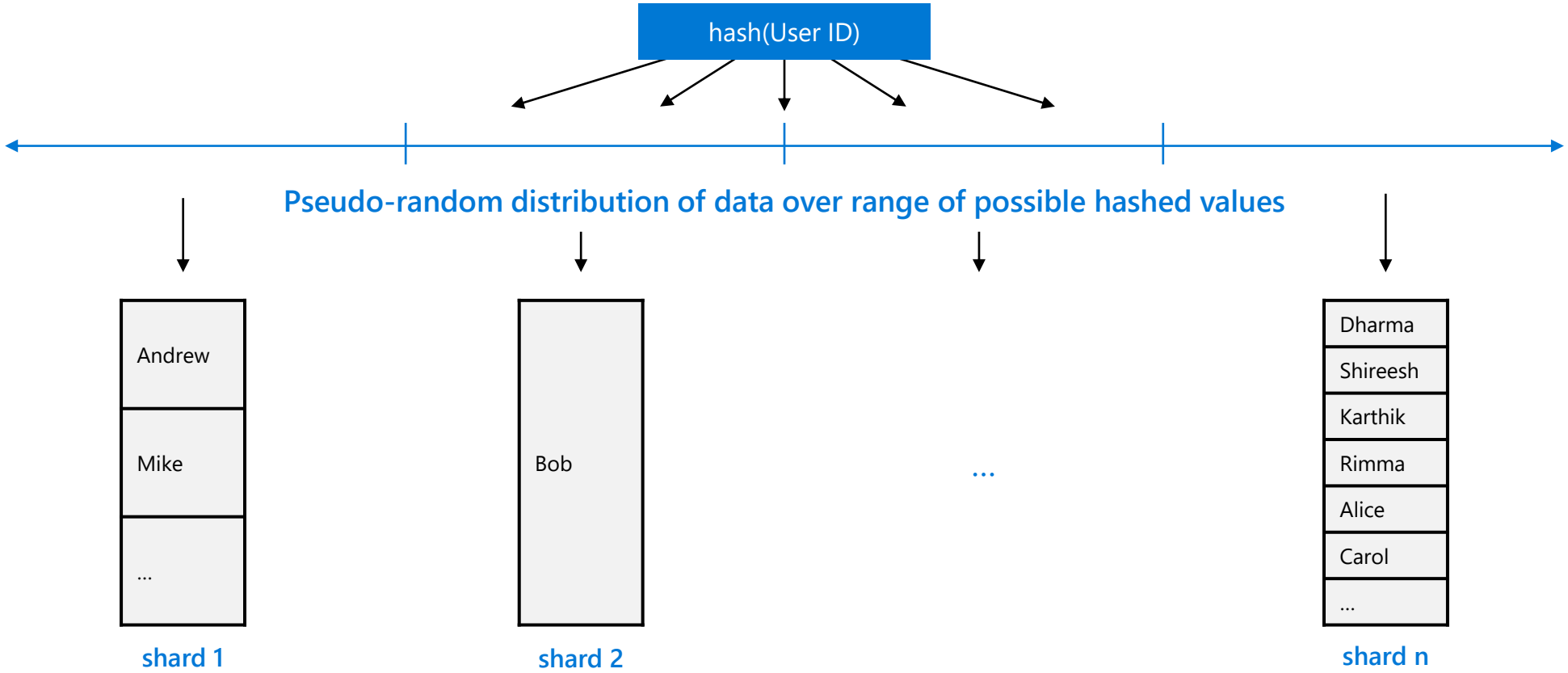

Azure Cosmos DB API
for MongoDB Collection
Shard Key: User ID
Logical shard abstraction
Behind the scenes:
Physical shard
hash(User ID)
Psuedo-random distribution of data over range of possible hashed values

Shards continued



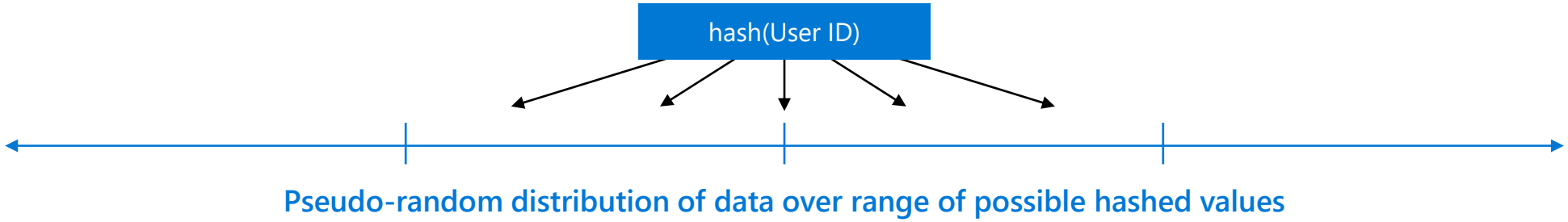
Frugal # of shards based on actual storage and throughput needs
(yielding scalability with low total cost of ownership)

Shards continued



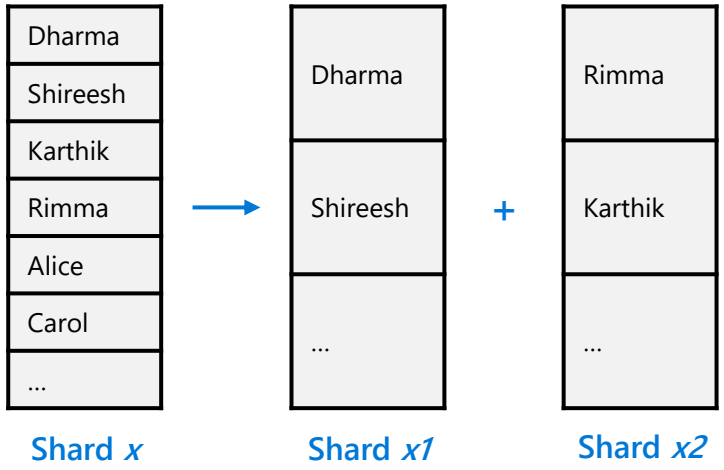
What happens when shards need to grow?

Shards continued



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

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



Shard key

Necessary only if collection size is >50 GB

If collection is <50 Gb, you do not need to pick a shard key.

- You can use a Fixed Collection.

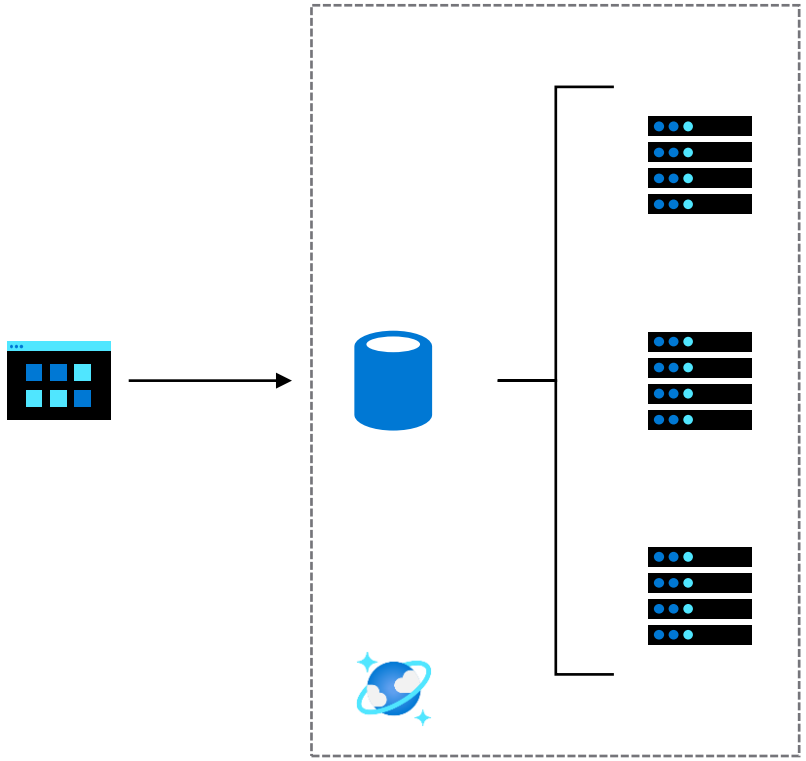
Choosing shard key

Important to select the “right” shard key

- Shard 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



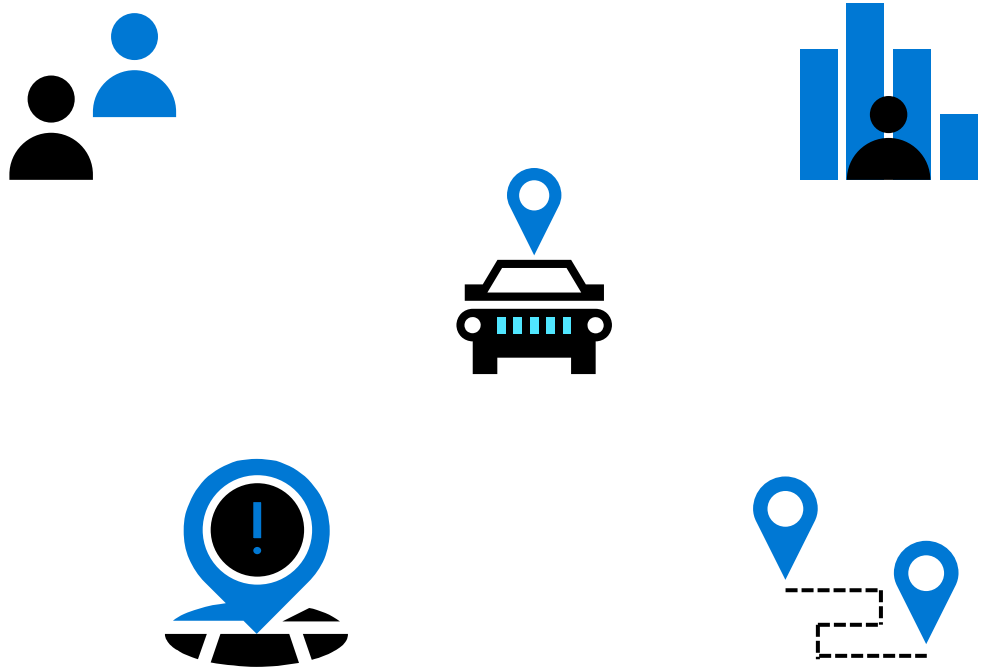
Choosing shard key continued

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

What are a few potential shard key choices?

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



Shard 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 shard key values; and is potentially the least granular option.

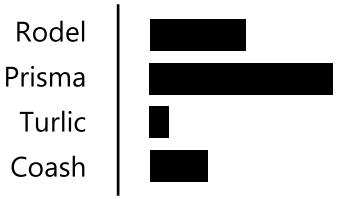
Depending how uniform sales are across various models – this introduces possibilities for hot shard keys on both storage and throughput.

Current month (e.g. 2021-04)

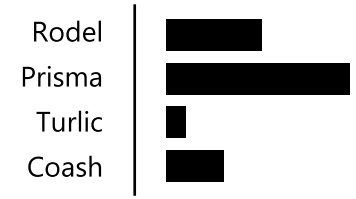
Auto manufacturers have transactions occurring throughout the year. This will create a more balanced distribution of storage across shards.

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

Storage Distribution



Throughput Distribution



Storage Distribution

Throughput Distribution

Example – Contoso Connected Car

Shard key choices continued

Device ID (e.g. DEVICE123)

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

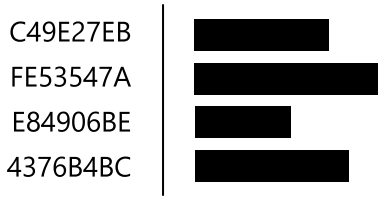
Depending on how many transactions occur per vehicle, it is possible to write more to a specific shard key that reaches the storage limit per shard key

Composite key (device ID + time)

This composite option increases the granularity of shard key values by combining the current month and a device ID. Specific shard 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.

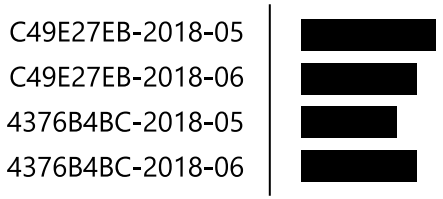
Throughput in this example would be distributed more to logical shard key values for the current month.

Storage Distribution

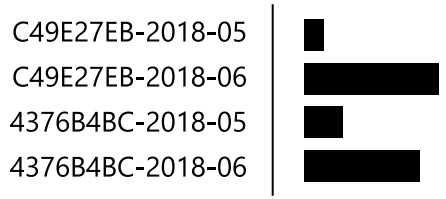


Throughput Distribution

Storage Distribution



Throughput Distribution

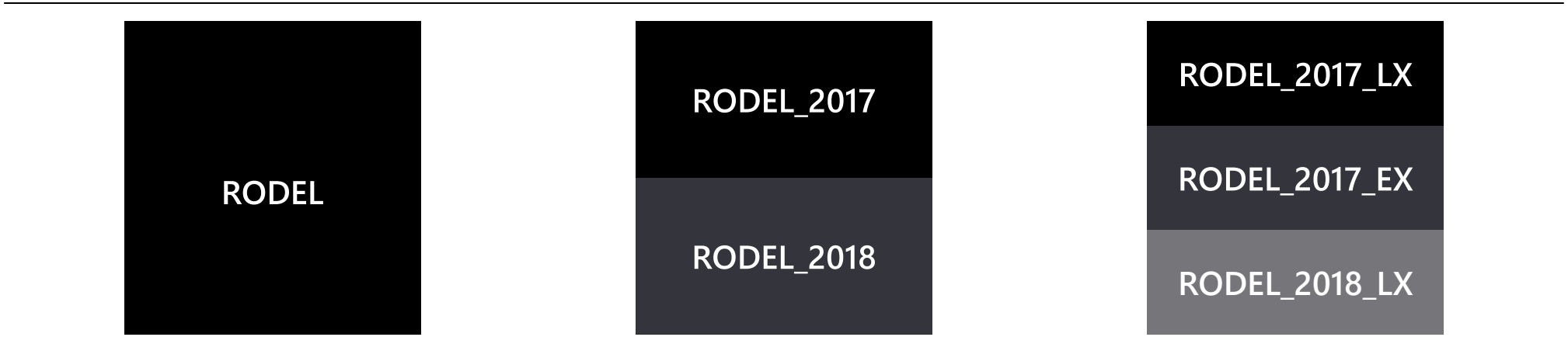


Example – Contoso Connected Car

SHARD granularity

Select the “right” level of granularity for your shards

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



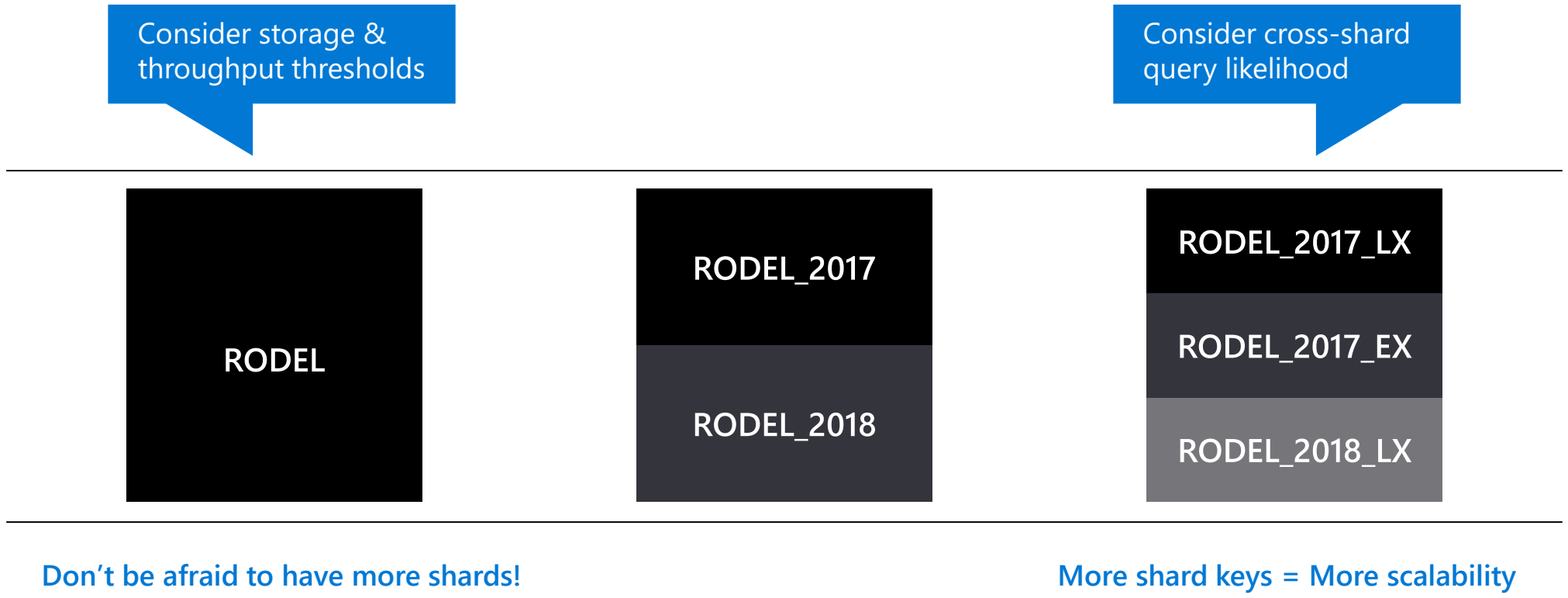
Don’t be afraid to have more shards!

More shard keys = More scalability

Example – Contoso Connected Car

Shard granularity continued

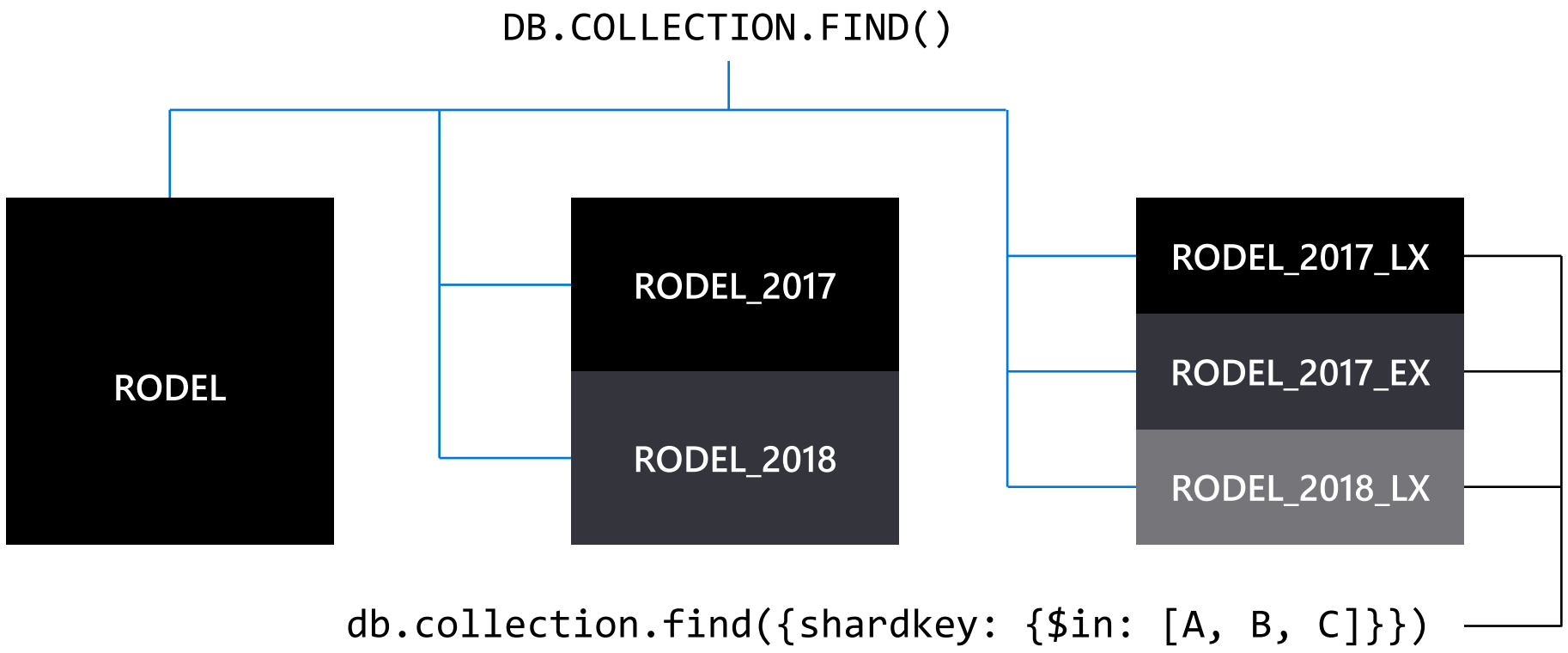
Select the “right” level of granularity for your shards



Example – Contoso Connected Car

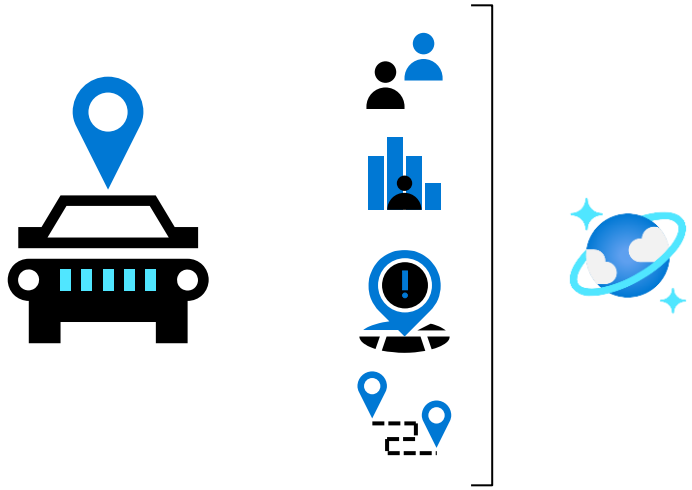
Shard granularity continued

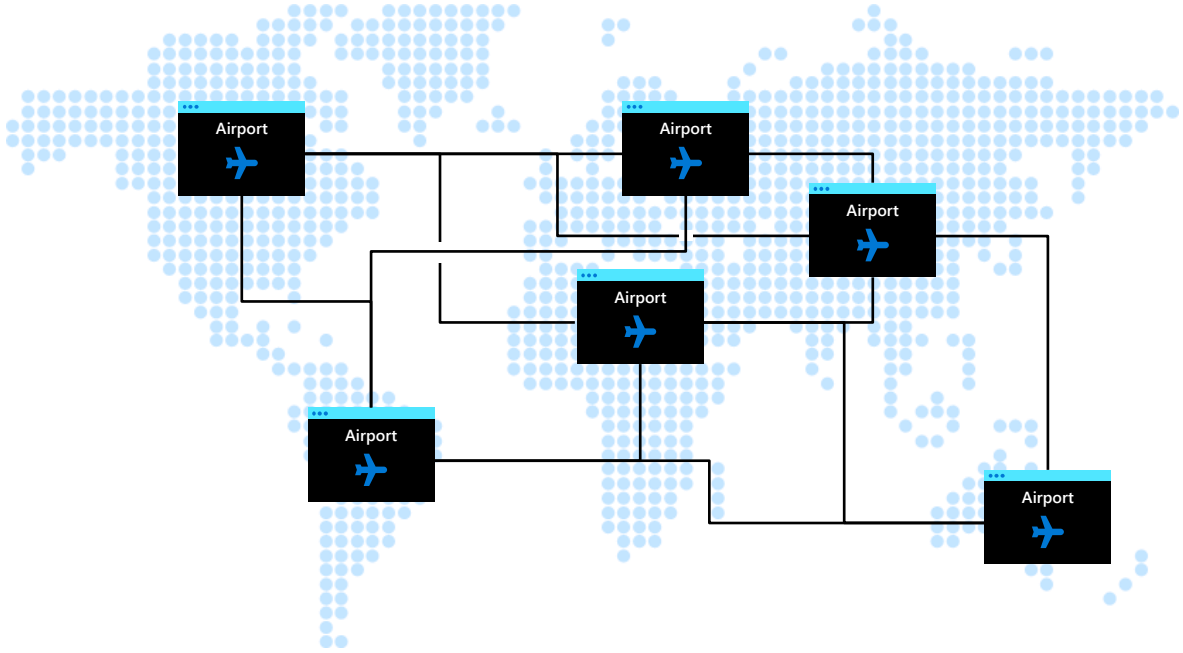
A cross-shard query is not always a blind fan out query



Shard key selection

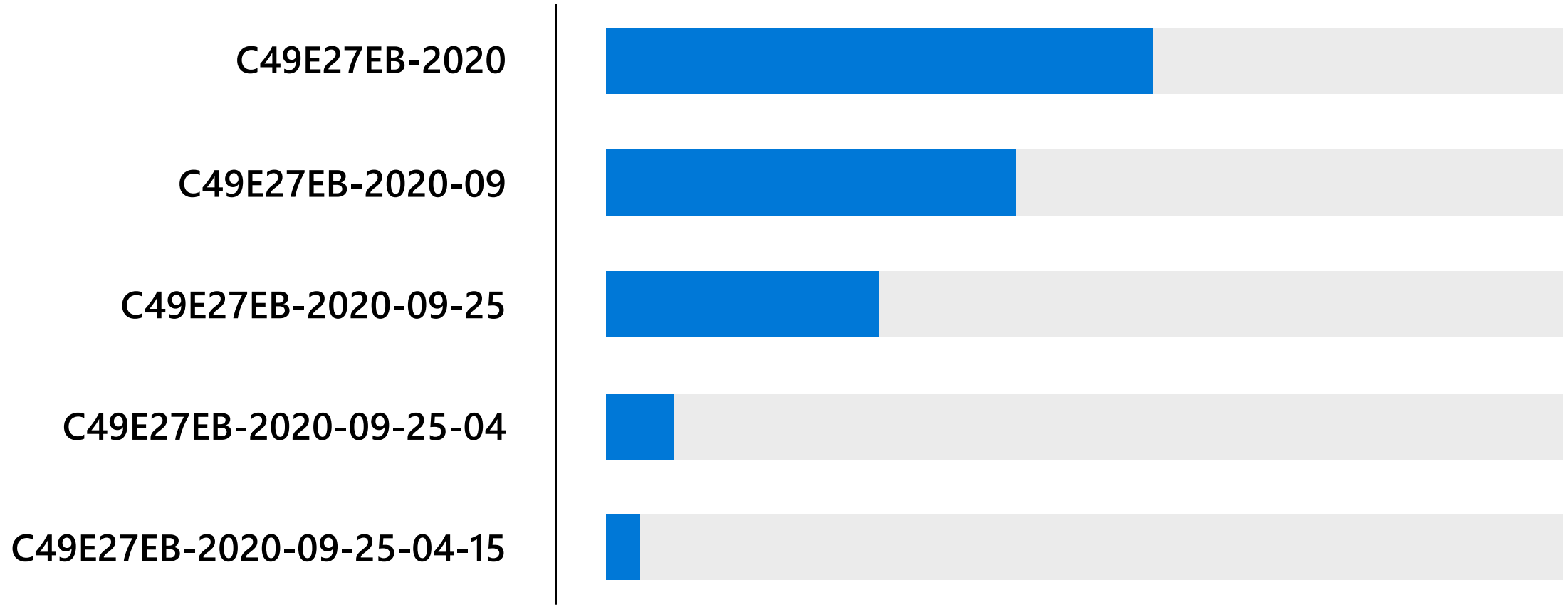
Contoso Connected Car is collecting and storing vehicle telemetry data from millions of vehicles. The team has decided to shard based on a composite key consisting of device id + current time when the interaction occurred.





