Azure Cosmos DB – Overview

Microsoft highly scalable, NoSQL database platform in Azure

Contents

[Overview 2](#_Toc29890522)

[Features of Cosmos DB: 2](#_Toc29890523)

[Horizontally Scalable: 2](#_Toc29890524)

[Globally Distributed: 2](#_Toc29890525)

[Schema-Agnostic Indexing: 3](#_Toc29890526)

[Multi model: 3](#_Toc29890527)

[Multi Consistency Support: 3](#_Toc29890528)

[Multi-Master Support: 3](#_Toc29890529)

[Multi API and Multi Language support: 3](#_Toc29890530)

[Multi Model Capabilities: 3](#_Toc29890531)

[Document Data Model: 4](#_Toc29890532)

[Columnar Data Model: 4](#_Toc29890533)

[Key-Value Pair Data Model (Table API): 4](#_Toc29890534)

[Graph Data Model: 4](#_Toc29890535)

[Measurement Unit: 5](#_Toc29890536)

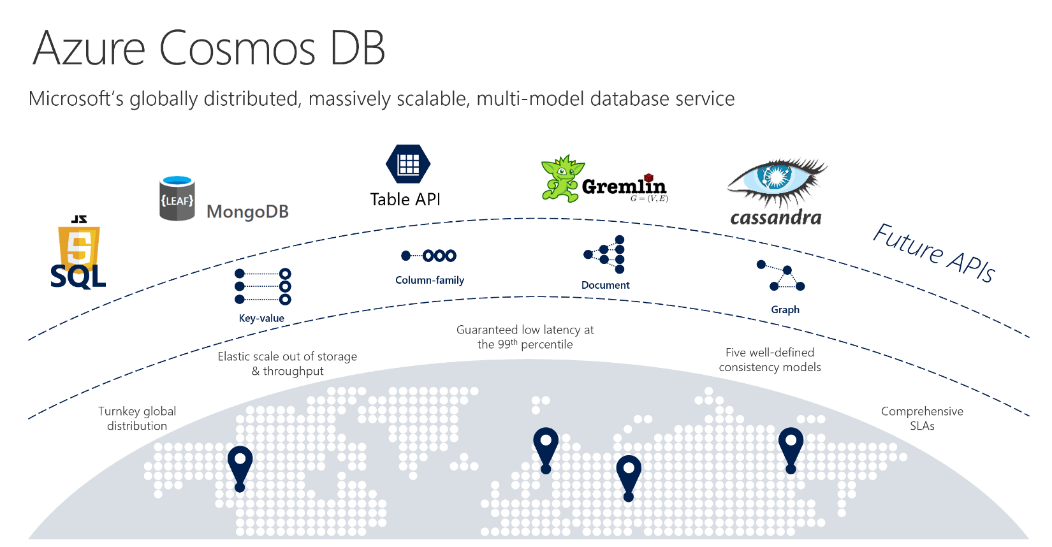
[Azure Cosmos DB Structure: 5](#_Toc29890537)

[Use Cases: 6](#_Toc29890538)

Overview:

Cosmos Database (DB) is a ***horizontally scalable, globally distributed, fully managed, low latency, multi-model, multi query-API database*** for managing data at large scale. Cosmos DB is a PaaS (Platform as a Service) offering from Microsoft Azure and is a cloud-based NoSQL database.

Since Azure Cosmos DB is available in all regions, the data can be distributed to any number of Azure regions. (i.e.) the data can be replicated to nearest region from where the users are accessing. So that it will help in serving data much faster to users with low latency.



# Features of Cosmos DB:

### Horizontally Scalable:

* Ability to handle the increased load by adding more servers to the cluster
* Can be scaled horizontally to support hundreds of millions of transactions per second for read and writes.

### Globally Distributed:

* Data can be replicated globally by adding more regions with just one click

### Schema-Agnostic Indexing:

* Enables auto indexing of the data
* Automatically indexes all the data without requiring schema and index management

### Multi model:

* Used for storing data in *Key-value Pair, Document-based, Graph-based & Column Family-based databases*
* Irrespective of the chosen data model, Global distribution, horizontal partitioning, provisioning throughput and auto indexing capabilities are same

### Multi Consistency Support:

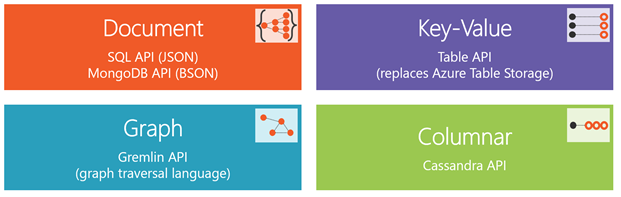
* Supports 5 consistency levels i.e., *Eventual, Prefix, Session, Bounded and Strong*

### Multi-Master Support:

* Supports multi master which means writes and reads can be scaled elastically across any number of azure regions across the world
* With this feature, all the data servers can act as write servers