Shard key scenario

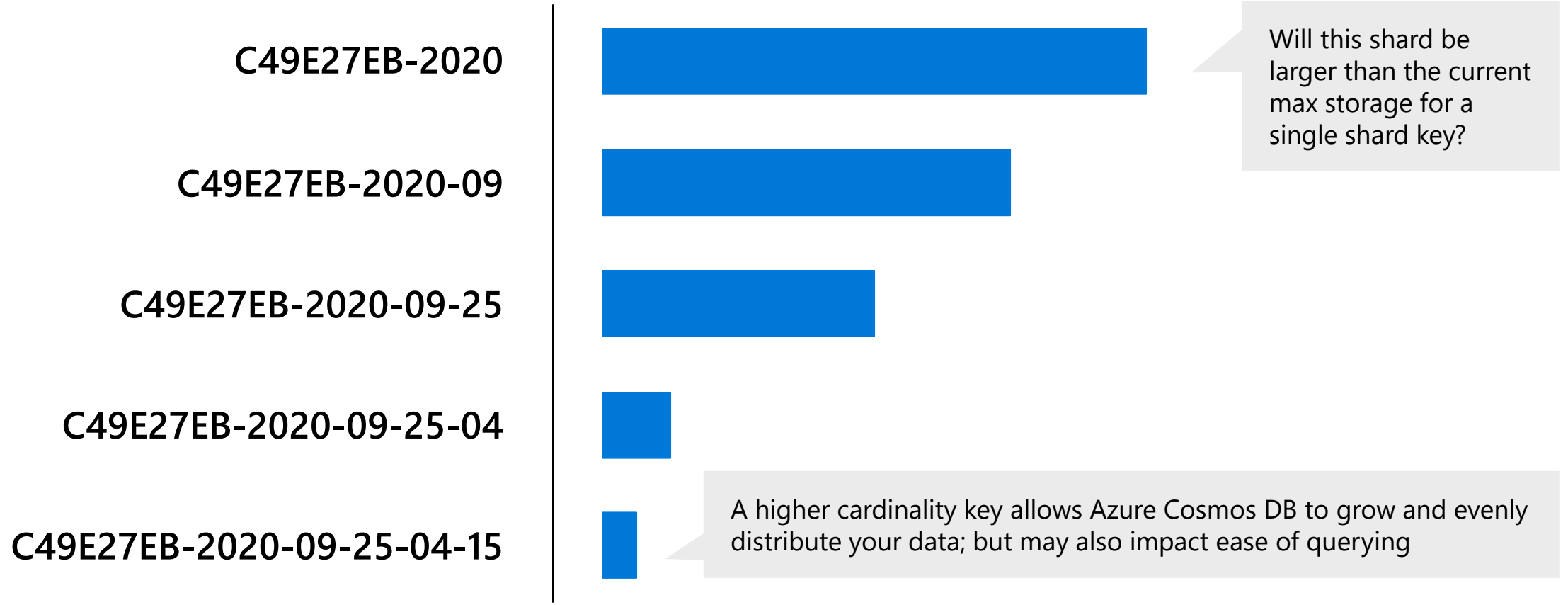
Interaction that occurred on: **September 25, 2020 at 4:15 AM UTC**



Example – Contoso Connected Car

Shard key scenario continued

Interaction that occurred on: **September 25, 2020 at 4:15 AM UTC**



Example – Contoso Connected Car

Shard tips

Best practices: design goals for choosing a good shard key

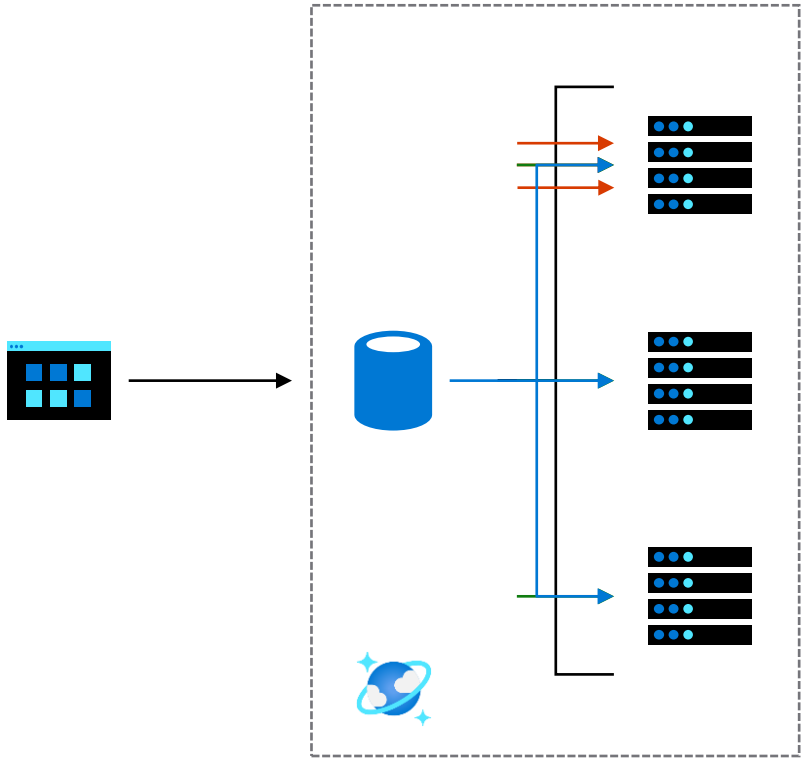
- Distribute the overall request + storage volume
  - Avoid "hot" shard keys
- Shard key is scope for multi-record transactions and routing queries
  - Queries can be intelligently routed via shard key
  - Omitting shard key on query requires fan-out

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 shard keys
  - Shards keys are logical
  - More shard keys = more scalability



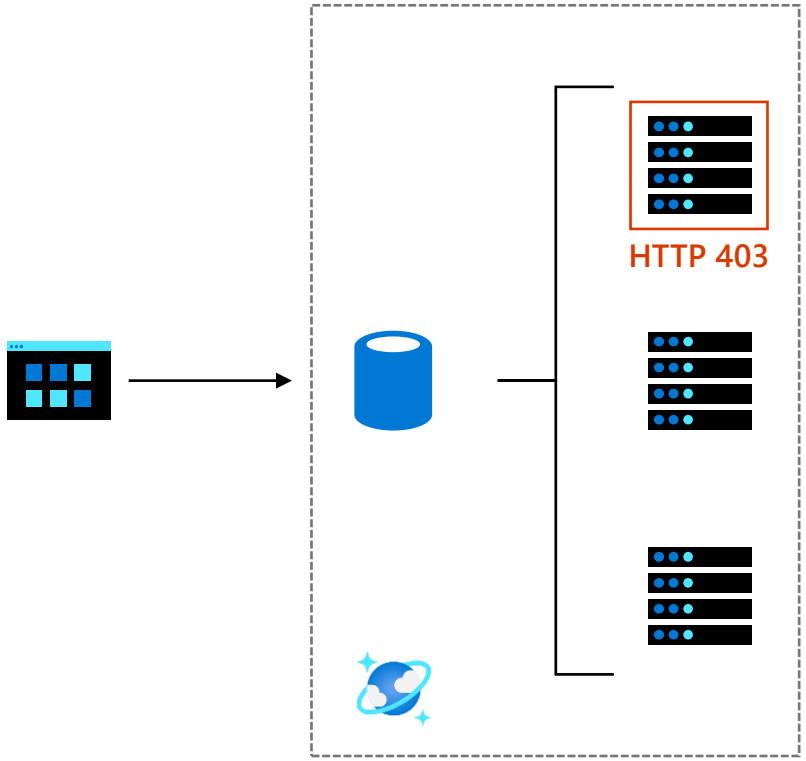
Shard key storage limits

Containers support unlimited storage by dynamically allocating additional physical shards

Storage quota for single shard (logical shard) is 20GB

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

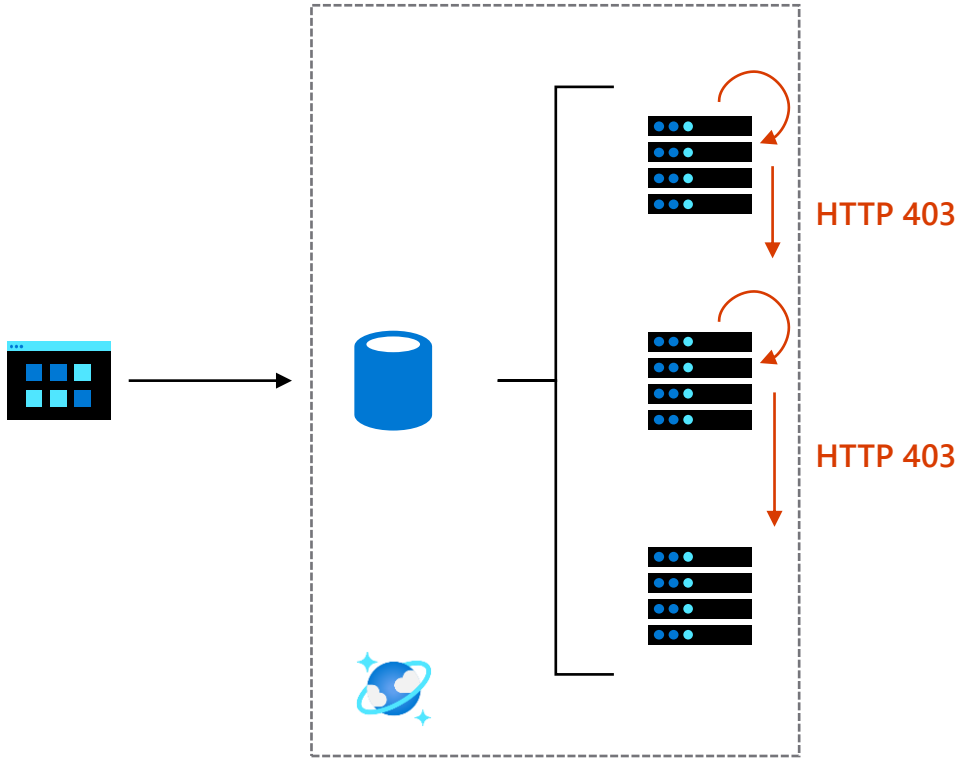
- Azure Cosmos DB will automatically add shards, and may also return a 403 if an authorization token has expired



Design patterns for large shard keys

"Linked list approach" by spreading data across incremental shard key values

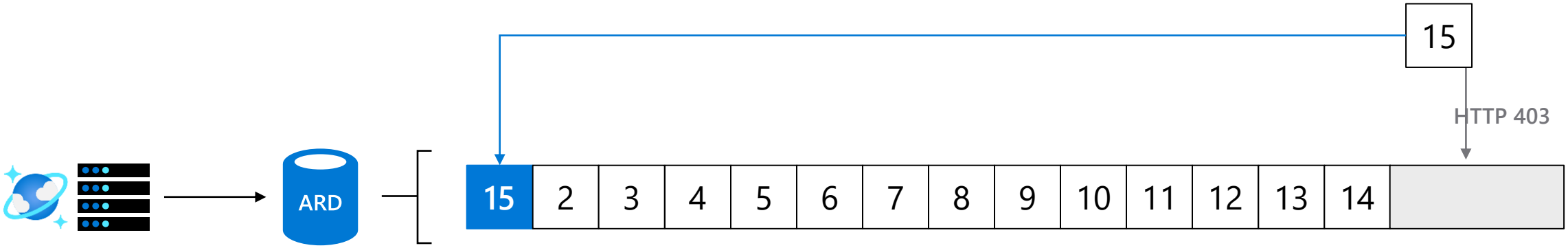
- For workloads that exceed quotas for a single shard key value, you can logically spread items across multiple shard keys within a container by using a suffix on the shard key value
- As a shard fills up, you can determine when to increment the shard key value by looking for the 403 status code in your application's logic



Design patterns for large shard keys

"Circular buffer" approach by reusing unique IDs

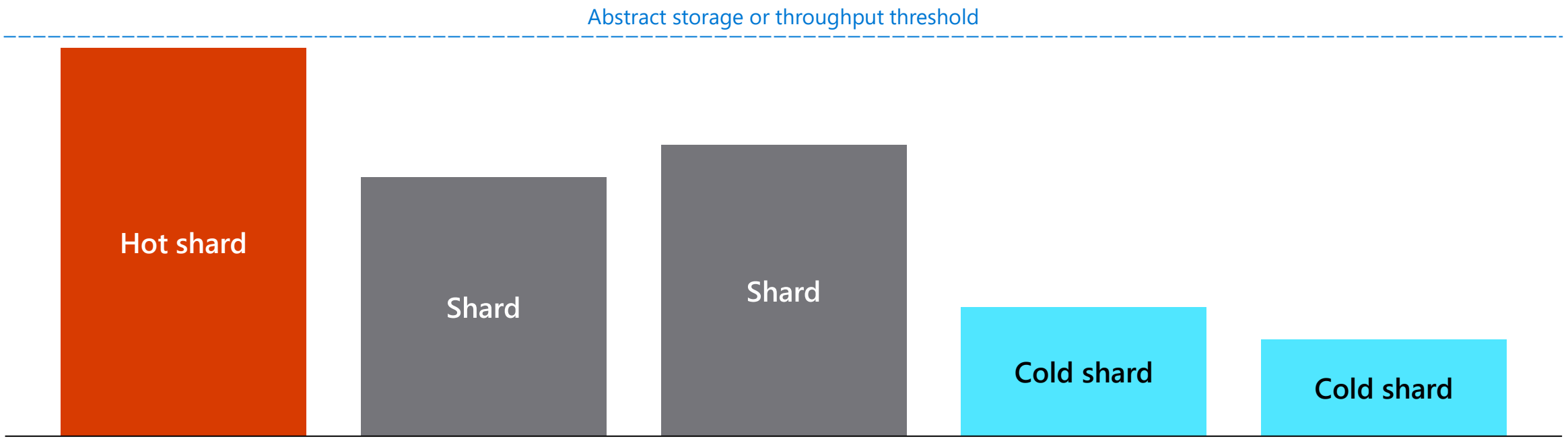
- As you insert new items into a container's shard, you can increment the unique id for each item in the shard
- When you get a 403 status code, indicating the shard is full, you can restart your unique id and upsert the items to replace older documents



HOT/COLD shards

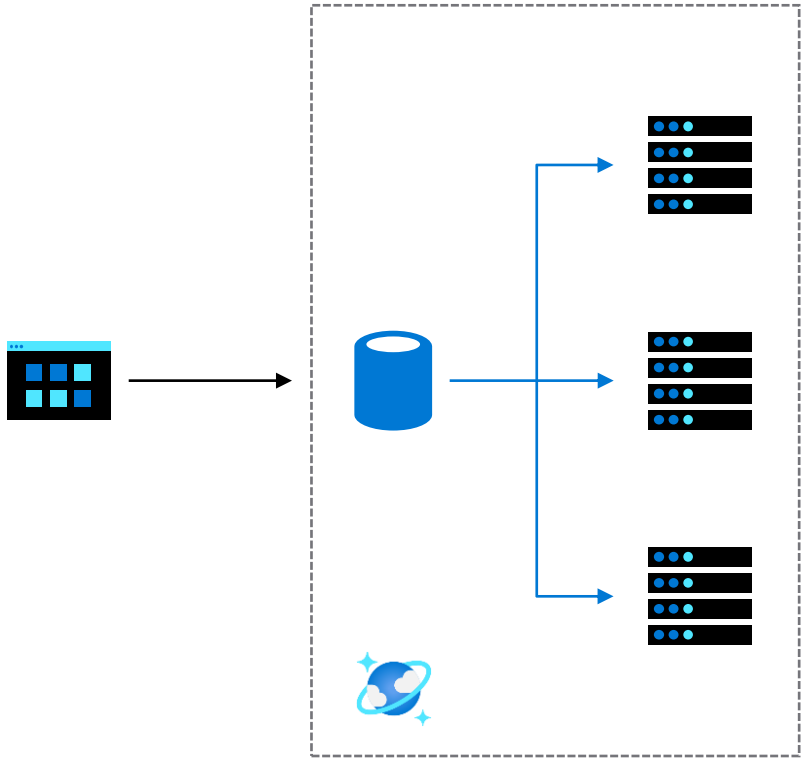
Shard usage can vary over time

- Shards that are approaching thresholds are referred to as hot
- Shards that are under-utilized are referred to as cold



Query fan-out

- Cross-shard queries can be performed server-side or client-side
  - Cross-shard queries are opt-in
  - Cross-shard queries can be tuned and parallelized
- Creates a bottleneck
  - Must wait for all shards to return before the query is “done”



Query fan-out continued

Querying across shards is not always a bad thing

- If you have relevant data to return, creating a cross-shard query is a perfectly acceptable workload with a predictable throughput
- In an ideal situation, queries are filtered to only include relevant shards

Blind query fan-outs can add up

- You are charged ~1 RU for each shard that doesn't have any relevant data.
- Multiple fan-out queries can quickly max out RU/s for each shard

Query fan-out continued

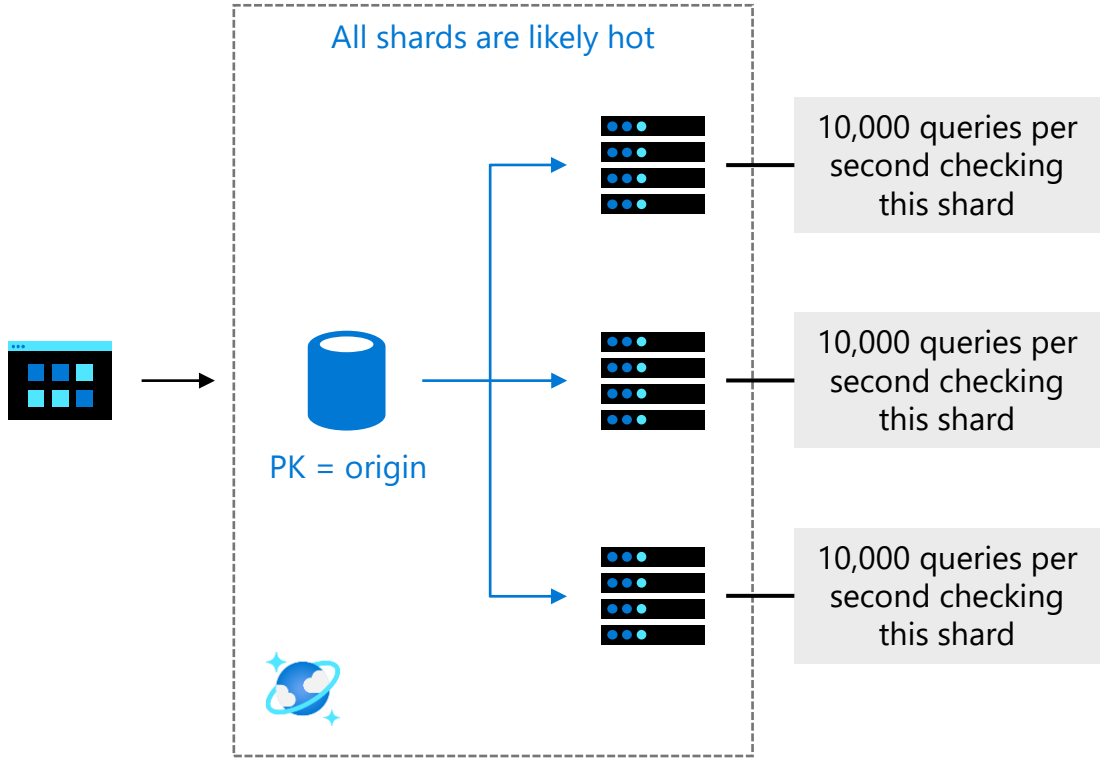
Concurrency and fan-out queries

- >10,000 fan-out queries in a second can leave all shards hot
- Example: Query on a vehicle database, sharded by model name, where the query is filtering by year without filtering to only include relevant shards.

```
db.collection.find({year: “2015”})
```

>10,000 more queries per second

```
db.collection.find({year: “2016”})
```



Query fan-out continued

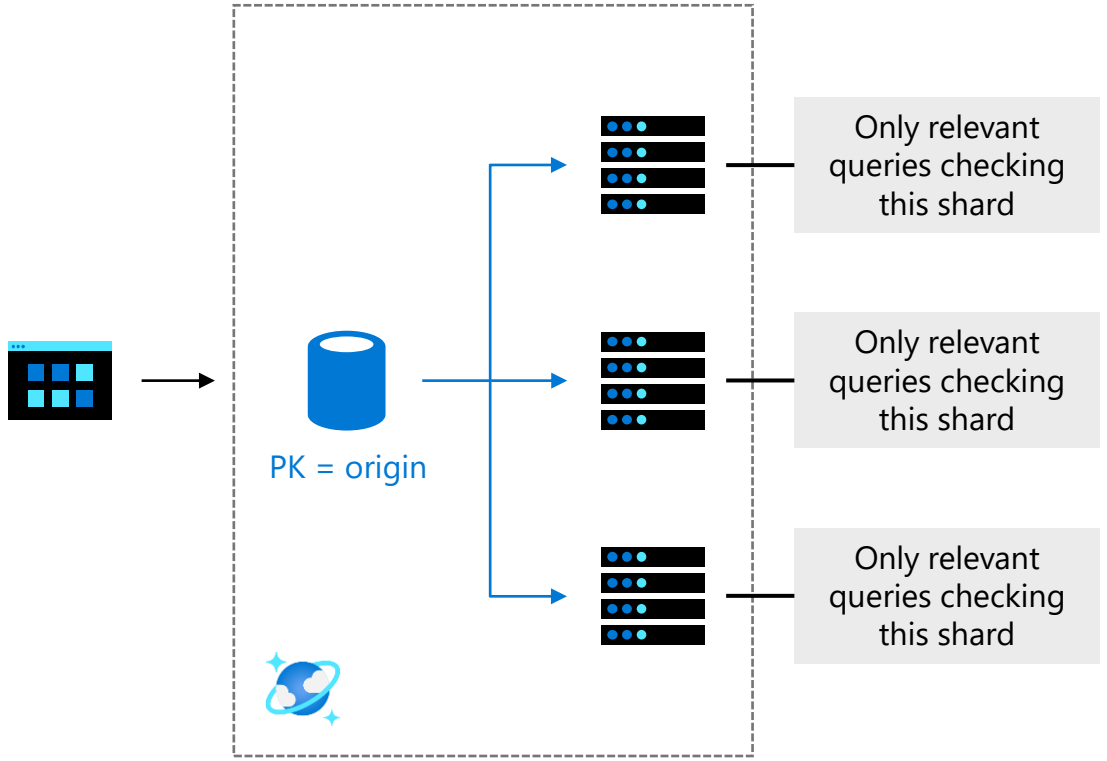
Concurrency and fan-out queries

- Filtering queries to only include relevant shard key values reduces the amount of wasted effort and focuses queries on those relevant shards.

```
db.collection.find({model: “TURLIC”, year: “2019”})
```

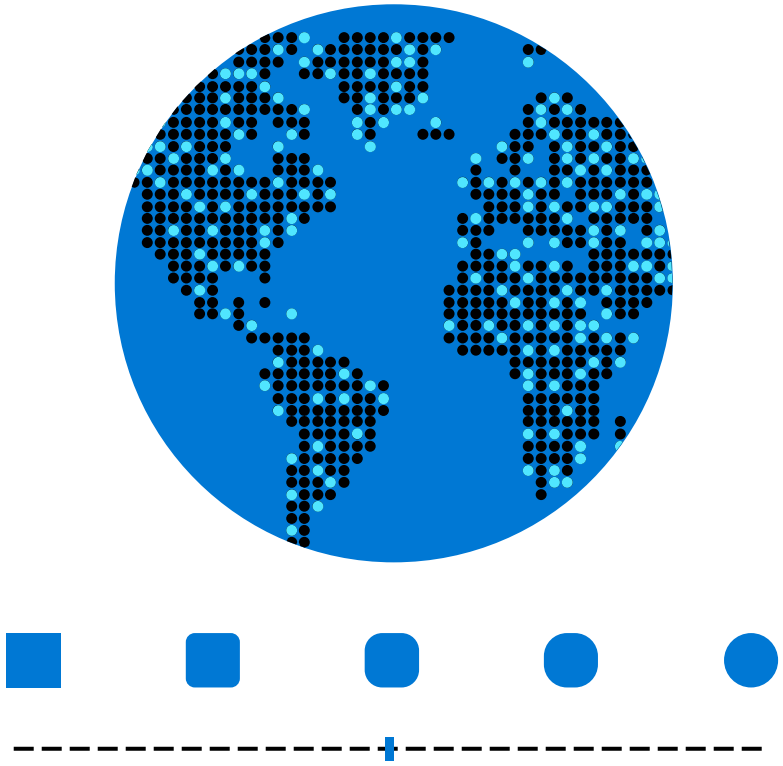
>10,000 more queries per second

```
db.collection.find({model: “COASH”, year: “2019”})
```



Modeling & Planning

- Modeling data & configuring collections to take advantage of Azure Cosmos DB strengths.



Considerations for data modelling

Although Azure Cosmos DB API for MongoDB has flexible schema, some planning will help achieve best results.

Focus on:

- Which fields your application wants to read or write together
- How can you achieve the best query performance

Note: Storage is cheaper than compute, so optimize for query performance over data duplication

Data modelling in NoSQL vs SQL

- In SQL, the design schema is independent of the queries you perform.
- In SQL, the prescribed approach is denormalization in 3rd form.

Embedding vs referencing

Embedding (De-normalizing)

- Retrieve all data with a single query
- Avoid joins with $lookup
- Update all data with a single atomic operation
- Data duplication
- May hit document size limit if too much data is embedded
- Larger document size may affect performance

Referencing (Normalizing)

- Need multiple queries to retrieve the entire data
- May need joins with $lookup
- Multiple writes needed
- No duplication of data
- More likely to be within size limit
- Smaller document size = better performance

How to decide whether to embed or reference?

Embed when

- 1:1 relationship
- 1:Few relationship
- Related items are queries or updated together

Referencing when

- 1:Many relationship (especially if unbounded)
- Many:Many relationship
- Relation items are queries or updated independently

Updating Normalized Data

Updates are atomic

- Update operations update the entire document, not specific fields or “parts” of the document.

De-normalized documents can be expensive to update

- De-normalization has benefits for read operations, but you must weigh this against the costs in write operations.
- De-normalization may require fanning out update operations.
- Normalization may require chaining a series of requests to resolve relationships.

```
{
    "id": "08259",
    "ticketPrice": 255.00,
    "flightCode": "3754",
    "origin": {
        "airport": "SEA",
        "gate": "A13",
        "departure": "2014-09-15T23:14:25.7251173Z"
    },
    "destination": {
        "airport": "JFK",
        "gate": "D4",
        "arrival": "2014-09-16T02:10:10.2379581Z"
    },
    "pilot": [{
        "id": "EBAMAO",
        “name": "Hailey Nelson"
    }]
}
```

Updating Normalized Data continued

Normalized: Optimized for writes over reads

```
{
     "id": "08259",
     "pilot": [{ "id": "EBAMAO", “name": "Hailey Nelson“ }]
},
{
     "id": "08259",
     "ticketPrice": 255.00,
     "flightCode": "3754"
},
{
     "id": "08259",
     "origin": {
          "airport": "SEA", "gate": "A13",
          "departure": "2014-09-15T23:14:25.7251173Z"
     },
     "destination": {
          "airport": "JFK", "gate": "D4",
          "arrival": "2014-09-16T02:10:10.2379581Z"
     }
}
```

De-normalized: Optimized for reads over writes

```
{
     "id": "08259",
     "ticketPrice": 255.00,
     "flightCode": "3754",
     "origin": {
          "airport": "SEA",
          "gate": "A13",
          "departure": "2014-09-15T23:14:25.7251173Z"
     },
     "destination": {
          "airport": "JFK",
          "gate": "D4",
          "arrival": "2014-09-16T02:10:10.2379581Z"
     },
     "pilot": [{
          "id": "EBAMAO",
          “name": "Hailey Nelson"
     }]
}
```

Updating Normalized Data continued

The solution is typically a compromise based on your workload

- Examine your workload. Answer the following questions:
  - Which fields are commonly updated together?
  - What are the most common fields included in all queries?
- Example: The `ticketPrice`, `origin` and `destination` fields are often updated together. The `pilot` field is only rarely updated. The `flightCode` field is included in almost all queries across the board.

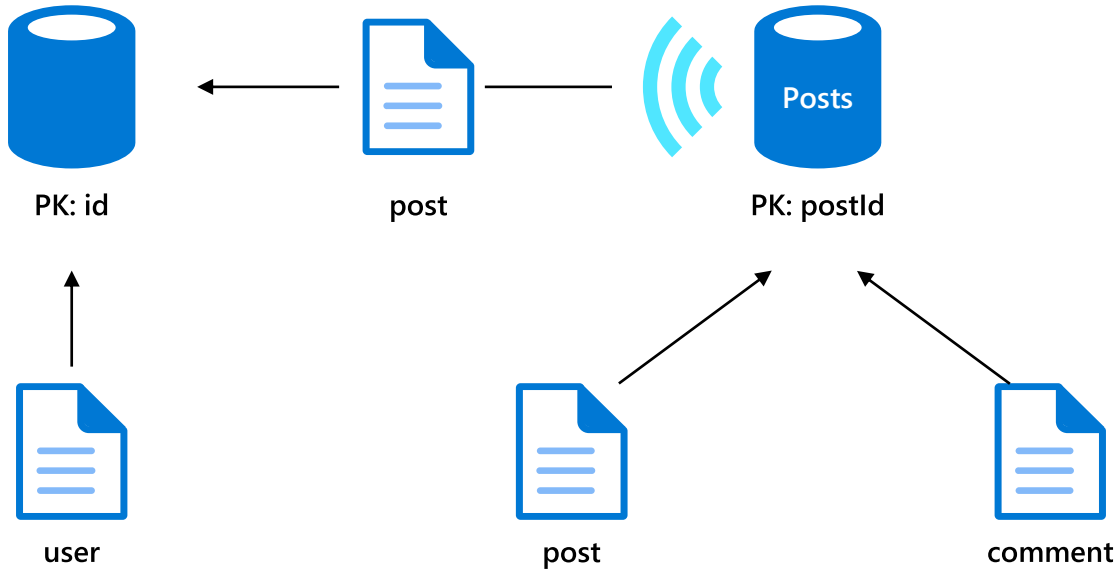
```
{
     "id": "08259",
     "flightCode": "3754",
     "pilot": [{ "id": "EBAMAO", “name": "Hailey Nelson" }]
},
{
     "id": "08259",
     "flightCode": "3754",
     "ticketPrice": 255.00,
     "origin": {
          "airport": "SEA", "gate": "A13",
          "departure": "2014-09-15T23:14:25.7251173Z"
     },
     "destination": {
          "airport": "JFK", "gate": "D4",
          "arrival": "2014-09-16T02:10:10.2379581Z"
     }
}
```

Limits to remember

Document size limit: 2 MB

Logical Shard limit: 20 GB

Use change stream to update duplicate fields



Mixing entity types

Collections

- Collections do NOT enforce schema
- There are benefits to co-locate multiple types in a collection
- Annotate records with a "type" property

Co-locating types in the same collection

- Ability to query across multiple entity types with a single network request.
- Ability to perform transactions across multiple types
- Cost: reduce physical partition footprint

Co-locating types

Ability to query across multiple entity types with a single network request
For example, we have two types of documents: cat and person.

```
{
        "id": "Andrew",
        "type": "Person",
        "familyId": "Liu",
        "worksOn": "Azure Cosmos DB"
}
```

```
{
        "id": "Ralph",
        "type": "Cat",
        "familyId": "Liu",
        "fur": {
                "length": "short",
                "color": "brown"
        }
}
```

We can query both types of documents without needing a JOIN simply by running a query without a filter on type:

```
db.collection.find({familyId: “Liu”})
```

If we wanted to filter on `type = “Person”`, we can simply add a filter on type to our query:

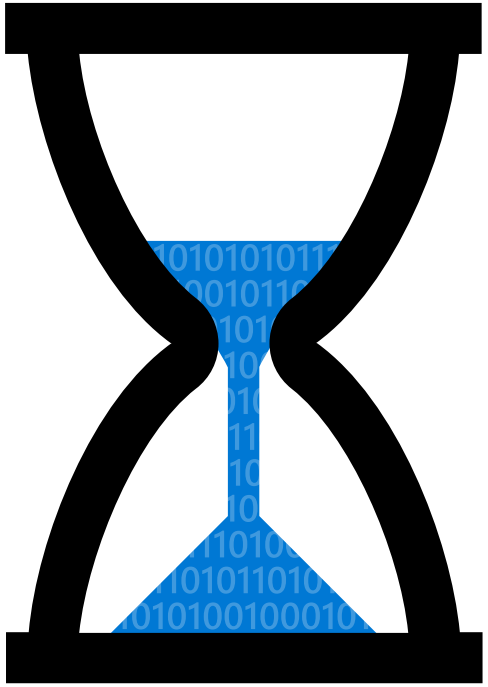
```
db.collection.find({familyId: “Liu”, type: “Person”})
```

Short-lifetime data

Some data produced by applications are only useful for a finite period of time:

- Machine-generated event data
- Application log data
- User session information

It is important that the database system systematically purges this data at pre-configured intervals.



Time-to-Live (TTL)

Automatically purge data

Azure Cosmos DB allows you to set the length of time in which documents live in the database before being automatically purged. A document's "time-to-live" (TTL) is measured in seconds from the last modification and can be set at the collection level with override on a per-document basis.

The TTL value is specified in the _ts field which exists on every document.

- The _ts field is a UNIX-style epoch timestamp representing the date and time. The _ts field is updated every time a document is modified.

Once TTL is set, Azure Cosmos DB will automatically remove documents that exist after that period of time.

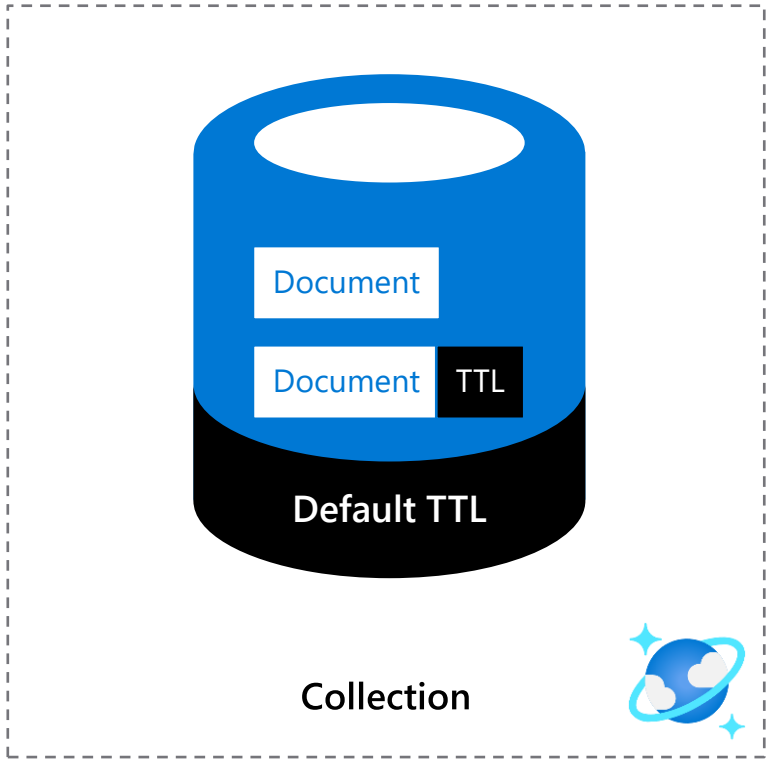
Expiring records using Time-to-Live

TTL behavior

The TTL feature is controlled by TTL properties at two levels – the collection level and the document level.

- DefaultTTL for the collection
  - If missing (or set to null), documents are not deleted automatically.
  - If present and the value is "-1" = infinite: documents don't expire by default
  - If present and the value is some number ("n"): documents expire "n" seconds after last modification
- TTL for the documents:
  - Property is applicable only if DefaultTTL is present for the parent collection.
  - Overrides the DefaultTTL value for the parent collection.

The values are set in seconds and are treated as a delta from the _ts that the document was last modified at.



Turnkey Global Distribution

High Availability

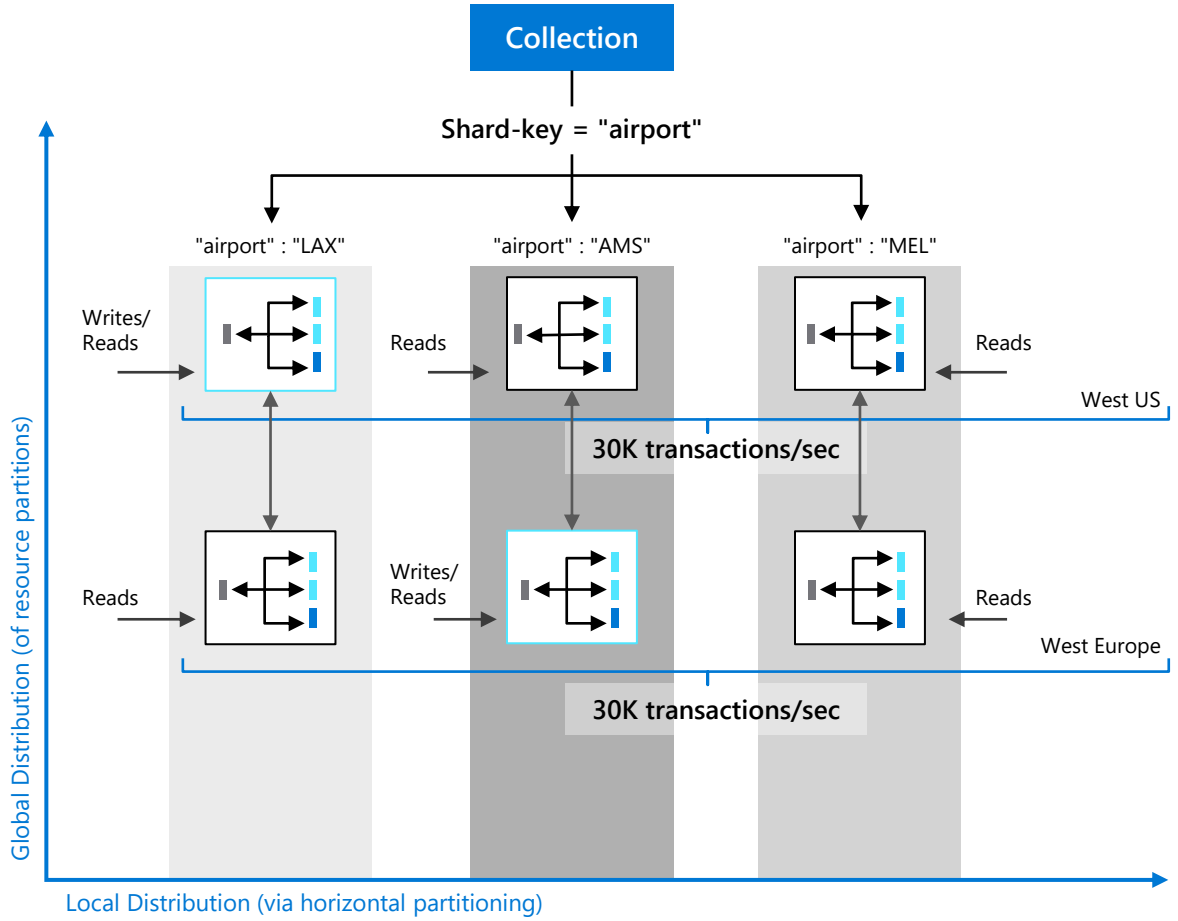
- Automatic and Manual Failover
- Multi-homing API removes need for app redeployment

Low Latency (anywhere in the world)

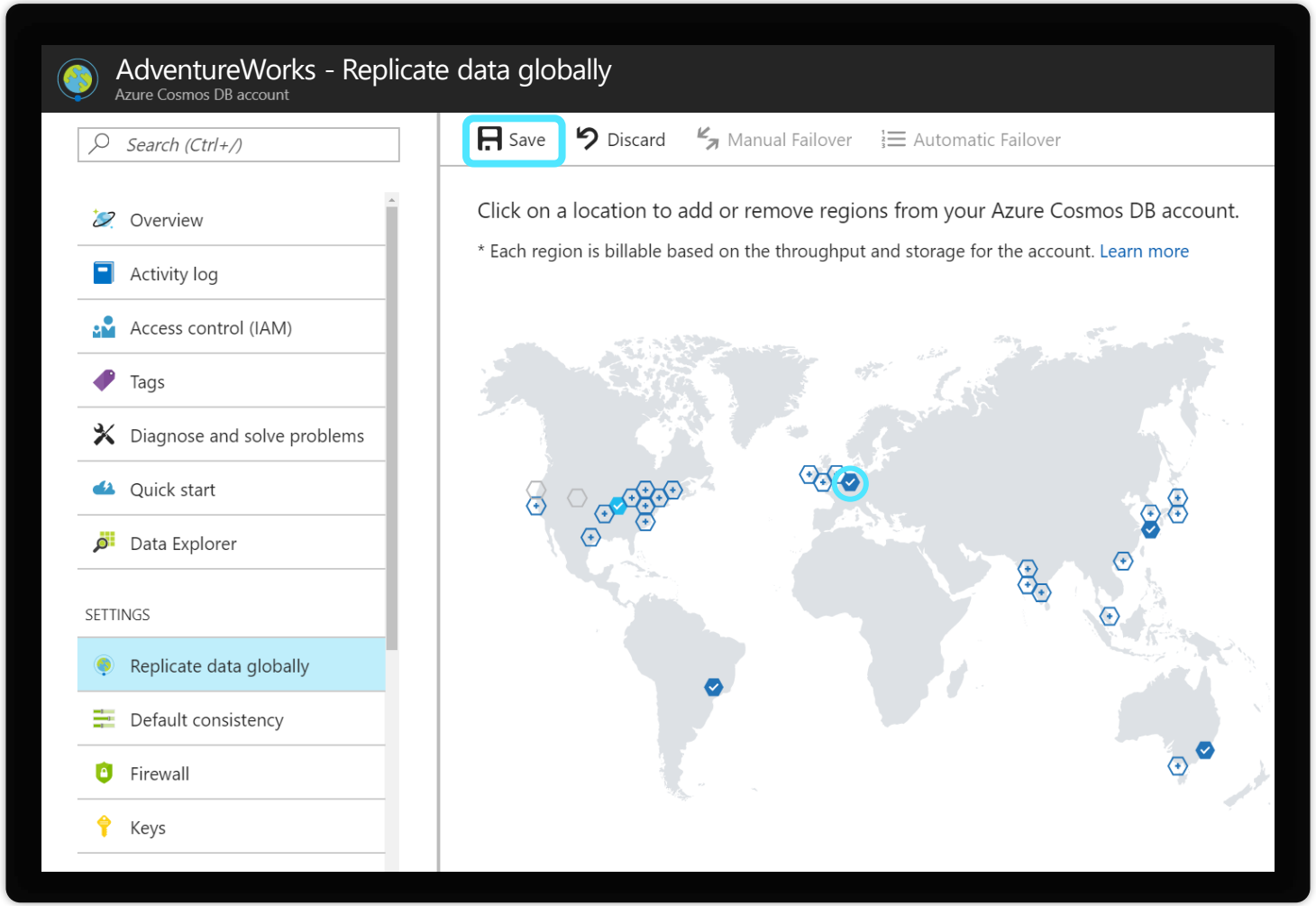
- Packets cannot move fast than the speed of light
- Sending a packet across the world under ideal network conditions takes 100's of milliseconds
- You can cheat the speed of light – using data locality
  - CDN's solved this for static content
  - Azure Cosmos DB solves this for dynamic content

Turnkey Global Distribution continued

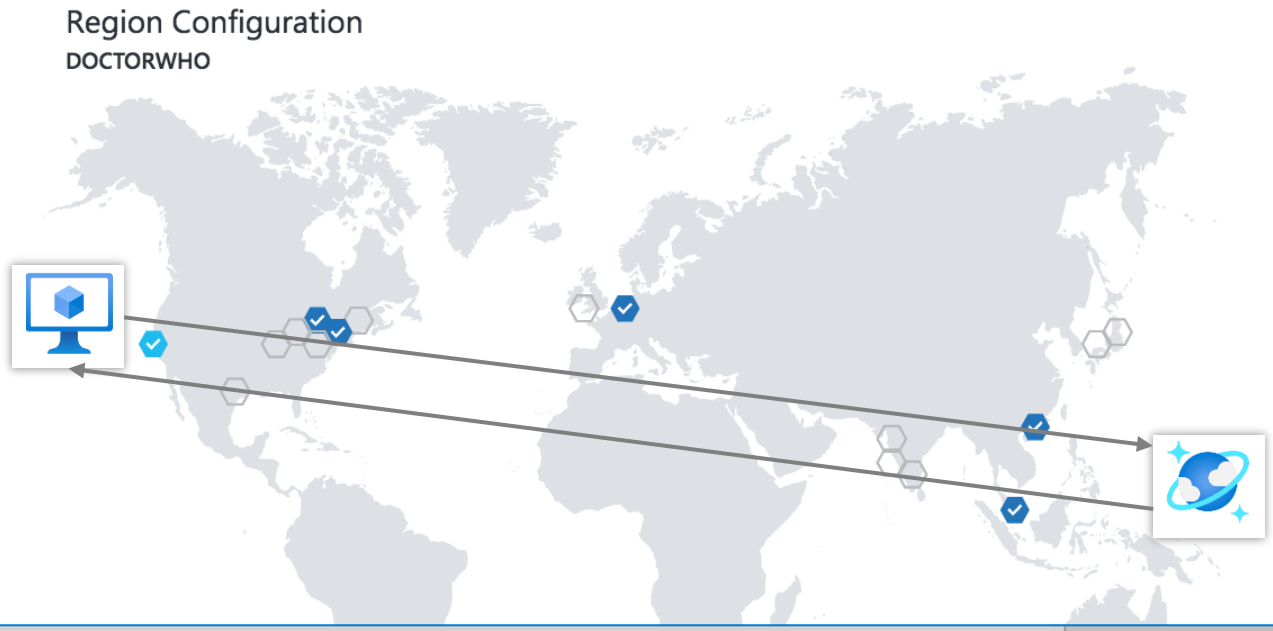
- Automatic and transparent replication worldwide
- Each partition hosts a replica set per region
- Customers can test end to end application availability by programmatically simulating failovers
- All regions are hidden behind a single global URI with multi-homing capabilities
- Customers can dynamically add / remove additional regions at any time



Replicating Data Globally

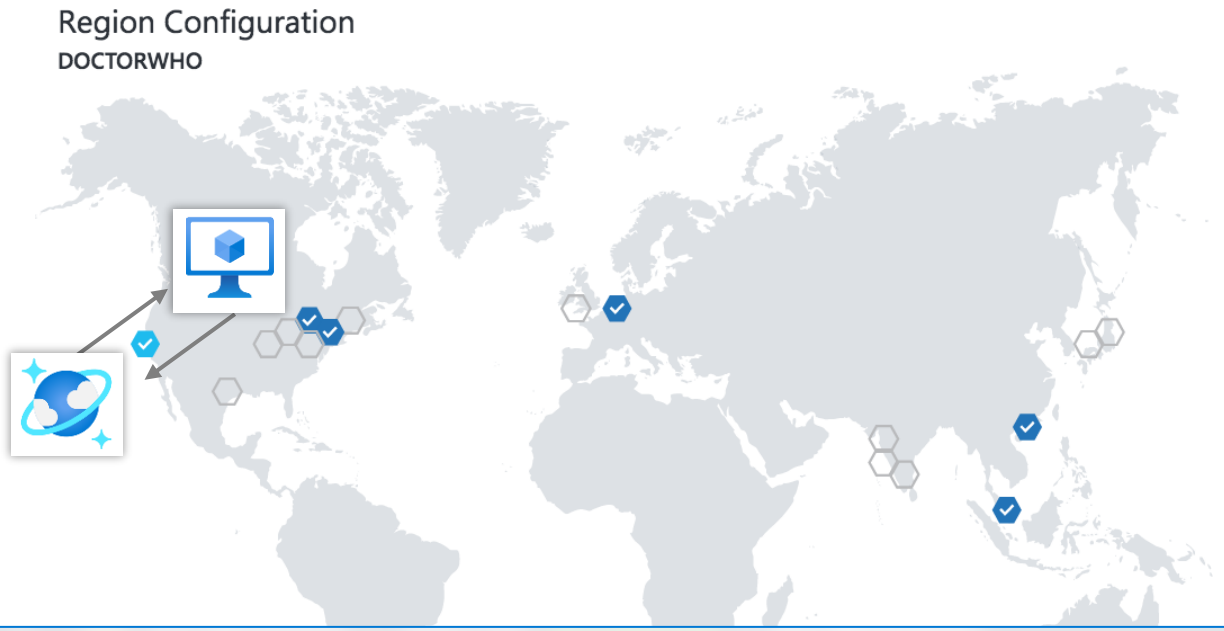


Replicating Data Globally continued



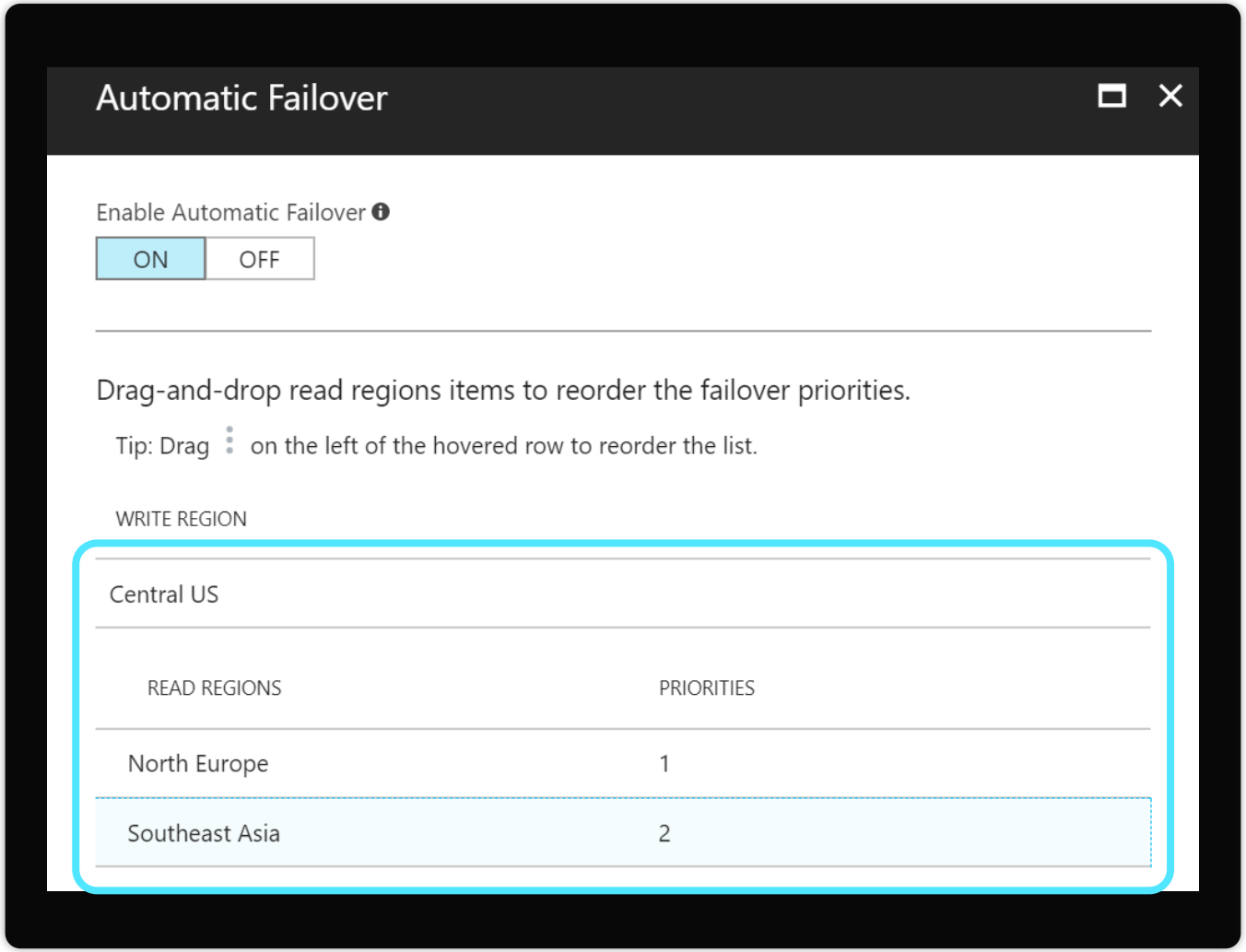

```
...rkspace/docdb/docdb — amypond@blink: ~/docdb — ssh amypond@104.42.108.173 -p 22
[amypond@blink:~/docdb$ node testQ2.js
210.788804 milliseconds, 2 RUs, ActivityId: 9025eac6-eb74-4a07-94cf-f2383caffbb3
178.825773 milliseconds, 2 RUs, ActivityId: ea53b736-b629-4290-9cdf-cf80c6139461
178.839173 milliseconds, 2 RUs, ActivityId: f143c992-ba67-4c7b-b7b8-0b5db8df4dbf
178.564573 milliseconds, 2 RUs, ActivityId: 1a8d7b5b-42c5-4c39-9160-d1e5ab2200d0
179.229073 milliseconds, 2 RUs, ActivityId: 483b85de-74e0-4f48-9206-70ac1268c60e
178.653772 milliseconds, 2 RUs, ActivityId: 50fbfe91-f41e-4f14-8a15-0b344c894727
178.464572 milliseconds, 2 RUs, ActivityId: cac6446a-79d4-4886-81d8-1dda835daa72
180.708475 milliseconds, 2 RUs, ActivityId: d40af8e4-582b-4479-bb9c-e3582eac6774
```

Replicating Data Globally continued

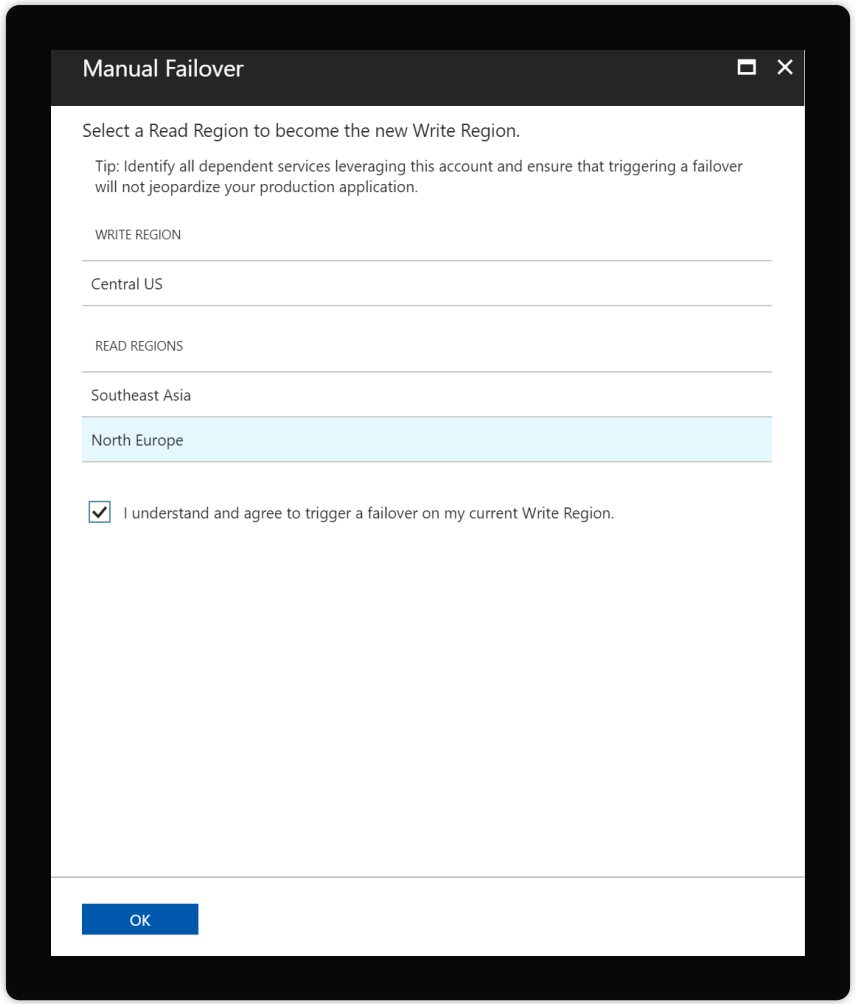



```
[amypond@blink:~/docdb$ node testQ2.js
12.736112 milliseconds, 2 RUs, ActivityId: dd3c17b9-1b76-445a-8e27-29b7486bd7e4
4.947605 milliseconds, 2 RUs, ActivityId: e2f4c899-9fb1-4f76-a4ab-e5718fac5742
5.044005 milliseconds, 2 RUs, ActivityId: 0fc5d216-78a0-4d92-a3d0-63efd9dd6552
5.351205 milliseconds, 2 RUs, ActivityId: 861155f0-81ba-4c8a-9933-e50d8708cc21
4.553505 milliseconds, 2 RUs, ActivityId: 3db9641f-70f1-4ef1-84bb-809280bbe1a5
5.427506 milliseconds, 2 RUs, ActivityId: 10d1b2e5-e795-4c77-8655-815f410ba11e
5.900106 milliseconds, 2 RUs, ActivityId: bda1df86-c5ad-45c5-bcfb-93d417c54751
4.895405 milliseconds, 2 RUs, ActivityId: f206d58d-64a2-47f2-9653-145eaf47ff97
5.244306 milliseconds, 2 RUs, ActivityId: 3aa7e177-b1a9-413d-8023-55f5054d1b74
```

Automatic Failover



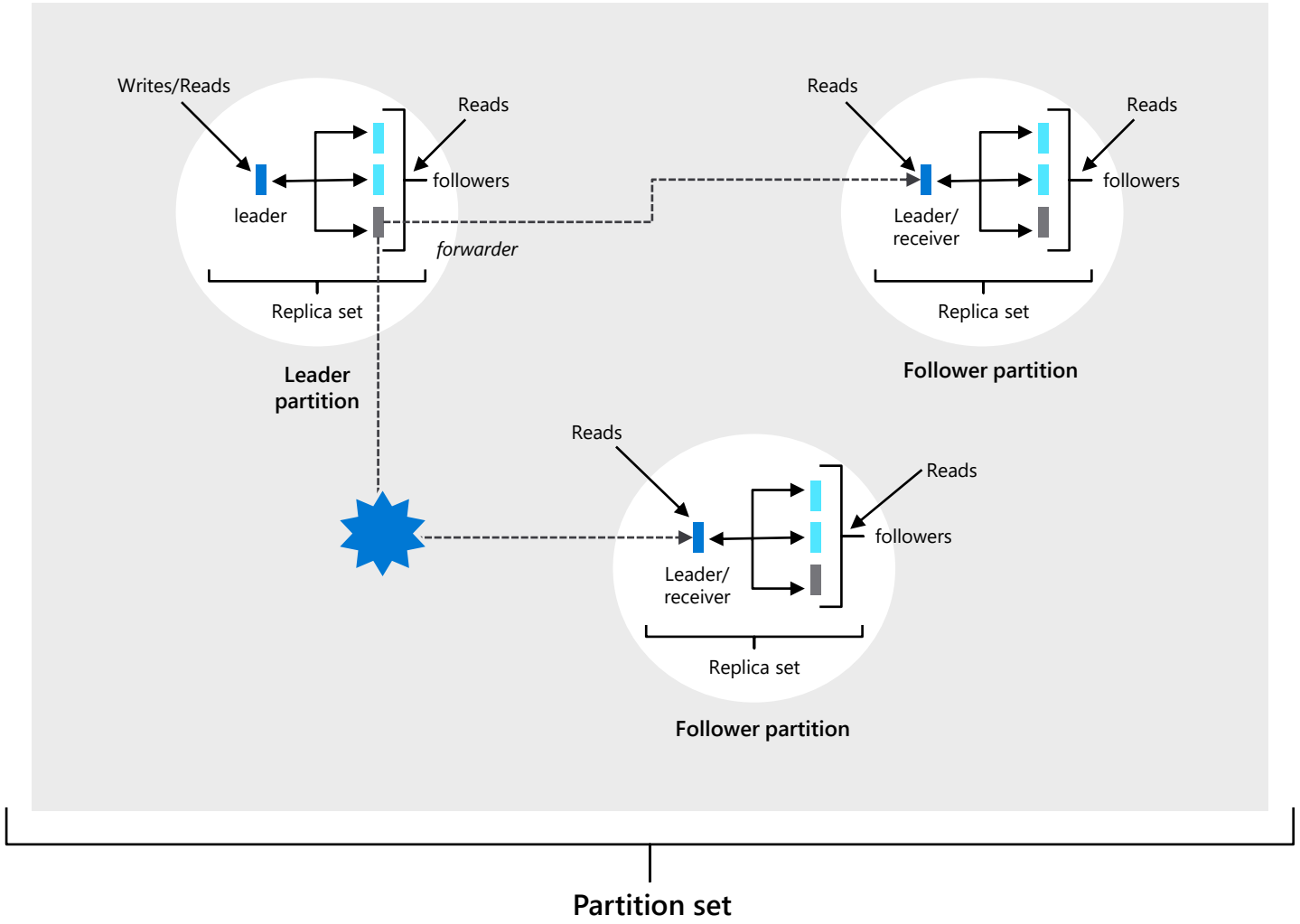
Manual Failover



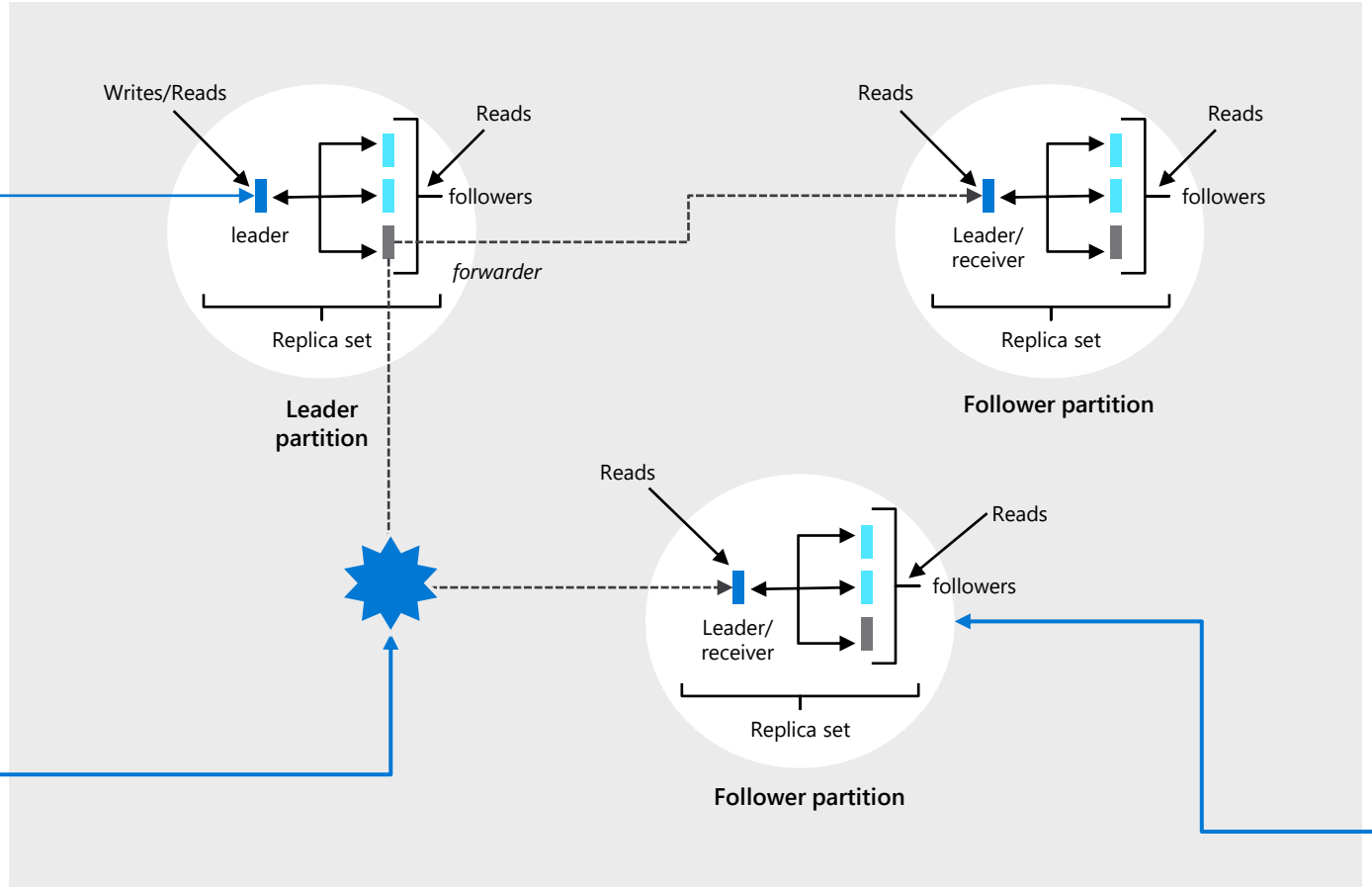
Brewer's CAP Theorem

Impossible for distributed data store to simultaneously provide more than 2 out of the following 3 guarantees:

- Consistency
- Availability
- Partition Tolerance



Consistency



(West US)
Value = ~~5~~
Update 5 ≥ 6

(East US)
Value = ~~5~~ 6

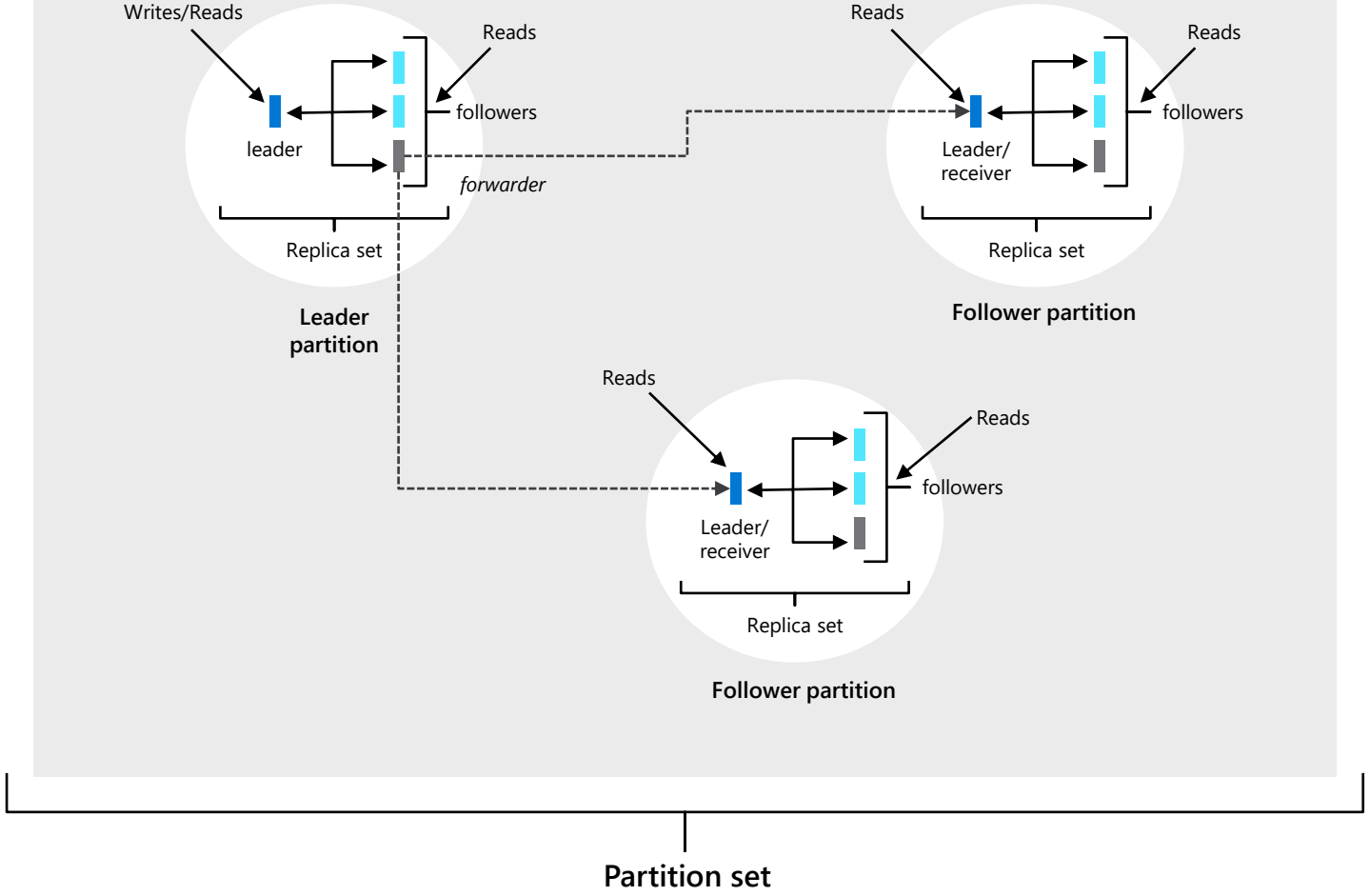
(North Europe)
Value = 5

What happens when a network partition is introduced?

Reader: What is the value? Should it see 5? (prioritize availability)
Or does the system go offline until network is restored? (prioritize consistency)

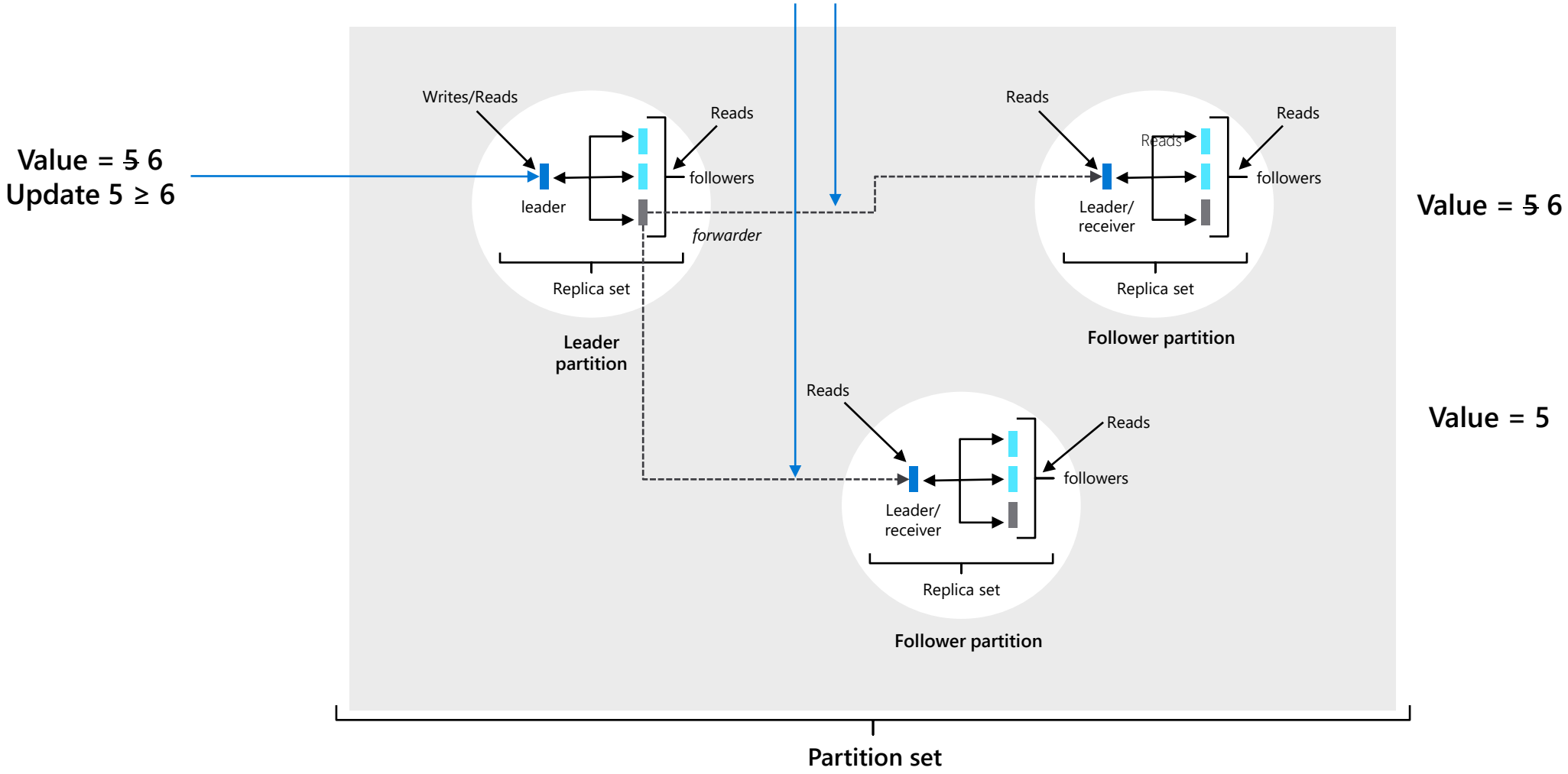
PACELC Theorem

In the 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).

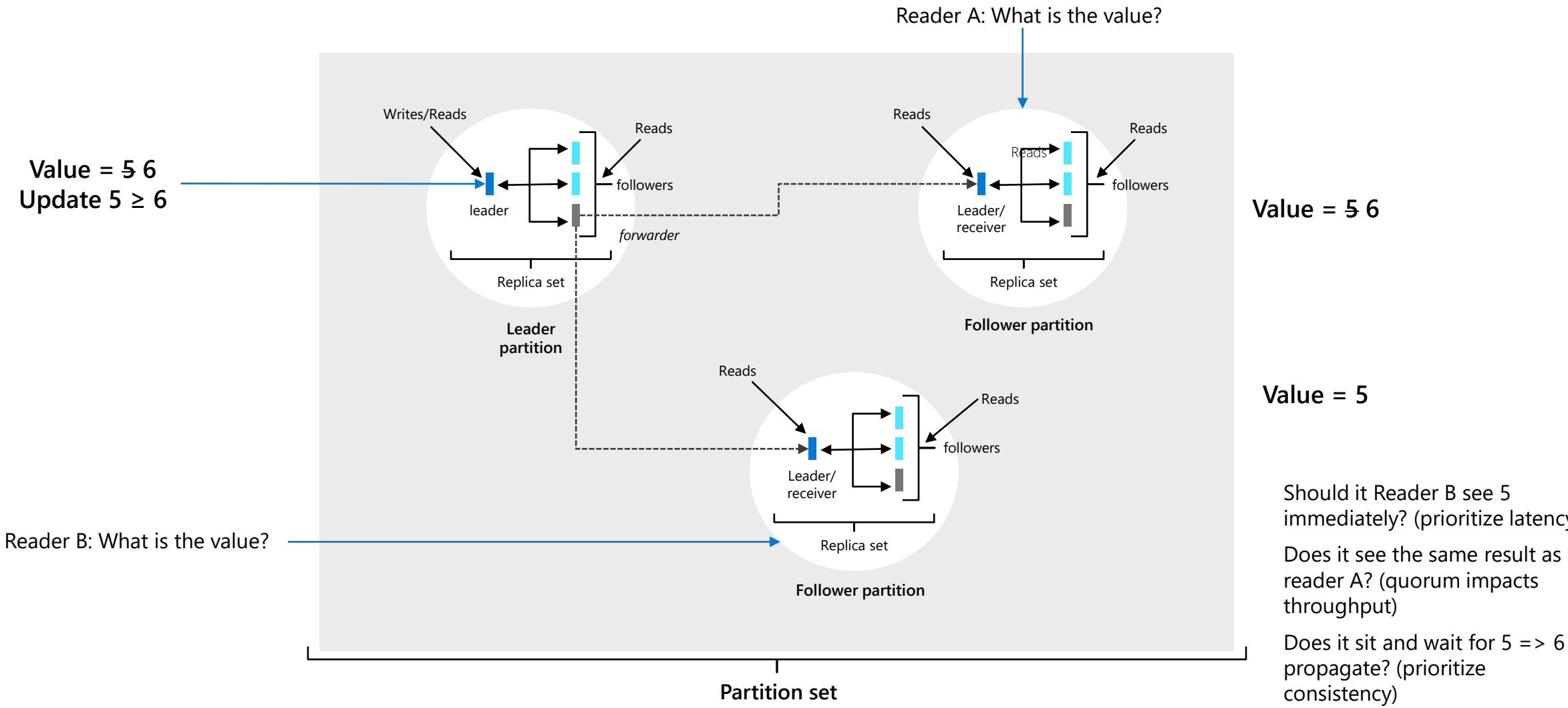


Consistency

Latency: packet of information can travel as fast as speed of light. Replication between distant geographic regions can take 100s of milliseconds.



Consistency continued



Should it Reader B see 5 immediately? (prioritize latency)

Does it see the same result as reader A? (quorum impacts throughput)

Does it sit and wait for 5 => 6 propagate? (prioritize consistency)

Five well-defined consistency models

Choose the best consistency model for your app

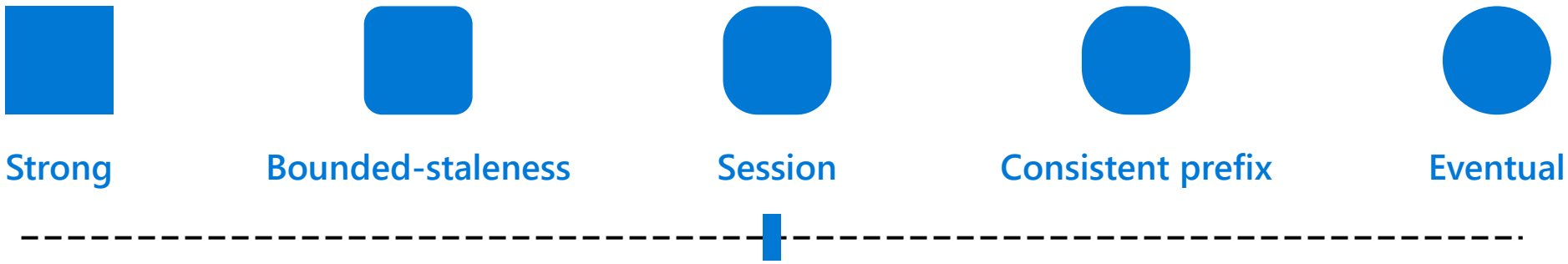
Five well-defined, consistency models

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



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.

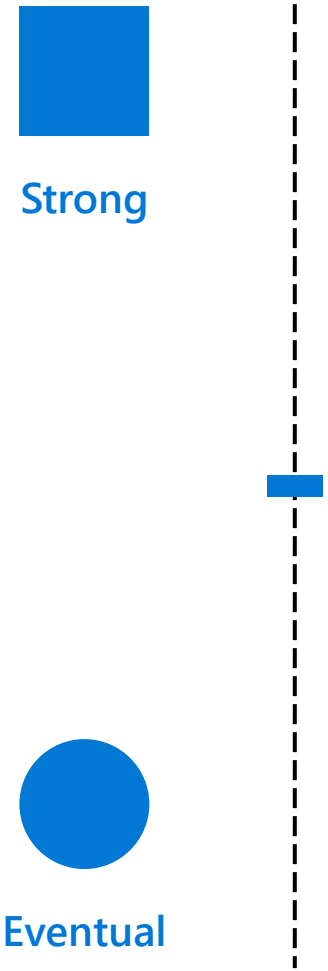
Demystifying consistency models

Strong consistency

- Guarantees linearizability. Once an operation is complete, it will be visible to all readers in a strongly-consistent manner across replicas.

Eventual consistency

- 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.



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.

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.

Demystifying consistency models continued

Bounded-staleness

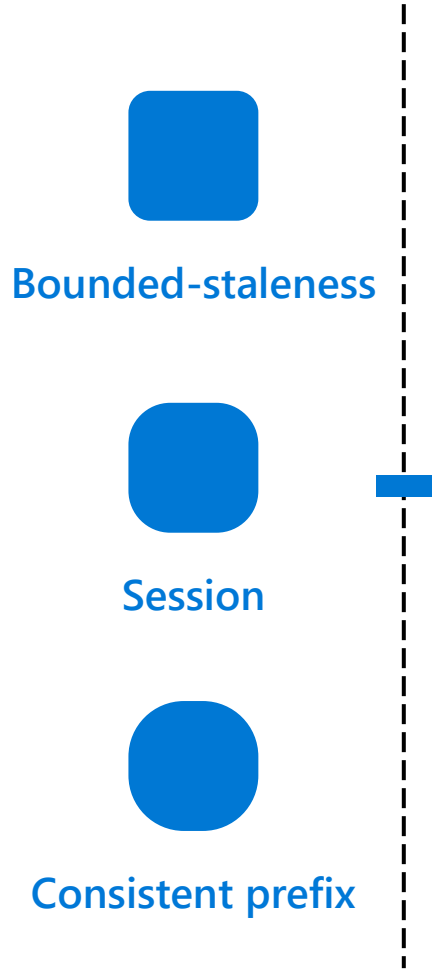
- Consistent prefix. Reads lag behind writes by at most k prefixes or t interval. Similar properties to strong consistency except within staleness window.

Session

- 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.

Consistent Prefix

- Reads will never see out of order writes.



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.

Session consistency

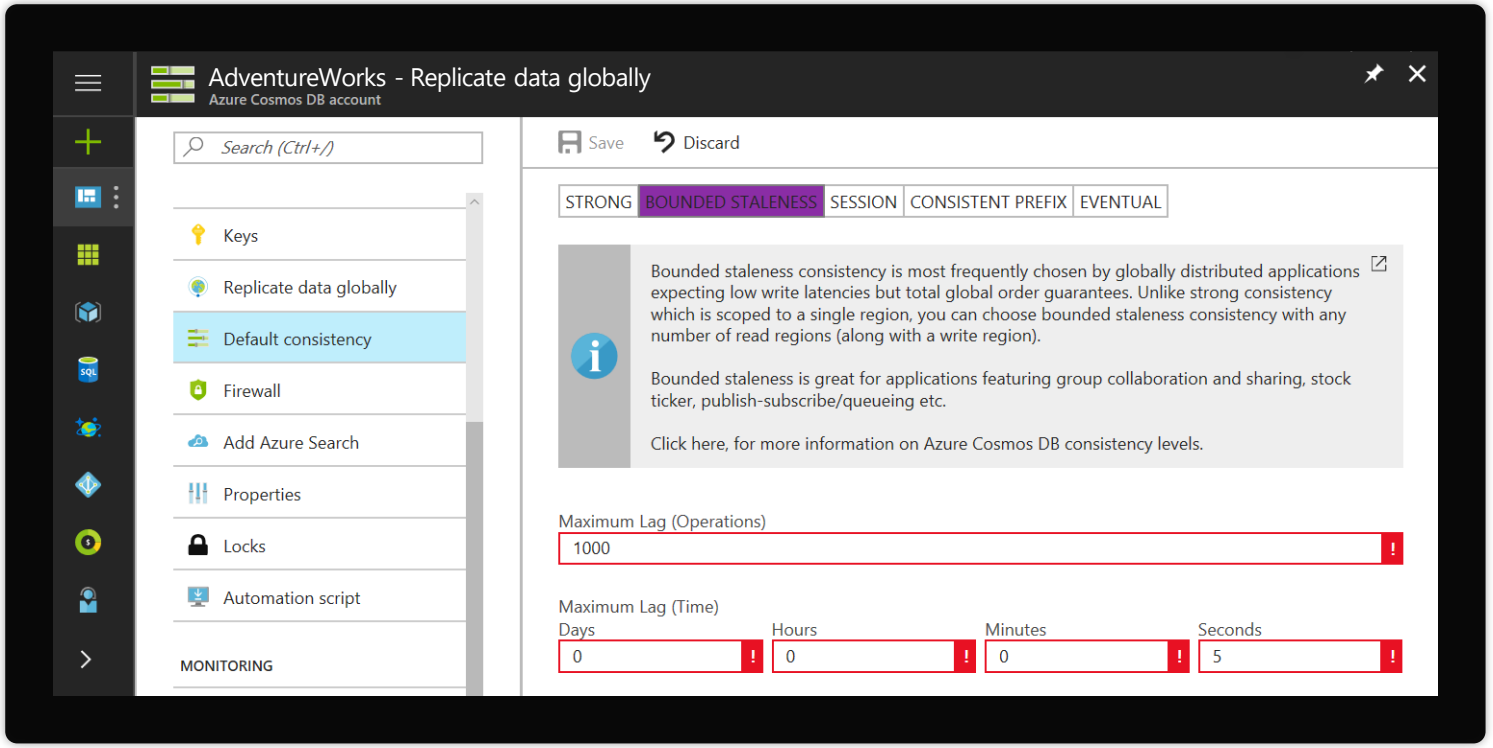
- Unlike the global consistency models offered by strong and bounded staleness consistency levels, session consistency is scoped to a client session.
- Session consistency is ideal for all scenarios where a device or user session is involved since it guarantees monotonic reads, monotonic writes, and 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 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**.

Bounded staleness in the portal

Bounds are set server-side via the Azure portal



Bounded-staleness

Mapping Native Mongo read-write concerns with Azure Cosmos DB consistency levels

MongoDB Write Concern	Default Consistency (set on the Cosmos account)	Region serving the reads	MongoDB Read Concern	Cosmos Consistency Level A(dynamically set on a read request)	Guarantees offered by native MongoDB	Guarantees offered by Cosmos MongoDB API
MAJORITY (W)	Strong	*IsMaster* = true	LOCAL, AVAILABLE	Bounded Staleness	Dirty Reads (Read Uncommitted) in steady state; Stale reads under split brain (network partitioning)	Linearizable reads in the region where the write is performed Bounded staleness in all other regions
			MAJORITY	Bounded Staleness	Linearizability in steady state; Stale reads under split brain (network partitioning)	Linearizable reads in the region where the write is performed Bounded staleness (*time-bounded* linearizability) in all other regions
			LINEARIZABILITY	Strong	Linearizability	Linearizability
			SNAPSHOT	Strong	Atomic reads across documents	Linearizability
		IsMaster = false	LOCAL, AVAILABLE	Consistent Prefix	Stale Reads; Dirty Reads	Consistent Prefix
			MAJORITY	Strong	Stale Reads	Linearizability
			LINEARIZABILITY	Not Allowed	Not Allowed	Not Allowed
			SNAPSHOT	Not Allowed	Not Allowed	Not Allowed
MINORITY (1 < W < Majority)	Bounded Staleness	*IsMaster* = true	LOCAL, AVAILABLE	Bounded Staleness	Dirty Reads in steady state; Stale reads under network partitioning	Linearizable reads in the region where the write is performed Bounded staleness (*time-bounded* linearizability) in all other regions
			MAJORITY	Bounded Staleness	Linearizability in steady state; Stale reads under network partitioning	Linearizable reads in the region where the write is performed Bounded staleness (*time-bounded* linearizability) in all other regions
			LINEARIZABILITY	Bounded Staleness	Linearizability	Linearizable reads in the region where the write is performed Bounded staleness (*time-bounded* linearizability) in all other regions
			SNAPSHOT	Bounded Staleness	Linearizability	Linearizable reads in the region where the write is performed Bounded staleness (*time-bounded* linearizability) in all other regions
		IsMaster = false	LOCAL, AVAILABLE	Consistent Prefix	Stale Reads; Dirty Reads	Consistent Prefix
			MAJORITY	Bounded Staleness	Stale Reads; Monotonic Reads	Bounded staleness configured via *K* and *T* along with monotonicity
			LINEARIZABILITY / SNAPSHOT	Not Allowed	Not Allowed	Not Allowed
None (W=0)	Consistent Prefix	*IsMaster* = true/false	ANY	Consistent Prefix	Stale Reads; Dirty Reads	Consistent Prefix

Pricing

Charges for database operations and consumed storage

Autoscale operations are charged at 1.5 times

Backup storage is charged for >2 copies (default 2 copies are free)

Continuous backup & point-in-time restore are in preview and are charged per GB

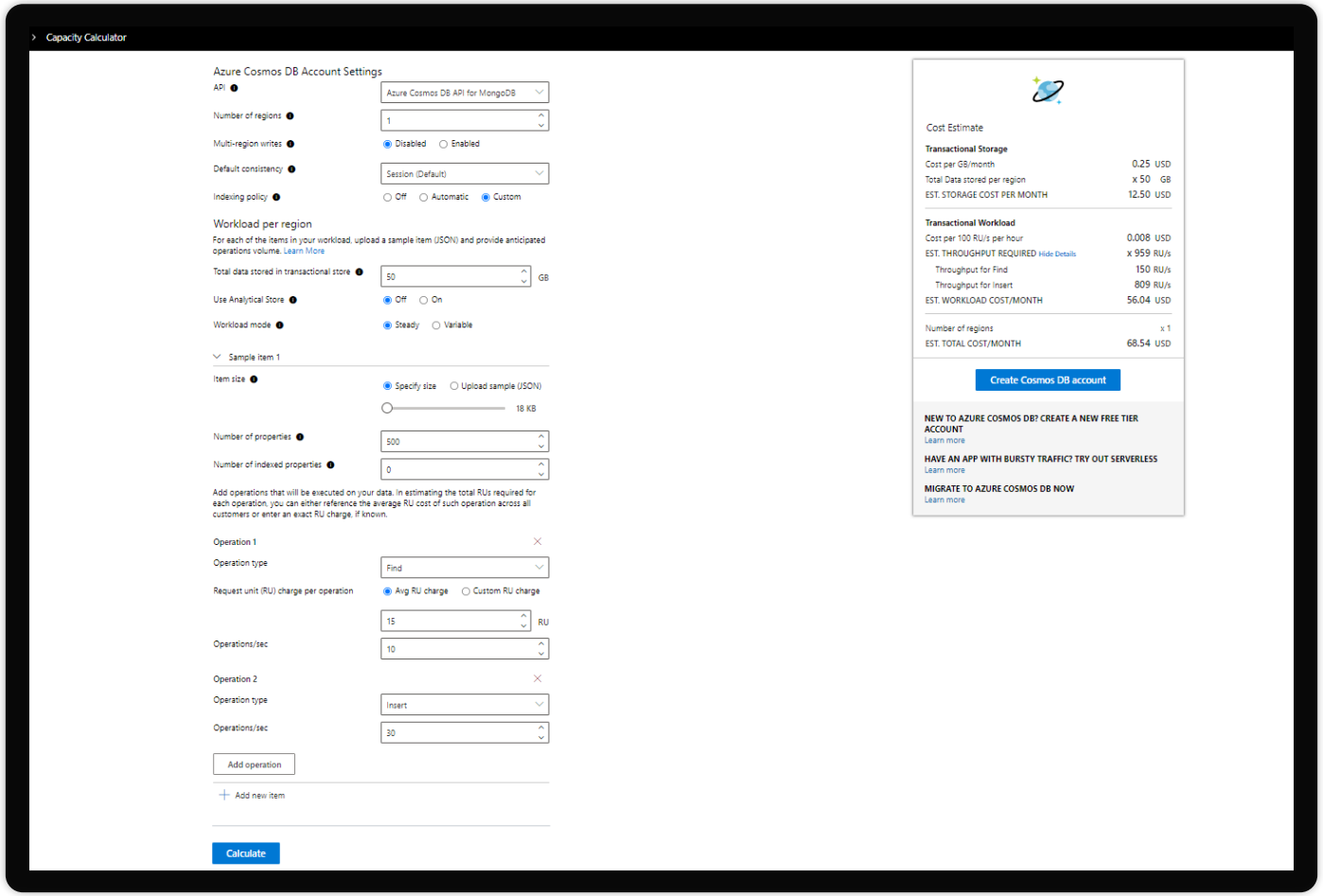
IO transactions for analytical storage are billed by quantity of operations in multiples of 10,000

Reserved capacity discounts available for 1 & 3 years

More details: azure.microsoft.com/en-in/pricing/details/cosmos-db/

Use Azure Cosmos DB calculator for estimating costs

cosmos.azure.com/capacitycalculator/



Know your workload

Volume of requests per request type

Access patterns

- Distribution of requests - read vs write %
- Types of queries used
- Whether the workload is going to be consistent or variable
- % of time spent at peak vs off-peak workload

Find RU charges for Find & Aggregation operations as suggested in Query Tuning section & provide them as inputs to the calculator

Calculator tips

Sign-in to utilize all calculator features.

Write charges are calculated based upon the number of regions, multi-region writes, consistency levels, indexing policy. Provide all these inputs.

Add operation types from Find, Find & Modify, Aggregate, Insert, Update, Delete and specify number of operations for each.

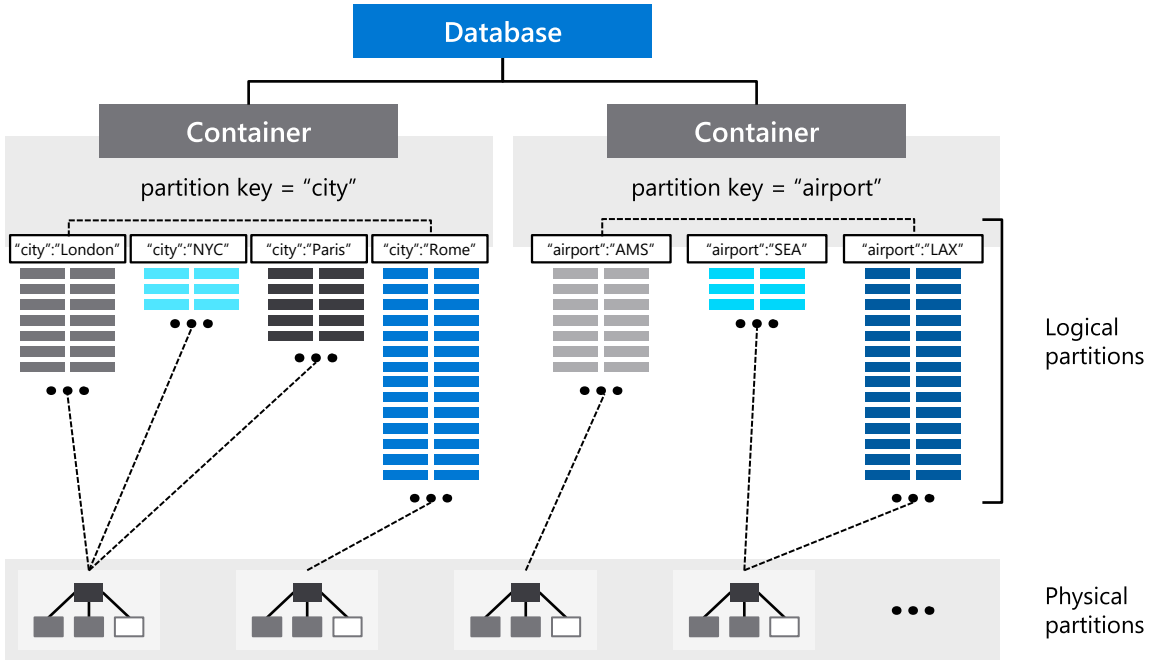
You may upload a sample json document for running estimations against.

Shared vs dedicated throughput

Collections with Dedicated RU

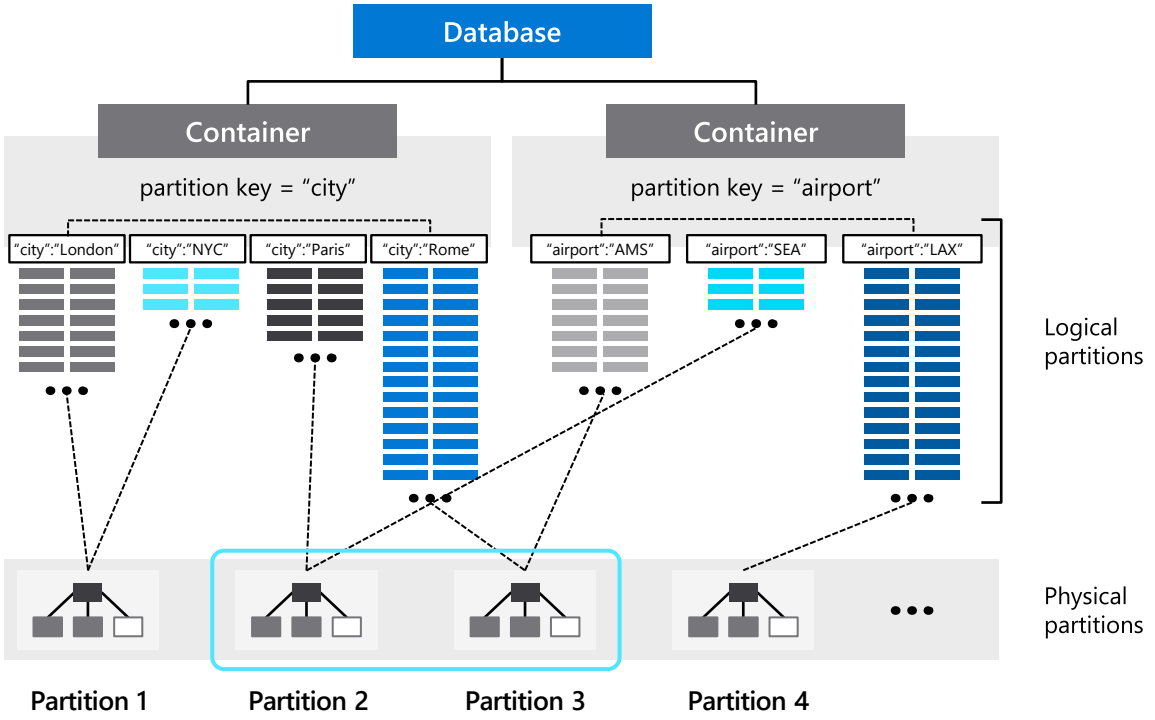
Container1: M RUs

Container2: N RUs



Collections with Database RU

Container 1 and 2 share: M RUs



Minimum RU limitations

Once a container is storing any data, minimum throughput is 10 RU/s per GB

Once a container has provisioned X RU/s, the future minimum throughput is 1%*X RU/s

Minimum provisioned throughput on a container is 400 RU/s

Autoscale can vary between 10% & 100% of the max throughput

In Autoscale, the RU limits apply to the minimum value of the throughput range

Hands-on exercise

Creating an account & database

Tasks:

1. **Create an account for Azure Cosmos DB API for MongoDB**
   - Account Name should be globally unique
   - Choose region closest to you
   - Use ‘Provisioned Throughput’ mode
   - Use API 3.6 (default is 4.0, choose 3.6 so that we can upgrade to 4.0 in a later lab)
   - Closely view & understand rest of the defaults. Keep the default settings.
2. **Create a database inside the account**
   - Go to ‘Data Explorer’ tab for the account in the portal.
   - Click on the arrow next to ‘New Collection’ to find the ‘New Database’ button
   - Use ‘provision throughput’ option to share throughput across collections
   - Use ‘Autoscale’ mode

Hands-on exercise

Creating a collection & documents using data explorer

Tasks:

1. **Create a collection with the database**
   - Choose ‘sharded collection’
   - Specify shard key as _id
   - Closely view and understand the ‘Advanced’ settings. Keep the default settings.

2. **Create document within the collection**
   - Click on ‘Documents’ within the collection.
   - Click on ‘New Document’ on the top menu to create new document
   - Create 2 new documents by pasting one entry at a time below in the document body

```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [ -72.622739, 42.070206 ], "pop" : 15338, "state" : "MA" }

{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [ -72.51564999999999, 42.377017 ], "pop" : 36963, "state" : "MA" }
```

Hands-on exercise

Enable global distribution & multi-region writes

Tasks:

1. **Add second region to replicate data to**
   - Go to ‘Replicate Data Globally’ blade for the Cosmos DB account
   - Add second region on the map
   - Click ‘Save’
2. **Enable multi-region writes**
   - In the same ‘Replicate Data Globally’ blade flip the switch to ‘Enable’ in the multi-region write section
   - Click ‘Save’

Hands-on exercise

Enable automatic failover

Tasks:

1. **Enable Automatic failover to second region**
   - Go to ‘Replicate Data Globally’ blade for the Cosmos DB account
     - Click on ‘Automatic Failover’
     - Flip the ‘Enable Automatic Failover’ switch to ‘On’
     - Click ‘Ok’

Implementation & development

API for MongoDB & features

Indexing

Query tuning

CRUD operations

Wire protocol versions & supported features

We implement the MongoDB wire protocol, which allows MongoDB clients to connect to Azure Cosmos DB as if it were a MongoDB replica set.

3 supported versions:

- Version 4.0
- Version 3.6
- Version 3.2

We maintain 4 replicas per region. The replica sets are managed automatically, you do not need to bother about managing the replica set or connecting to the right replica.

Continue to use existing drivers & clients

MongoDB Drivers

- C
- C++
- C#
- Go
- Java
- Node.js
- PHP
- Python
- Ruby
- Rust
- Scala
- Swift

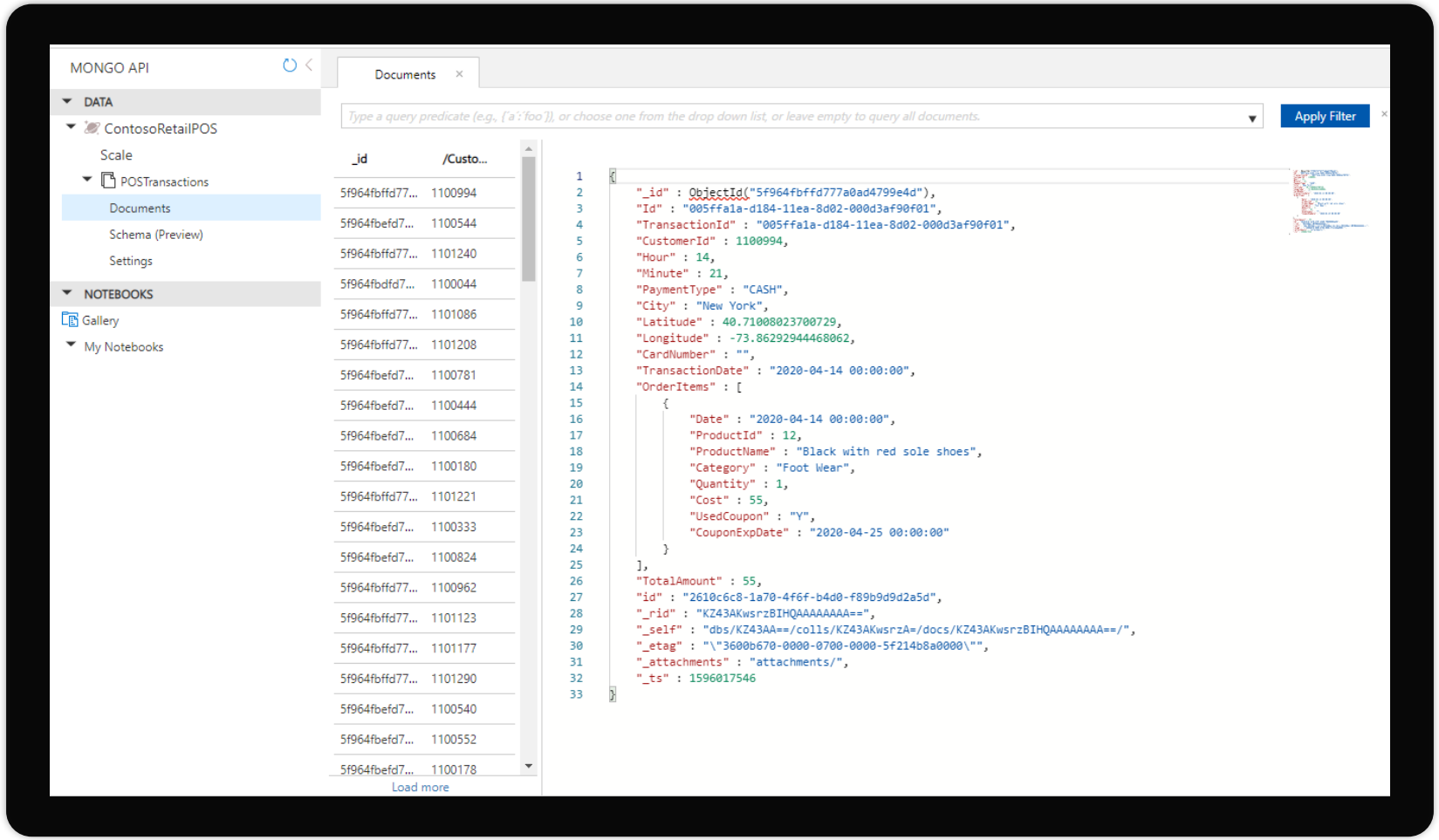
MongoDB Tools

- Monogo Shell
- Compass
- Studio 3T
- Mongoose
- Robo 3T

Improvements in API 4.0

- Support for multi-document ACID transactions within unsharded collections.
- New aggregation operators added
  - Conversion expressions such as $convert, $toBool, $toDate & others
  - More string expressions such as $trim, $ltrim, $rtrim
- Enhanced scan performance
  - 45% more performant queries
- Faster, more efficient BSON storage
  - New encoding format

Manage the database & collections from Data explorer in the portal



Zero downtime upgrades

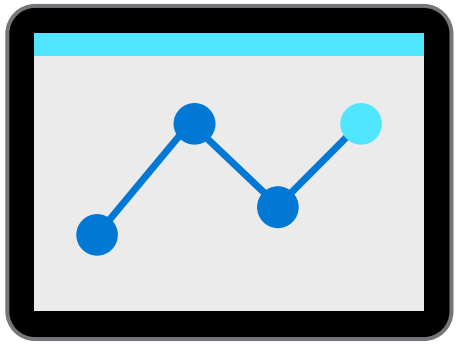
- All APIs in the same codebase
- Upgrade & rollback is easy – Flip of a switch
- Continued support for older API versions
- Endpoints:
  - *.mongo.cosmos.azure.com → For API for MongoDB > 3.6
  - *.documents.azure.com → For API for MongoDB 3.2
- v3.2 endpoint still available post upgrading to newer API versions

Developer experience & enterprise features

- Built-in Mongo Shell within the Azure Portal
- Deep integration with key Azure services used in cloud-native app development including
  - Azure Synapse Analytics
  - Azure Networking
  - Azure Monitoring
- Windows and Linux Emulator provides free local environment that emulates Azure Cosmos DB service for development purposes.

Indexing features & considerations

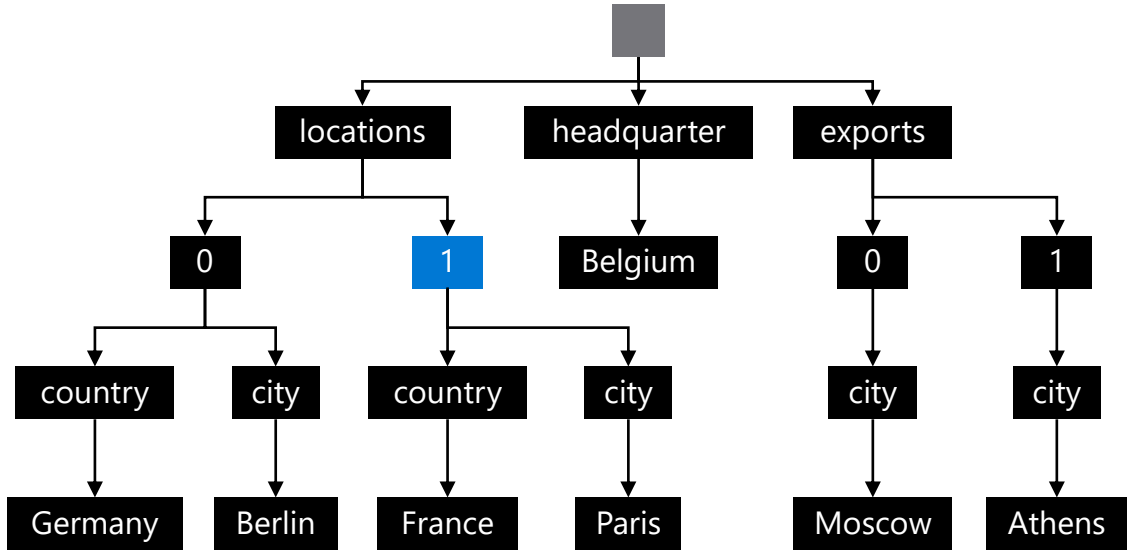
- Synchronous auto-indexing
  - _id field indexed by default
  - Wildcard indexes recommended in development stage
- Considerations
  - Index needed to sort on a field
  - Compound indexes cannot be created on nested properties or arrays
  - Unique indexes can only be created when a collection is empty



Item	Color	Microwave safe	Liquid capacity	CPU	Memory	Storage
Geek mug	Graphite	Yes	16ox	???	???	???
Coffee bean mug	Tan	No	12oz	???	???	???
Surface book	Gray	???	???	3.4 GHz Intel Skylake Core i7-6600U	16GB	1 TB SSD

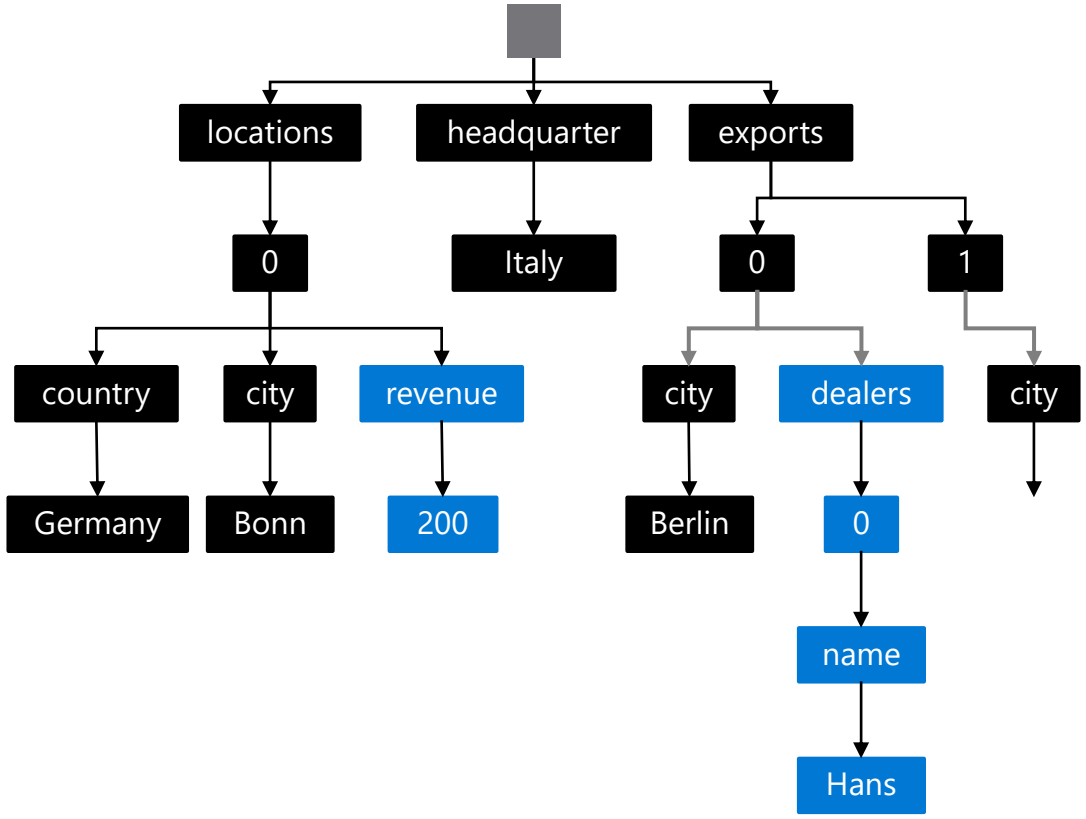
Indexing JSON documents

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

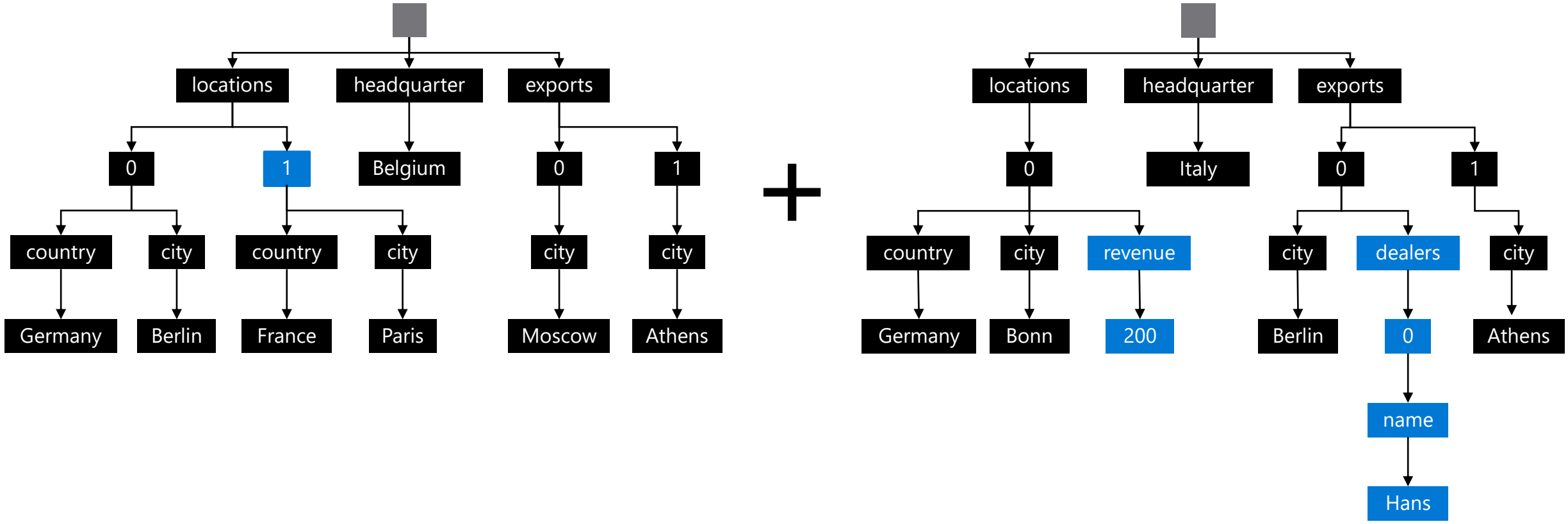


Indexing JSON documents continued

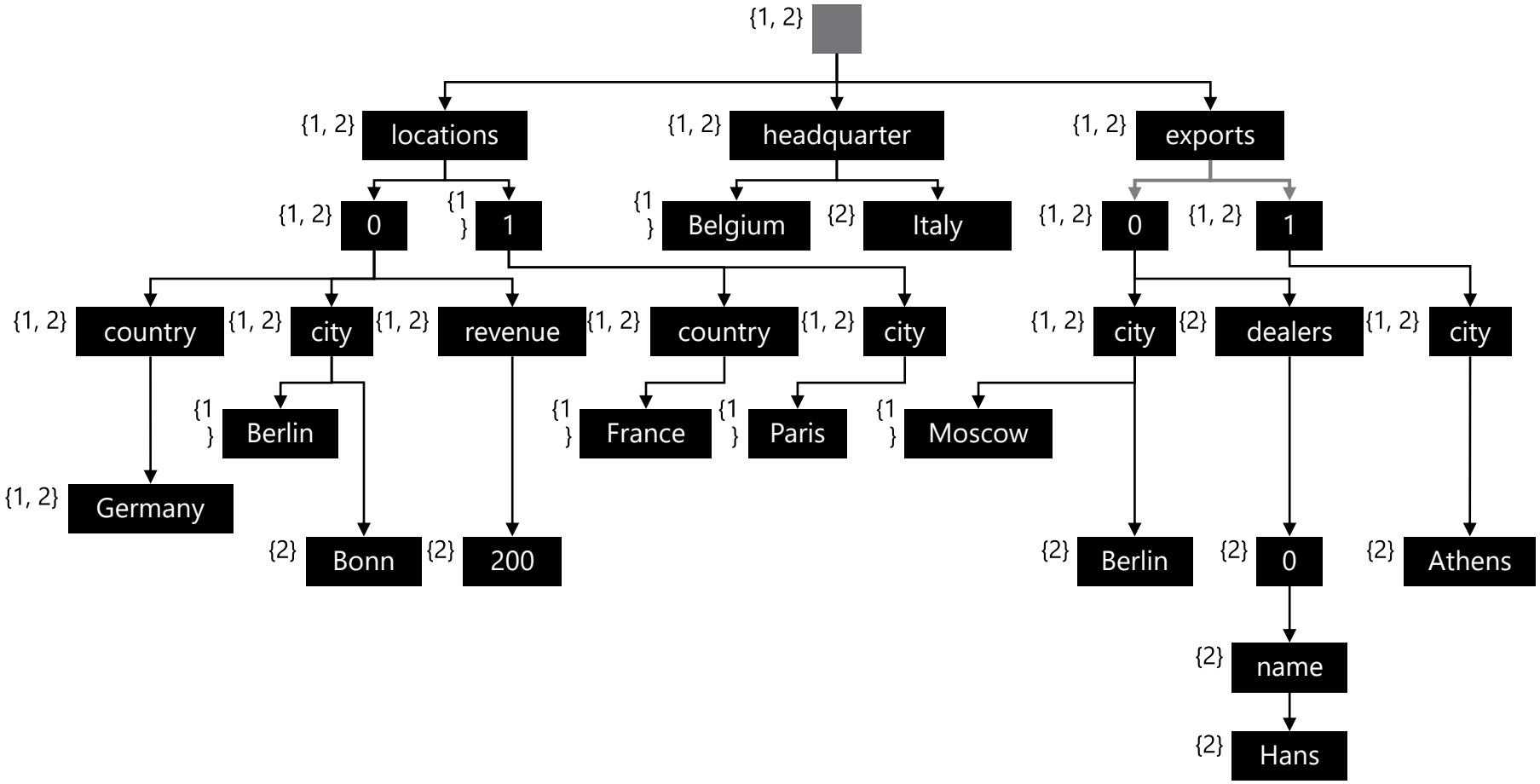
```
{
    "locations": [
        {
            "country": "Germany",
            "city": "Bonn",
            "revenue": 200
        }
    ],
    "headquarter": "Italy",
    "exports": [
        {
            "city": "Berlin",
            "dealers": [
                { "name": "Hans" }
            ]
        },
        { "city": "Athens" }
    ]
}
```



Indexing JSON documents continued



Inverted index



Index policies

Custom indexing policies

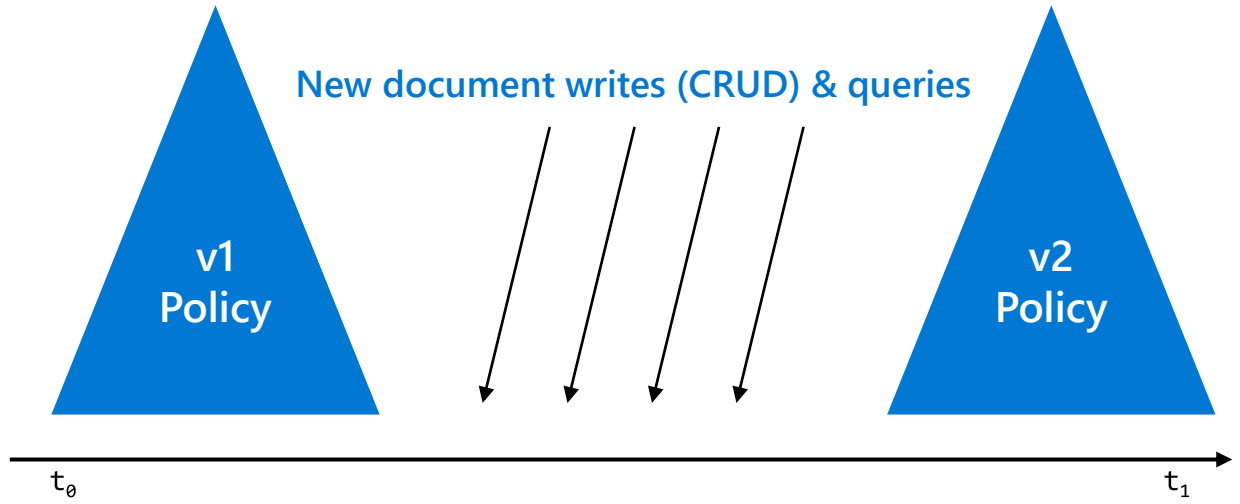
- Custom indexing policies allow you to design and customize the shape of your index while maintaining schema flexibility.
  - Define trade-offs between storage, write and query performance, and query consistency
  - Include or exclude documents and paths to and from the index
  - Configure various index types

```
{
    "automatic": true,
    "indexingMode": "Consistent",
    "includedPaths": [{
        "path": "/*",
        "indexes": [{
            "kind": "Hash",
            "dataType": "String",
            "precision": -1
        }, {
            "kind": "Range",
            "dataType": "Number",
            "precision": -1
        }, {
            "kind": "Spatial",
            "dataType": "Point"
        }]
    }],
    "excludedPaths": [{
        "path": "/nonIndexedContent/*"
    }]
}
```

Online index transformations

On-the-fly index changes

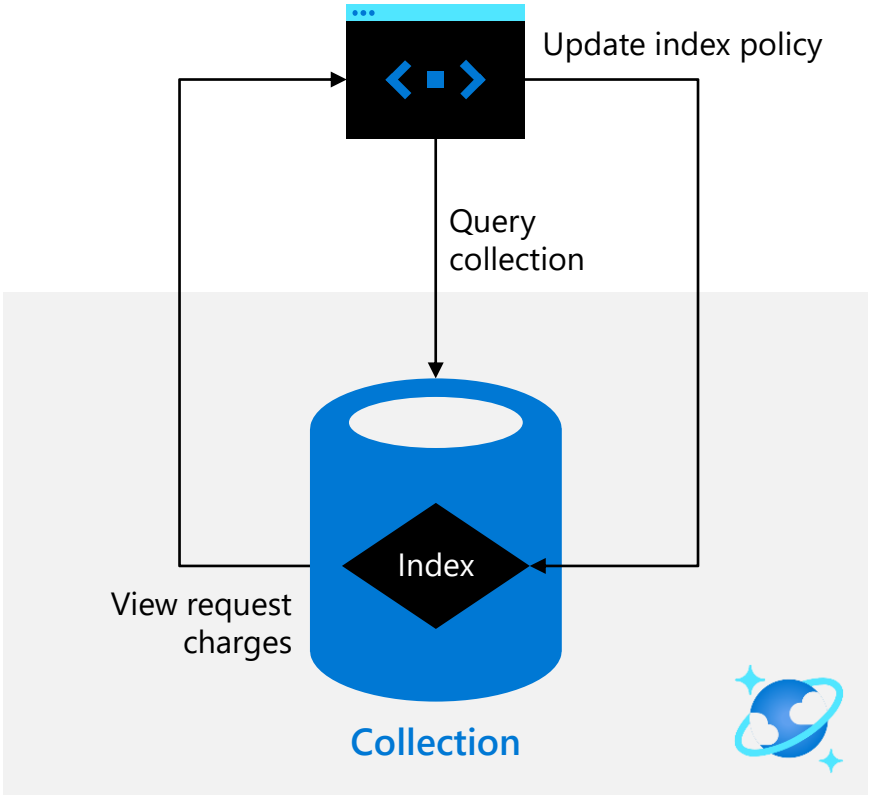
- In Azure Cosmos DB, you can make changes to the indexing policy of a collection on the fly. Changes can affect the shape of the index, including paths, precision values, and its consistency model.
- A change in indexing policy effectively requires a transformation of the old index into a new index.



Index tuning

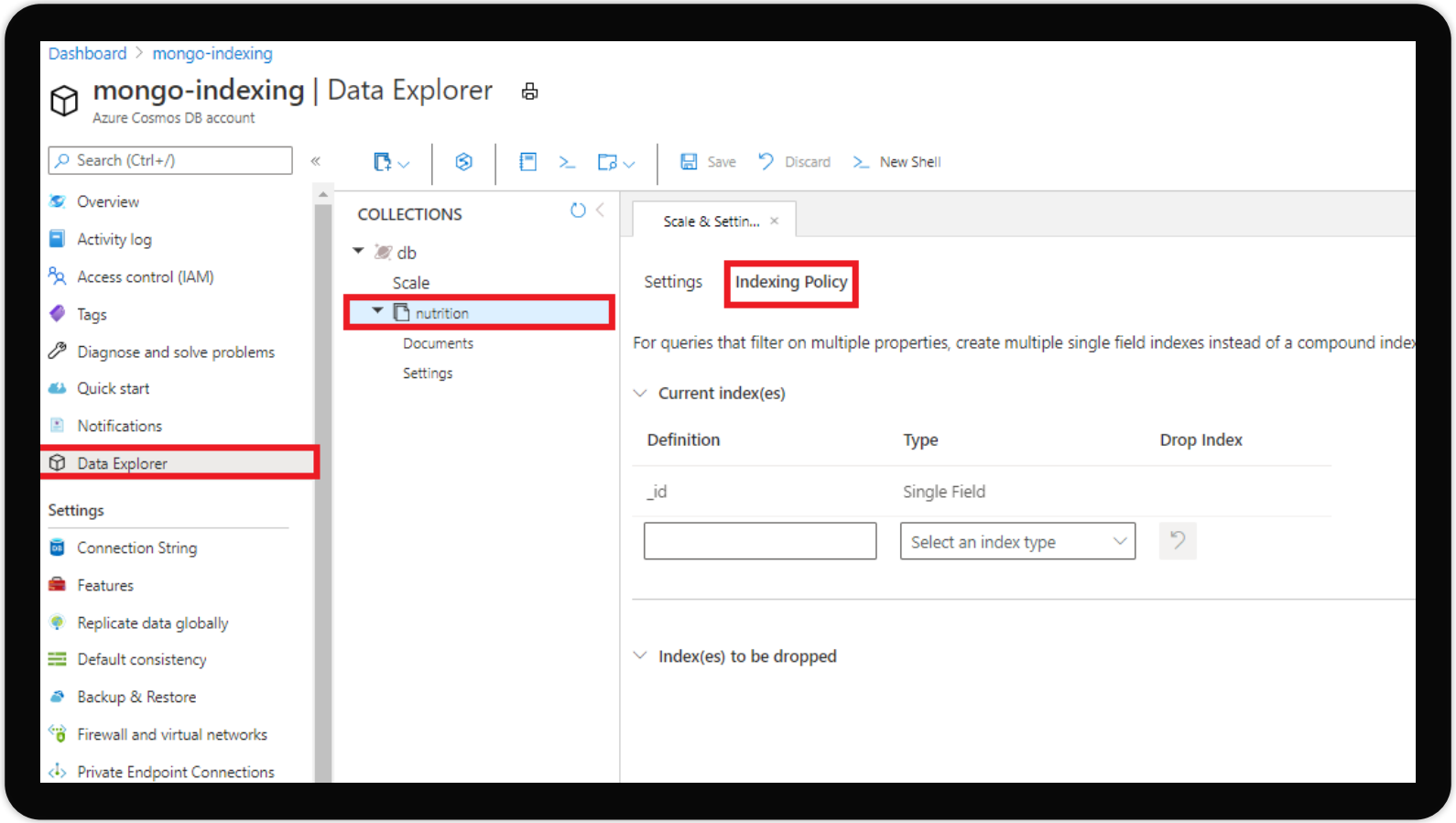
Metrics analysis

- Use getLastRequestStatistics command to compare various indexing policies, and for performance tuning.
- You can use this information to compare various indexing policies, and for performance tuning.



Managing index

From portal



Managing index continued

From shell

Get indexes

```
db.zips.getIndexes()
```

Single field Index

```
db.zips.createIndex({ "pop": 1})
```
compound index for 2 fields

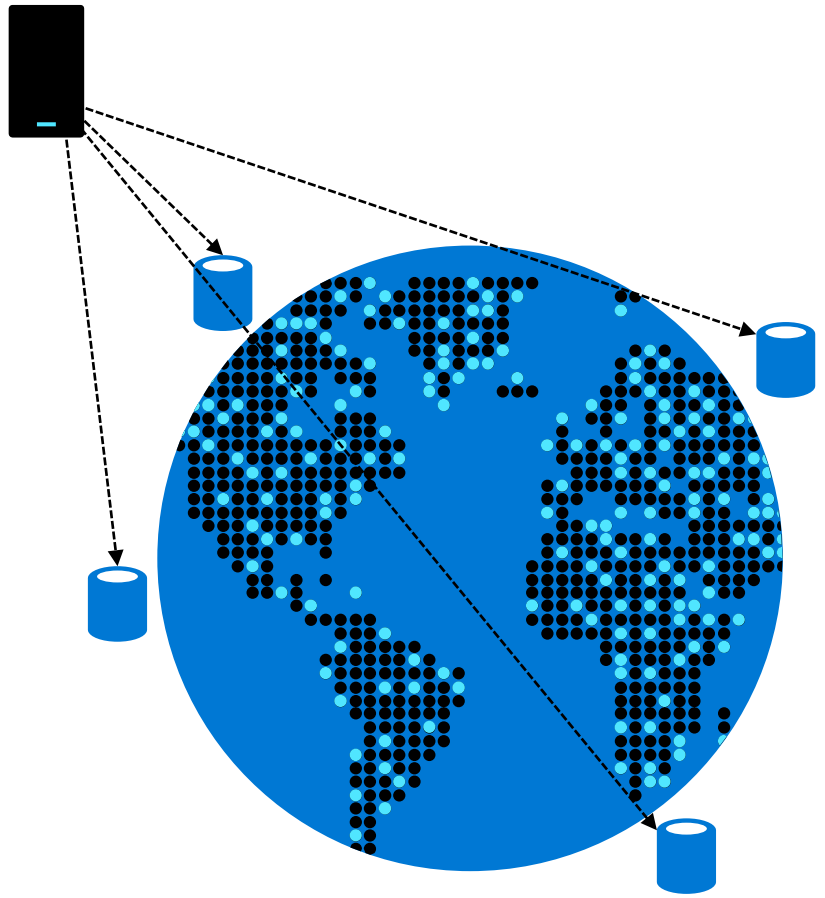
Compound index for 2 fields

```
db.zips.createIndex({ "pop": 1, "city": -1})
```

Mongo Shell

Querying

Tuning query techniques and parameters to make the most efficient use of a globally distributed database service.



Query tuning

Multiple things can impact the performance of a query running in Azure Cosmos DB

Provisioned throughput

- Measure RU per query, and ensure that you have the required provisioned throughput for your queries

Sharding and shard keys

- Favor queries with the shard key value in the filter clause for low latency

Network latency

- Account for network overhead in measurement, and use multi-homing APIs to read from the nearest region

Query tuning continued

Many things can impact the performance of a query running in Azure Cosmos DB.

Indexing policy

- Ensure that you have the required indexing paths/policy for the query

Query complexity

- Use simple queries to enable greater scale.

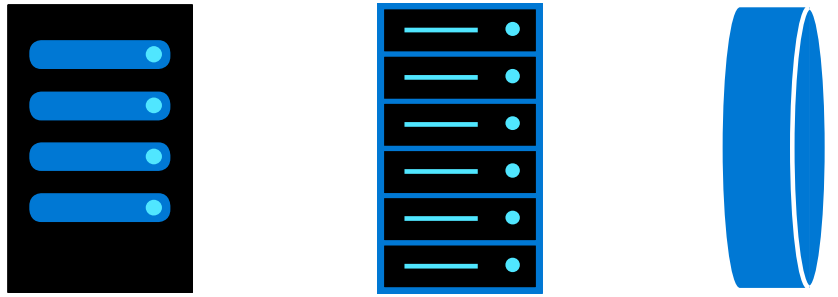
Query execution metrics

- Analyze the query execution metrics to identify potential rewrites of query and data shapes

Request units

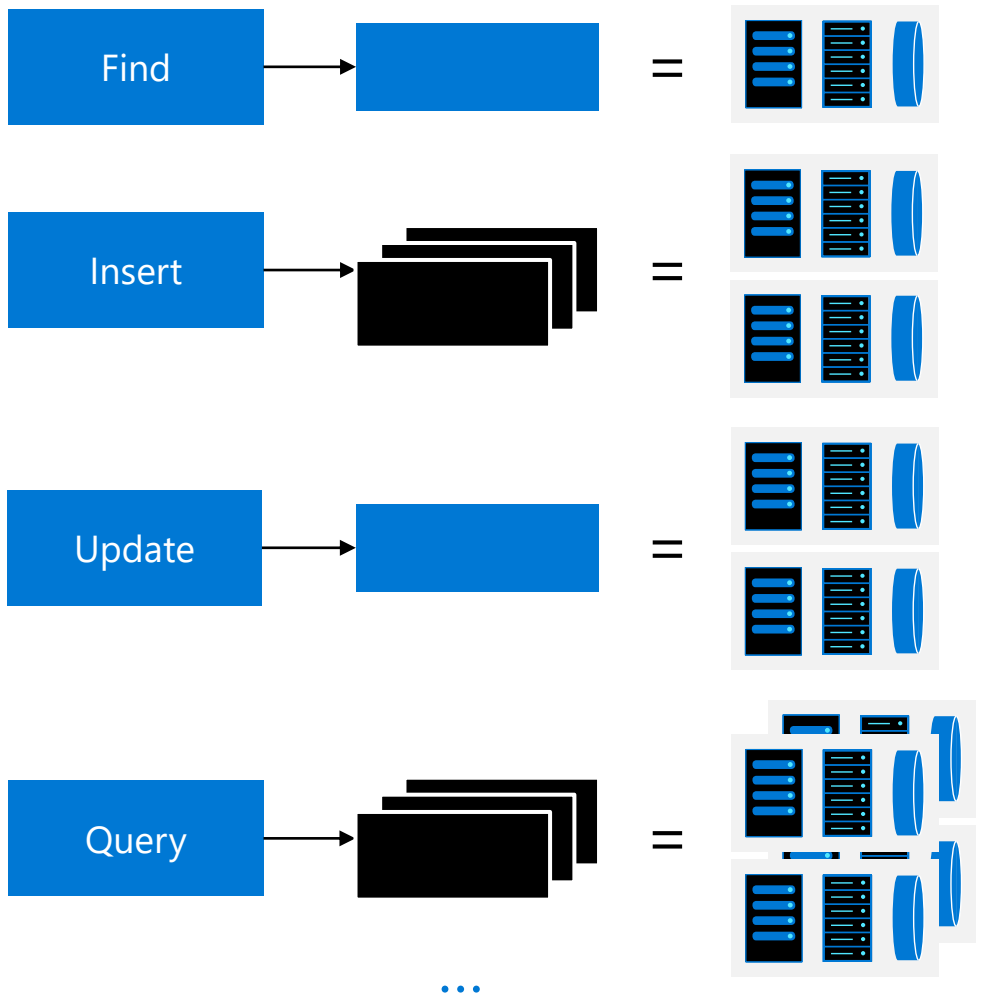
Request Units (RUs) is a rate-based currency

- Abstracts physical resources for performing requests
- Key to multi-tenancy, SLAs, and COGS efficiency
- Foreground and background activities



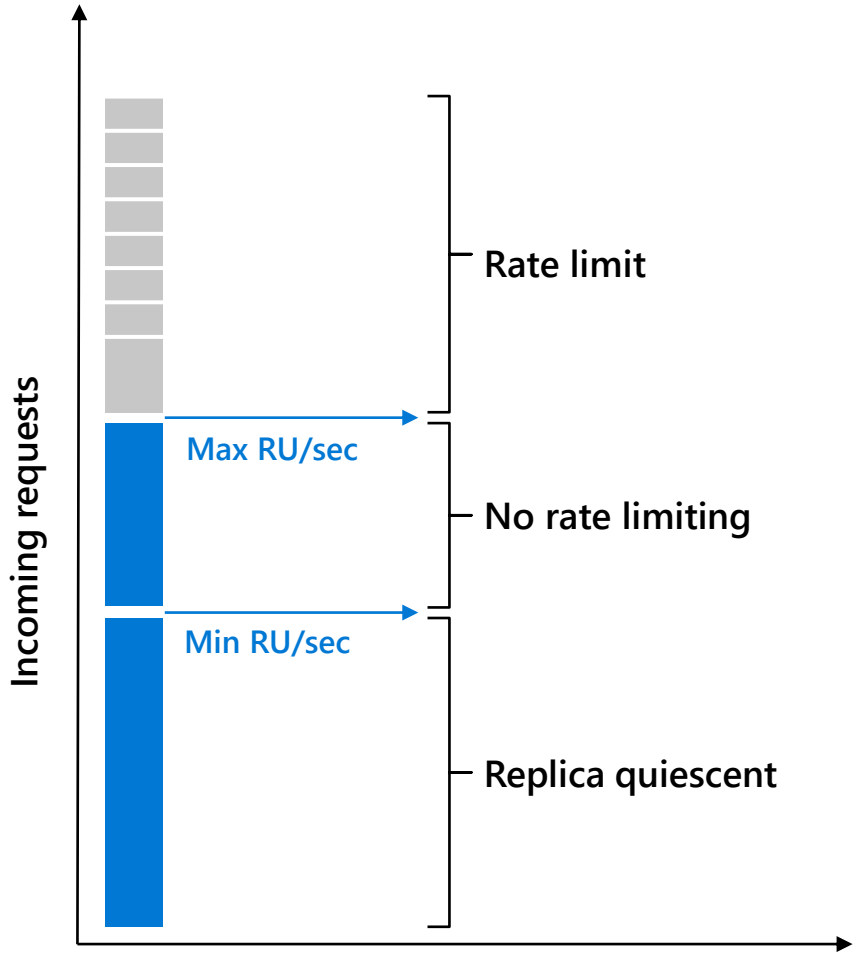
Request units continued

- Normalized across various operations
  - 1 RU = 1 read of 1 KB document
- Each request consumes fixed RUs
  - Applies to reads, writes, queries



Request units continued

- Provisioned in terms of RU/sec
  - Rate limiting based on amount of throughput provisioned
  - Can be increased or decreased instantaneously
- Metered hourly
  - Background processes like TTL expiration, index transformations scheduled when quiescent



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):

- Run `db.runCommand({getLastRequestStatistics: 1})` in Mongo Shell
- Check `getLastRequestStatistics` from Mongo driver
- Check query charges from portal data explorer

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 having an effect on 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.

Measuring RU charge continued

Stabilized logical charges

- Azure Cosmos DB uses information about past runs to produce a stable logical charge for the majority of CRUD or query operations.
- Since this stable charge exists, we can rely on our operations having a high degree of predictability with very little variation. We can use the predictable RU charges for future capacity planning.

Bulk of query RU charges is IO

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

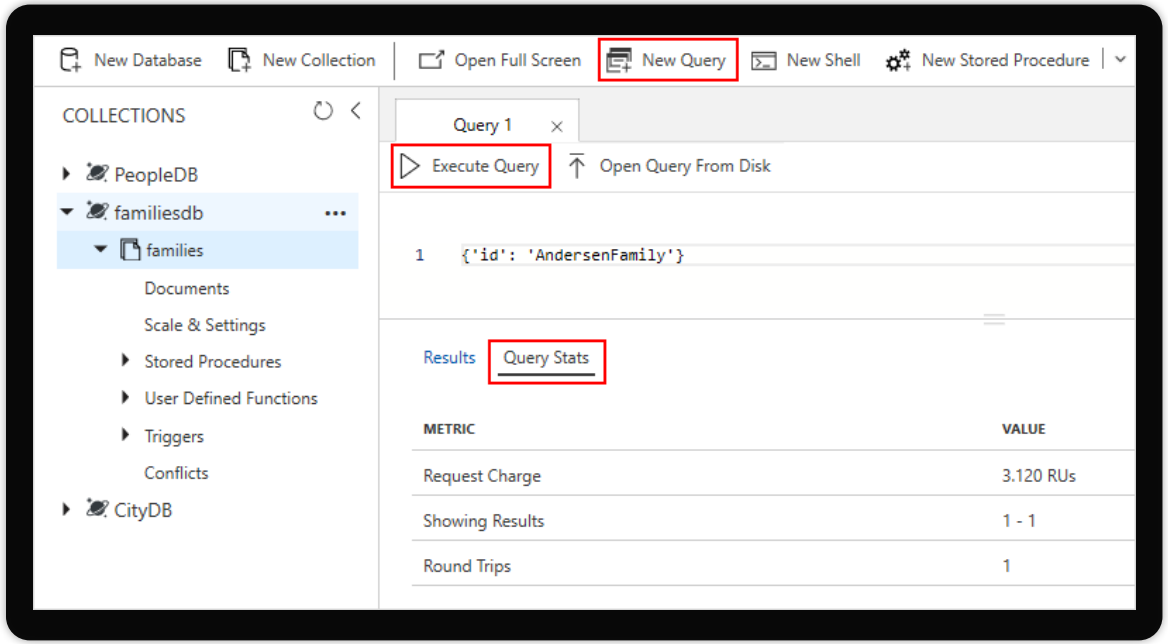
RU charge measurement example

From Shell

```
db.runCommand({getLastRequestStatistics: 1})

{
        "CommandName" : "find",
        "RequestCharge" : 400.58,
        "RequestDurationInMilliSeconds" : NumberLong(203),
        "EstimatedDelayFromRateLimitingInMilliseconds" :
NumberLong(0),
        "RetriedDueToRateLimiting" : false,
        "ActivityId" : "2b906ab6-73f1-46a0-b240-
781c73226144",
        "ok" : 1
}
```

From Portal



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/sec	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

[Total Monthly Cost]	=	[Monthly Cost for Storage]	+	[Monthly Cost for Throughput]
$79 per month	=	$25	+	$54

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

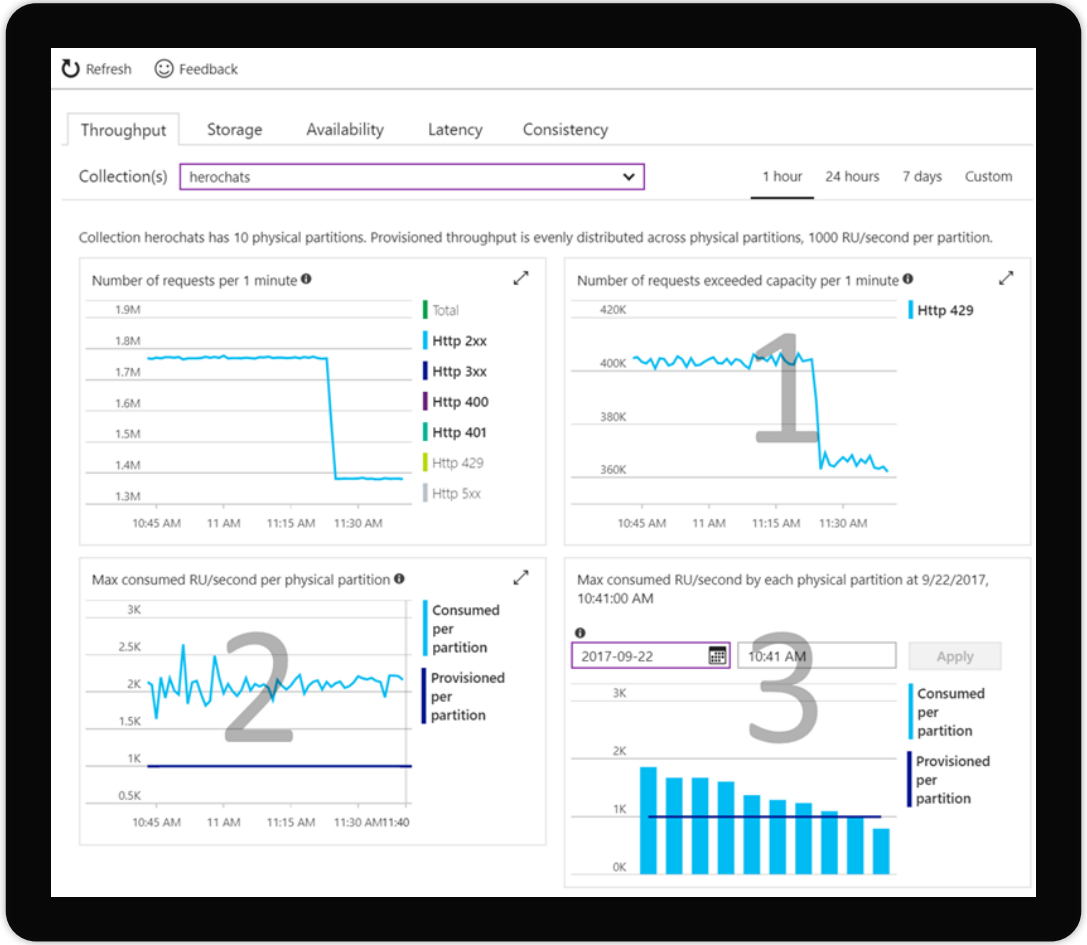
$explain

```
db.coll.find({foodGroup: "Baby Foods"}).explain({"executionStatistics": true })
{
    "stages" : [
        {
            "stage" : "$query",
            "timeInclusiveMS" : 905.2888,
            "timeExclusiveMS" : 905.2888,
            "in" : 362,
            "out" : 362,
            "details" : {
                "database" : "db-test",
                "collection" : "collection-test",
                "query" : {
                    "foodGroup" : {
                        "$eq" : "Baby Foods"
                    }
                },
                "pathsIndexed" : [],
                "pathsNotIndexed" : [
                    "foodGroup"
                ],
                "shardInformation" : [
                    {
                        "activityId" : "e68e6bdd-5e89-4ec5-b053-3dbbc2428140",
                        "shardKeyRangeId" : "0",
                        "durationMS" : 788.5867,
                        "preemptions" : 1,
                        "outputDocumentCount" : 362,
                        "retrievedDocumentCount" : 8618
                    }
                ],
            …
```

```
…….
"queryMetrics" : {
                    "retrievedDocumentCount" : 8618,
                    "retrievedDocumentSizeBytes" : 104963042,
                    "outputDocumentCount" : 362,
                    "outputDocumentSizeBytes" : 2553535,
                    "indexHitRatio" : 0.0016802042237178,
                    "totalQueryExecutionTimeMS" : 777.72,
                    "queryPreparationTimes" : {
                        "queryCompilationTimeMS" : 0.19,
                        "logicalPlanBuildTimeMS" : 0.14,
                        "physicalPlanBuildTimeMS" : 0.09,
                        "queryOptimizationTimeMS" : 0.03
                    },
                    "indexLookupTimeMS" : 0,
                    "documentLoadTimeMS" : 687.22,
                    "vmExecutionTimeMS" : 774.09,
                    "runtimeExecutionTimes" : {
                        "queryEngineExecutionTimeMS" : 37.45,
                        "systemFunctionExecutionTimeMS" : 10.82,
                        "userDefinedFunctionExecutionTimeMS" : 0
                    },
                    "documentWriteTimeMS" : 49.42
                }
            }
        }
    ],
    "estimatedDelayFromRateLimitingInMilliseconds" : 0.0,
    "continuation" : {
        "hasMore" : false
    },
    "ok" : 1.0
}
```

Validating throughput level choice

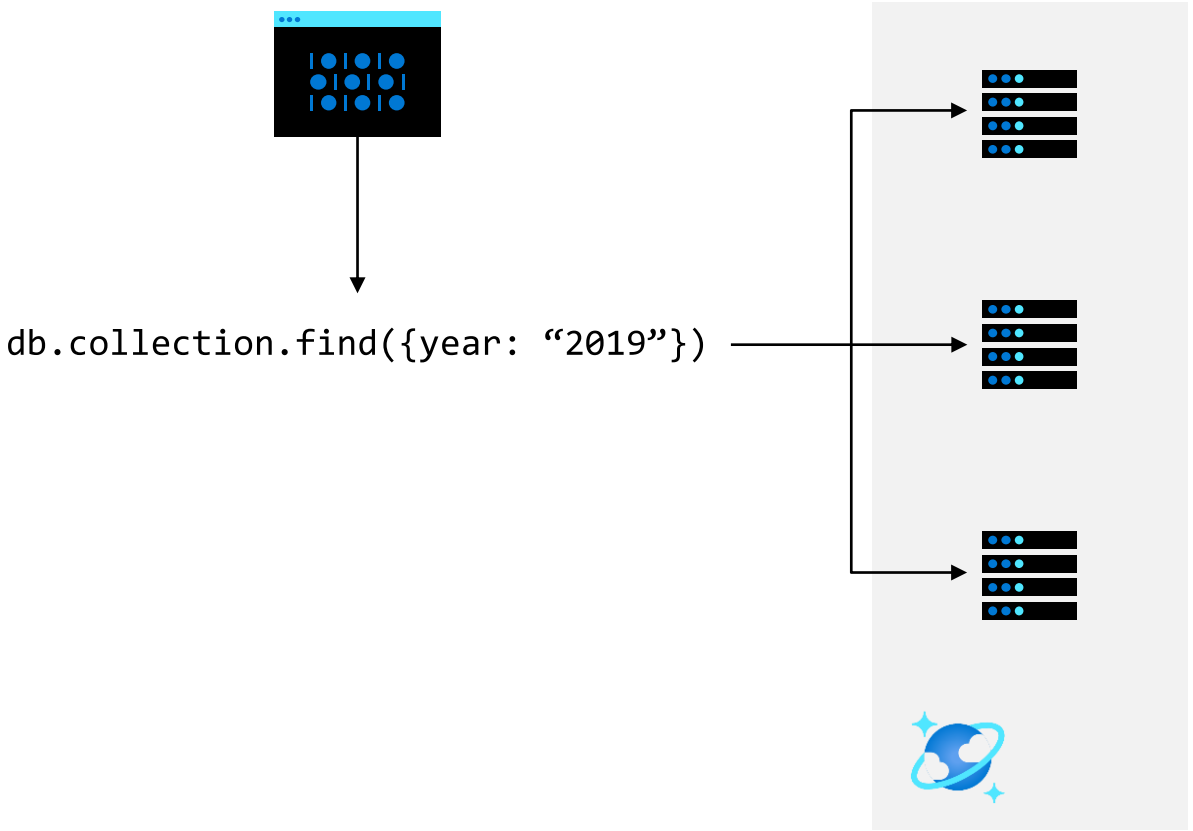
1. Check if your operations are getting rate limited.
   - Requests exceeding capacity chart
2. Check if consumed throughput exceeds the provisioned throughput on any of the physical shards
   - Max RU/second consumed per shard chart
3. Select the time where the maximum consumed throughput per shard exceeded provisioned on the chart
   - Max consumed throughput by each shard chart



Cross-shard aggregate

Low-latency aggregation works across multiple shards

- You can submit a simple Mongo query and Azure Cosmos DB handles the routing of the query among data shards and merges results to return the final aggregate values.



Hands-on exercise

CRUD operations

Tasks:

1. Run the commands in the next few slides against the imported sample data either using Mongo Shell from portal or Mongo Shell within Studio 3T

Working with database & collections

View database list

```
show dbs
```

Switch to a db or create a new one

```
use db_name
```

View collection list

```
show collections
```

Create a new collection

```
db.createCollection("test")
```

Delete a collection

```
db.test.drop()
```

Mongo Shell

Find

Find all docs where state is NY

```
db.zips.find({"state": "NY"})
```

Return one document that satisfies the specified query

```
db.zips.findOne()
```

Count the number of docs that satisfies the specified query

```
db.zips.find({"state": "NY"}).count()
```

Comparison operator: Find all documents where the tripduration was less than or equal to 70 seconds and the usertype was not Subscriber.

```
db.trips.find({ "tripduration": { "$lte" : 70 }, "usertype": { "$ne": "Subscriber" } })
```

Mongo Shell

Find continued

Sort: Sort all documents by population

```
db.zips.find().sort({ "pop": 1 })
```

Limit: Limit the number of documents displayed in the result

```
db.zips.find().sort({ "pop": -1 }).limit(10)
```

Sort over 2 fields: Sort over 2 fields using the compound index

```
db.zips.find().sort({ "pop": 1, "city": -1 })
```

Mongo Shell

Find continued

Logic operator: Finds all documents where airplanes CR2 or A81 left or landed in the KZN airport.

```
db.routes.find({ "$and": [ { "$or" :[ { "dst_airport": "KZN" },
                                      {"src_airport": "KZN" }] },
                           { "$or" :[ { "airplane": "CR2" }, { "airplane": "A81" }]}
                ]})
```

Projection: Find all documents and display their price and address.

```
db.listingsAndReviews.find({"price": 1, "address": 1})
```

Array operator $elemMatch: Find all documents where the student had a quiz score

```
db.grades.find({ "scores": { "$elemMatch": { "type": "quiz" } } })
```

Regex: Find all documents where the company had a CEO named Mark

```
db.companies.find({ "relationships.0.person.first_name": "Mark",
                    "relationships.0.title": { "$regex": "CEO" } }, { "name": 1 })
```

Mongo Shell

Insert

Insert the specified doc into the collection

```
db.zips.insert({"city" : "ALPINE",
                "zip" : "35014",
                "loc" : {"y" : 33.331165,
                         "x" : 86.208934    },
                "pop" : 3062.0,
                "state" : "AL"})
```

Insert with order set to 'false' - Attempts all inserts irrespective of failures.

```
db.inspections.insert([  { "_id": 1, "test": 1 },
                         { "_id": 1, "test": 2 },
                         { "_id": 3, "test": 3 }], { "ordered": false })
```

Mongo Shell

Update

Update a single document in the zips collection where the zip field is equal to "12534" by setting the value of the "pop" field to 17630

```
db.zips.updateOne({ "zip": "12534" }, { "$set": { "pop": 17630 } })
```

Update all documents in the zips collection where the city field is equal to "HUDSON" by adding 10 to the current value of the "pop" field.

```
db.zips.updateMany({ "city": "HUDSON" }, { "$inc": { "pop": 10 } })
```

Mongo Shell

upSERT

Updates the doc if exists, else inserts.

```
db.iot.updateOne({ "sensor": 5, "date": Date("2021-06-11"),
                       "valcount": { "$lt": 48 } },
                   { "$push": { "readings": { "v": 70, "t": "0000" } },
                       "$inc": { "valcount": 1, "total": 156 } },
                   { "upsert": true })
```

Mongo Shell

Delete

Delete one document that has test field equal to 3.

```
db.inspections.deleteOne({ "test": 3 })
```

Delete all the documents that have test field equal to 1.

```
db.inspections.deleteMany({ "test": 1 })
```

Mongo Shell

Administration

Troubleshooting & monitoring

Log analytics

Rate limiting errors

Azure Cosmos DB API for MongoDB operations may fail with rate-limiting (16500/429) errors if they exceed a collection's throughput limit (RUs).

- Enable Server Side Retry (SSR). Retries up to timeout of 60 seconds.
- Also include retry logic in the client code.

Troubleshooting tips

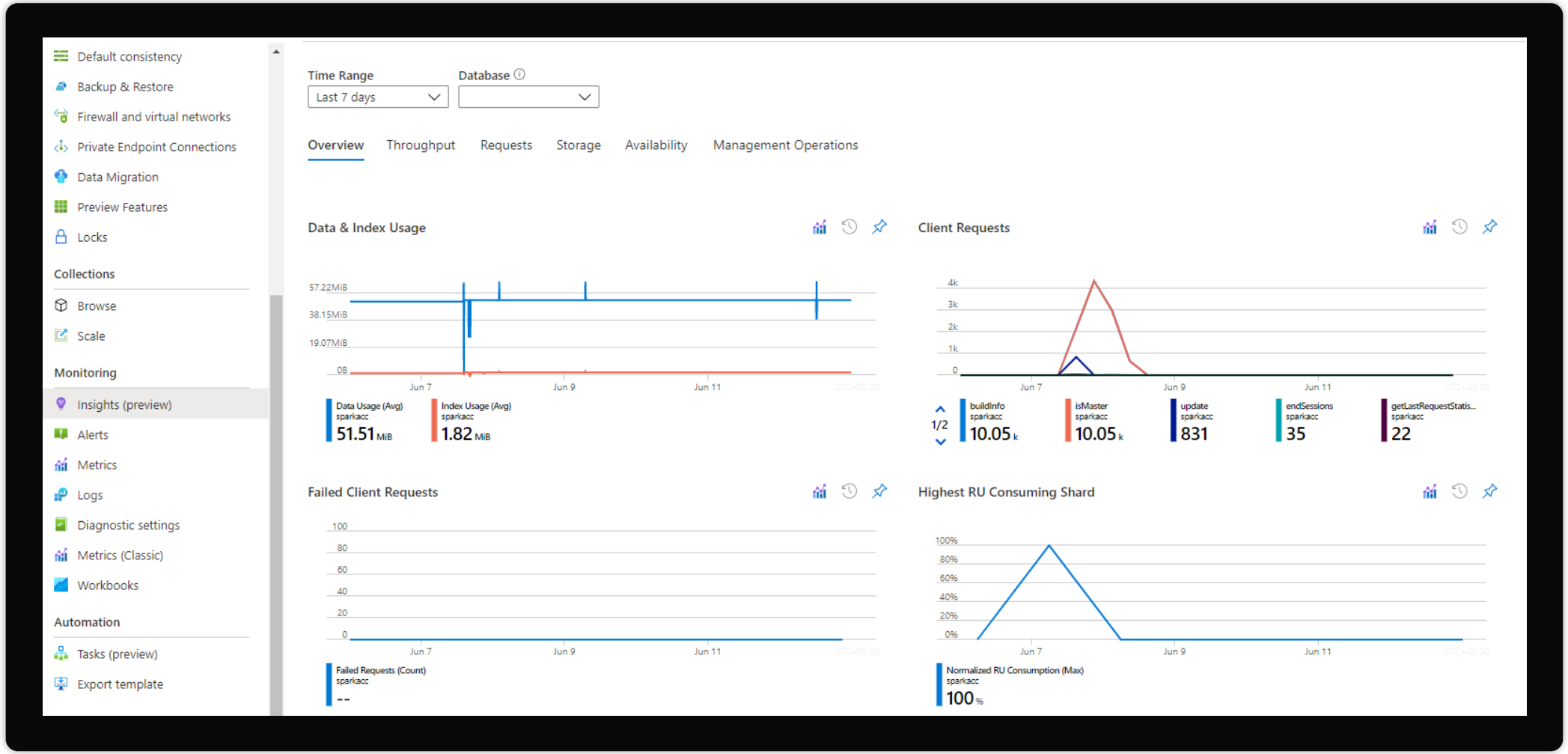
We support the wire protocol, so any driver can be used. It is recommended to use the latest driver version.

Common errors and solutions

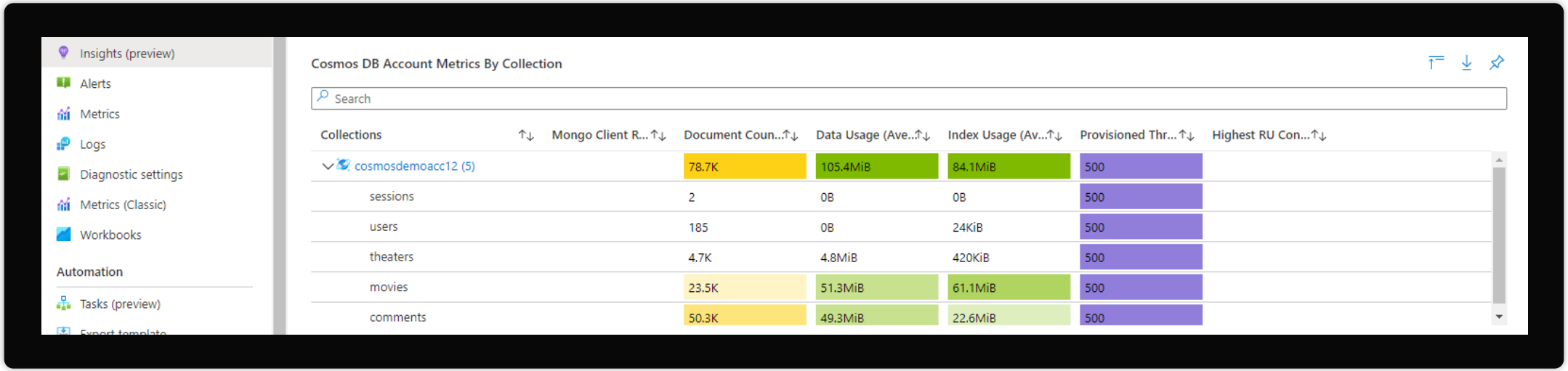
- https://docs.microsoft.com/en-us/azure/cosmos-db/mongodb-troubleshoot

In case of errors, collect repro details, client driver logging, sample document, etc.

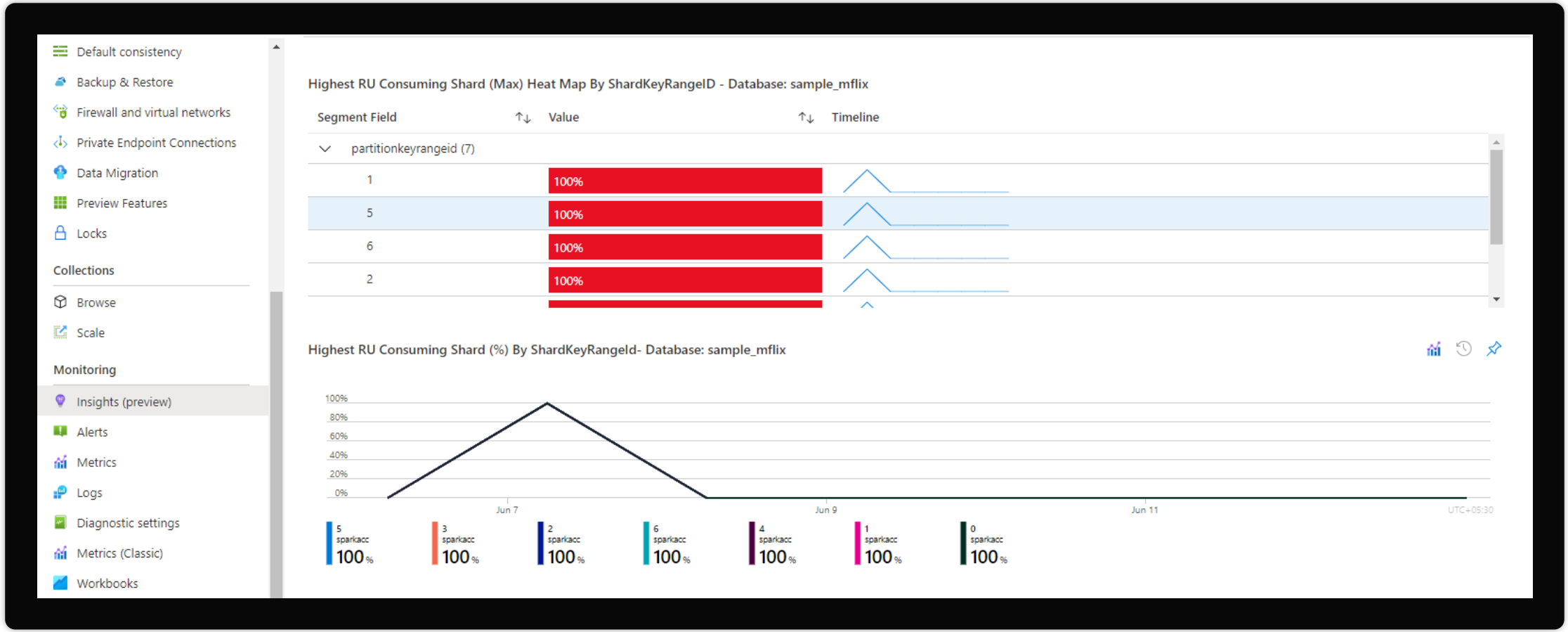
Insights (preview)



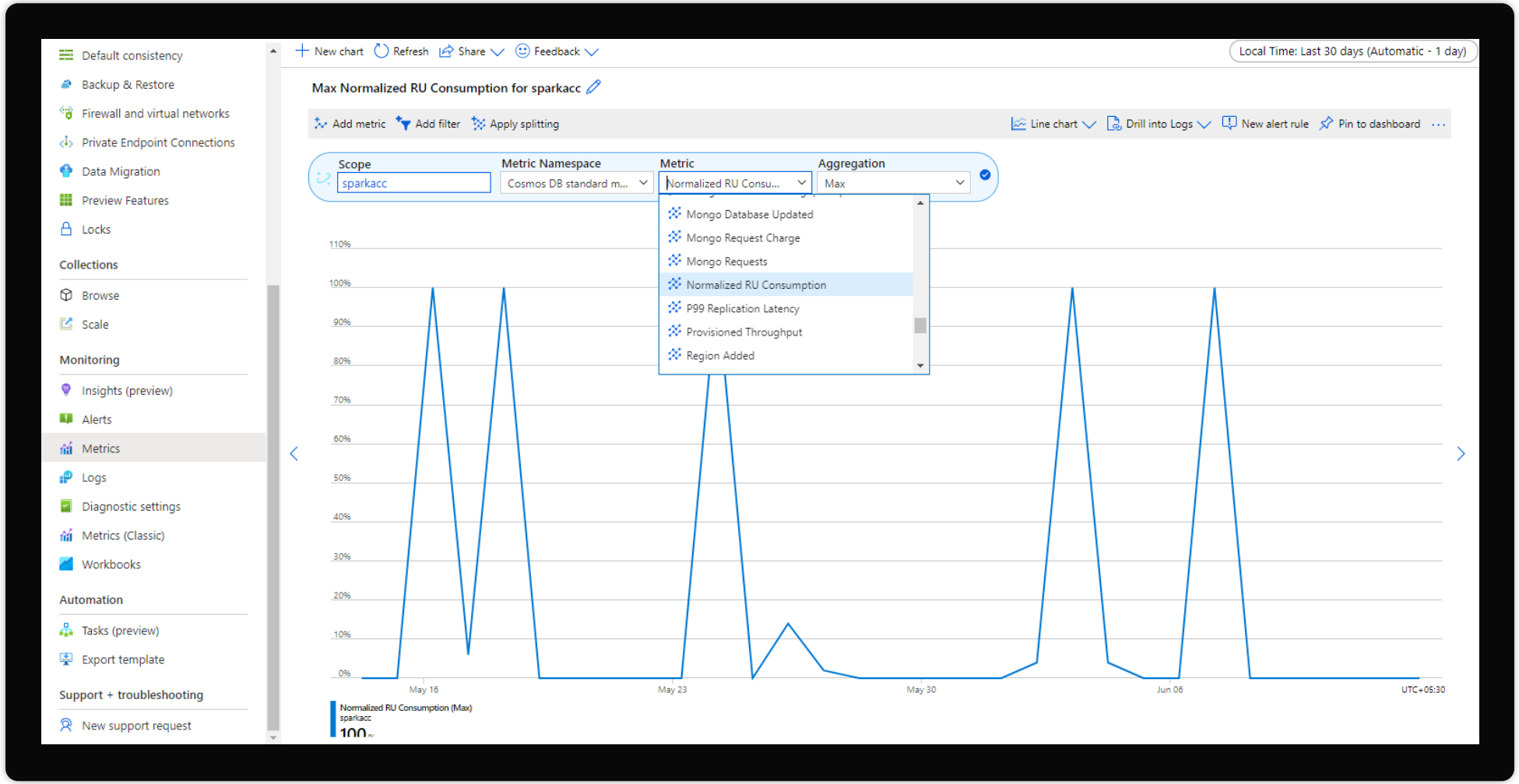
Insights (preview) continued



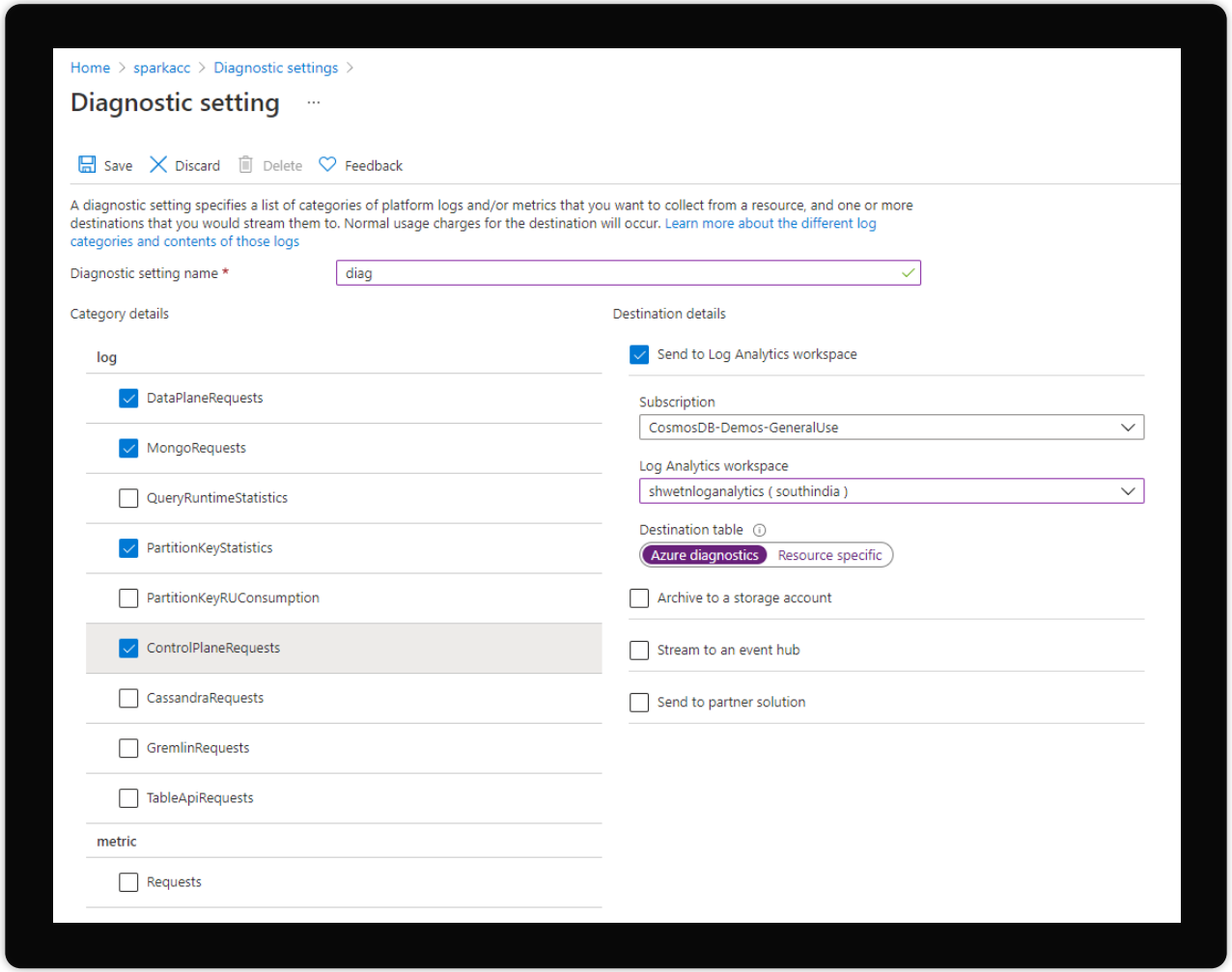
Insights (preview) continued



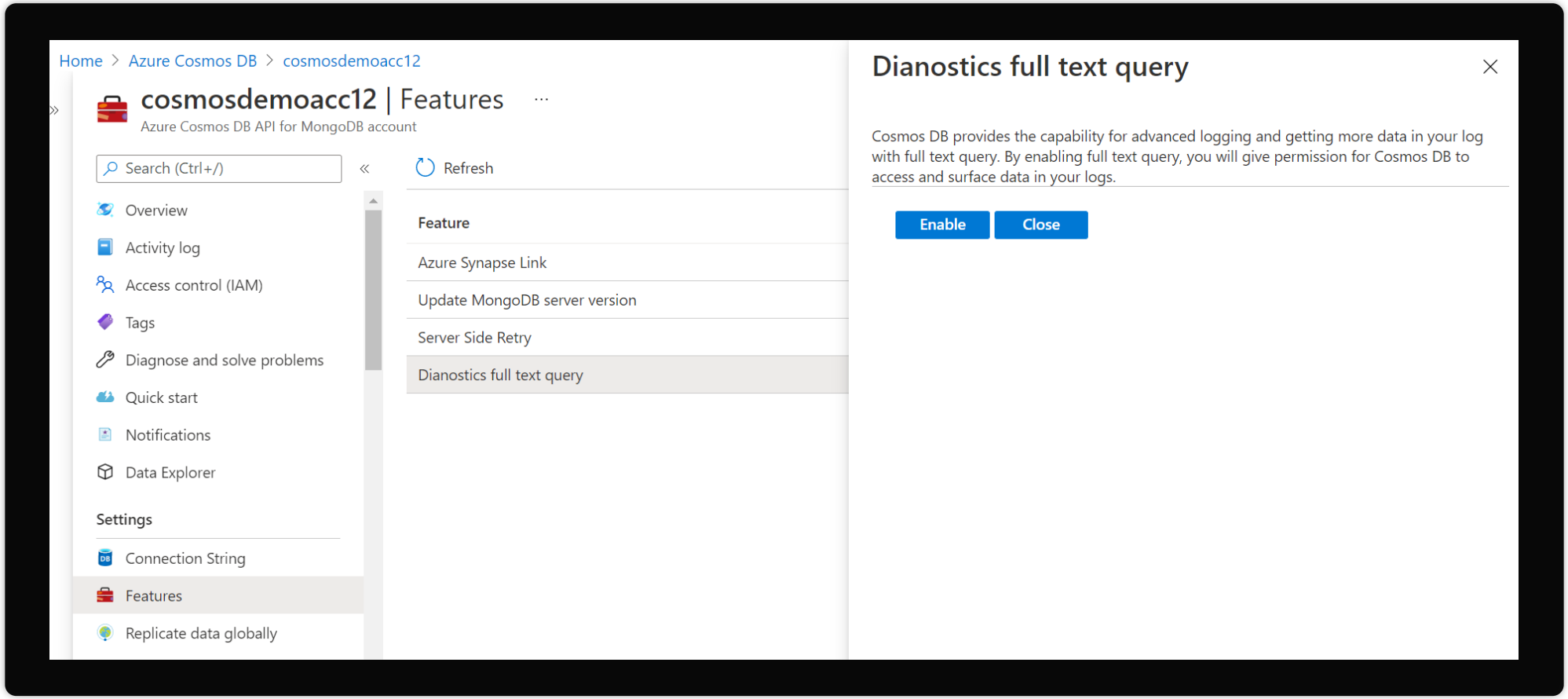
Metrics


Default consistency
Backup & Restore
Firewall and virtual networks
Private Endpoint Connections
Data Migration
Preview Features
Locks
Collections
Browse
Scale
Monitoring
Insights (preview)
Alerts
Metrics
Logs
Diagnostic settings
Metrics (Classic)
Workbooks
Automation
Tasks (preview)
Export template
Support + troubleshooting
New support request
New chart
Refresh
Share
Feedback
Local Time: Last 30 days (Automatic - 1 day)
Max Normalized RU Consumption for sparkacc
Add metric
Add filter
Apply splitting
Line chart
Drill into Logs
New alert rule
Pin to dashboard
Scope
sparkacc
Metric Namespace
Cosmos DB standard m...
Metric
Normalized RU Consu...
Aggregation
Max
Mongo Database Updated
Mongo Request Charge
Mongo Requests
Normalized RU Consumption
P99 Replication Latency
Provisioned Throughput
Region Added
110%
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
May 16
May 23
May 30
Jun 06
UTC+05:30
Normalized RU Consumption (Max)
sparkacc
100

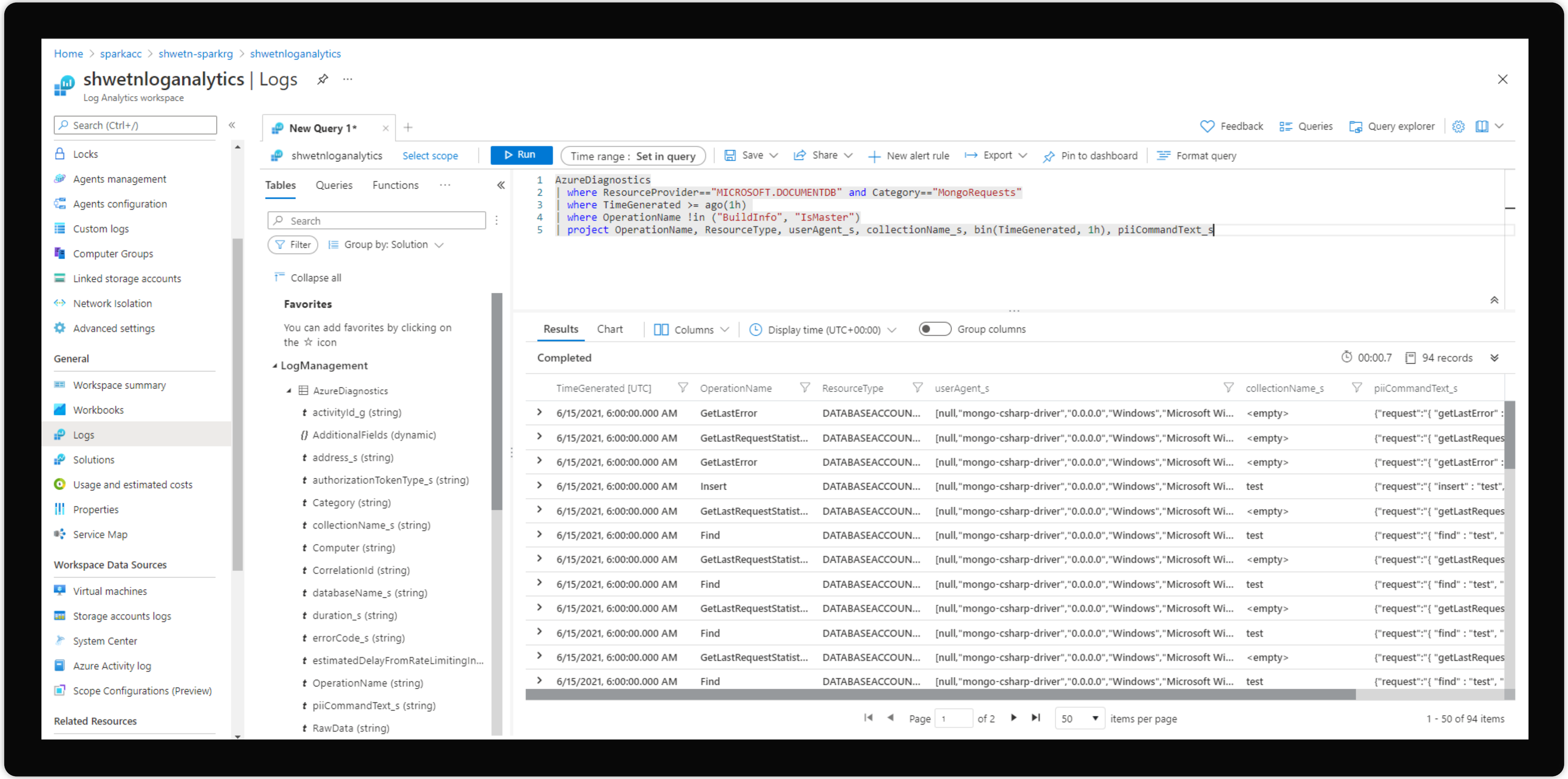
Enable diagnostic setting from portal



Enable full text query



Viewing the logs



Enregistrez vous dès maintenant au prochain Webinars Data AI

Event Webinar (Les jeudis de la Data & AI) - L200/300	Date	Duration (min)	Link
Azure Synapse	22/09/2022	120	https://msevents.microsoft.com/event?id=857781749
Les solutions SQL dans Azure (PaaS, IaaS, SaaS)	29/09/2022	120	https://msevents.microsoft.com/event?id=502366997
Déploiement et sécurisation des workspaces Azure Machine learning	06/10/2022	120	https://msevents.microsoft.com/event?id=1505714138
Azure Scale Analytics - Architectures Data Mesh dans Azure avec Azure Synapse, Microsoft Purview et Azure Data Share	13/10/2022	120	https://msevents.microsoft.com/event?id=139685175
MLOps avec Azure Machine Learning	20/10/2022	120	https://msevents.microsoft.com/event?id=1245885767
SQL Server 2022 et hybridation native avec Azure SQL Managed Instance	10/11/2022	120	https://msevents.microsoft.com/event?id=145826476
Machine Learning dans Azure Synapse Analytics	17/11/2022	120	https://msevents.microsoft.com/event?id=3637723312
Azure Cosmos DB et IA	24/11/2022	120	https://msevents.microsoft.com/event?id=2646013445
Azure et les Services Cognitifs	08/12/2022	120	https://msevents.microsoft.com/event?id=3772037220
La gouvernance de données dans Azure avec Microsoft Purview	15/12/2022	120	https://msevents.microsoft.com/event?id=1499560981
MLOps avec Azure Machine Learning	12/01/2023	120	https://msevents.microsoft.com/event?id=4115194515
Data processing dans Azure ave Azure Synapse, Azure Batch, Spark, Notebook, etc.	19/01/2023	120	https://msevents.microsoft.com/event?id=1537241181
Déploiement et sécurisation des workspace Azure Synapse	26/01/2023	120	https://msevents.microsoft.com/event?id=1806467748
Azure Machine Learning pour les Citizen Data Scientists	09/02/2023	120	En cours
PowerBI - Self Service Analytics	16/02/2023	120	https://msevents.microsoft.com/event?id=1401519679
L'IA responsable avec Azure machine learning	09/03/2023	120	https://msevents.microsoft.com/event?id=2072953112
Machine Learning dans Azure Synapse Analytics	16/03/2023	120	https://msevents.microsoft.com/event?id=3413014857
Les bases de données Open Source dans le cloud Azure	23/03/2023	120	https://msevents.microsoft.com/event?id=2727487131
Hybridation des services de Machine Learning Azure	06/04/2023	120	https://msevents.microsoft.com/event?id=1624914222
La gouvernance de données dans Azure avec Microsoft Purview	13/04/2023	120	https://msevents.microsoft.com/event?id=3909342839
Les solutions SQL dans Azure (PaaS, IaaS, SaaS)	04/05/2023	120	https://msevents.microsoft.com/event?id=1162207895
Data processing dans Azure ave Azure Synapse, Azure Batch, Spark, Notebook, etc.	16/05/2023	120	https://msevents.microsoft.com/event?id=3517068442
Hybridation des services de données Azure	24/05/2023	120	https://msevents.microsoft.com/event?id=2996507398
Self Service Analytics	01/06/2023	120	En cours

END