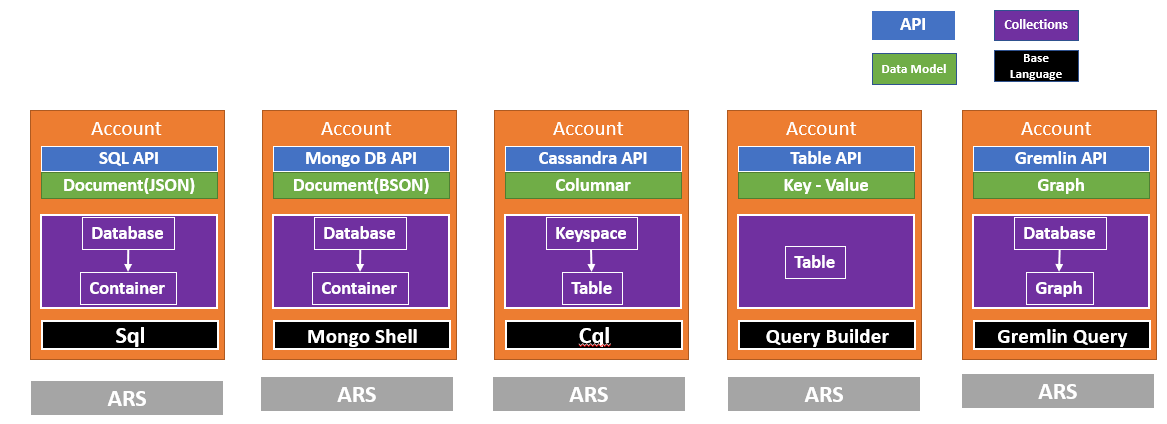
### Multi API and Multi Language support:

* Supports multiple APIs (SQL API, Mongo DB API, Cosmos DB Table API, Gremlin API and Cassandra API
* Supports multiple programming languages that includes Java, Python, .Net, Java script and Node.js



# Multi Model Capabilities:

Cosmos DB supports the following data models: Key-Value, Column-Family, Document, and Graph database models. Regardless of which data model we use, core content model of the Cosmos DB engine is based on ARS (Atom-Record-Sequence, which defines persistent layer for key-value pairs) and projects different data models in different APIs. Cosmos DB exposes the data in JSON format.



## Document Data Model:

This model supports SQL API and MongoDB APIs; both of these APIs give the document data model. These two APIs are different, though they are similar in data modeling. SQL API allows building transactional stored procedures, triggers, and user-defined functions. SQL API stores entities in JSON in a hierarchical key-value document. MongoDB API stores in BSON (Binary encoded version of JSON, which extends JSON with additional data types and multi-language support).

SQL API works with Document DB protocols, whereas MongoDB API works with MongoDB APIs. Both these APIs allow interacting with the documents in the database. If you already have a MongoDB solution and you want to make it scale out or globally aware, you can switch to Cosmos DB and use MongoDB API for interacting with the documents.

## Columnar Data Model:

This data model supports Cassandra API. Existing Cassandra implementations can be easily and quickly moved to Cosmos DB and use column family format, which is used in Cassandra. Columnar data model is similar to key-value data model except that the items in the data model adhere to the defined schema.

## Key-Value Pair Data Model (Table API):

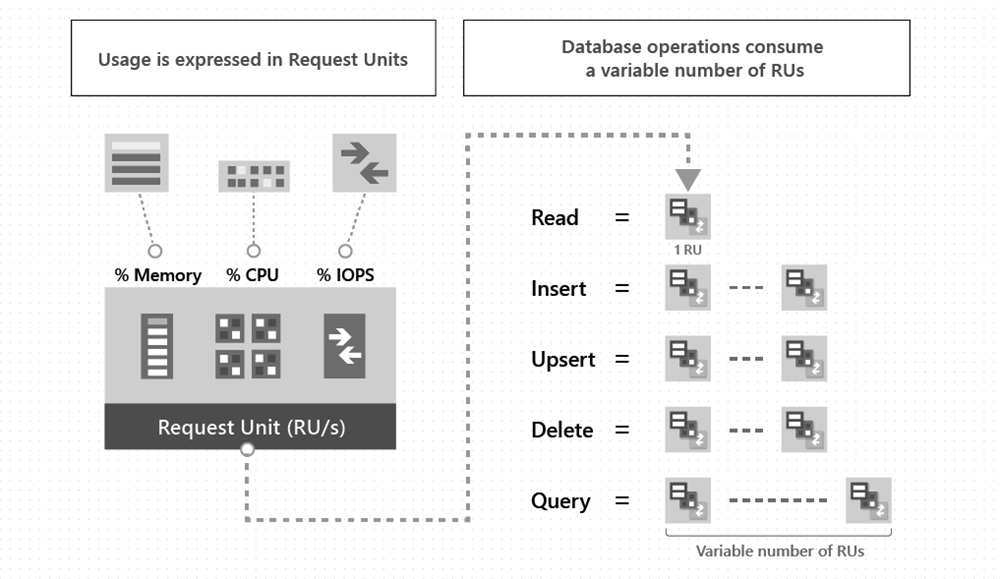
In this data model, each entity consists of a key and a value pair. The value itself can be a set of key-value pairs. This is very similar to a table in a relational database where each row has the same set of columns. The Key-Value pair solution is supported on standard Azure table API. This Table API is primarily helpful for existing Azure Table Storage customers in migration to Cosmos DB.

## Graph Data Model:

Graph database implements a collection of interconnected entities and relationships. Microsoft has chosen to use Gremlin API from Apache Tinkerpop open source project. Gremlin API allows you to interact with a Graph database globally scaled and provides a graph traversal language, which enables to efficiently query across many relationships exist in a graph database.

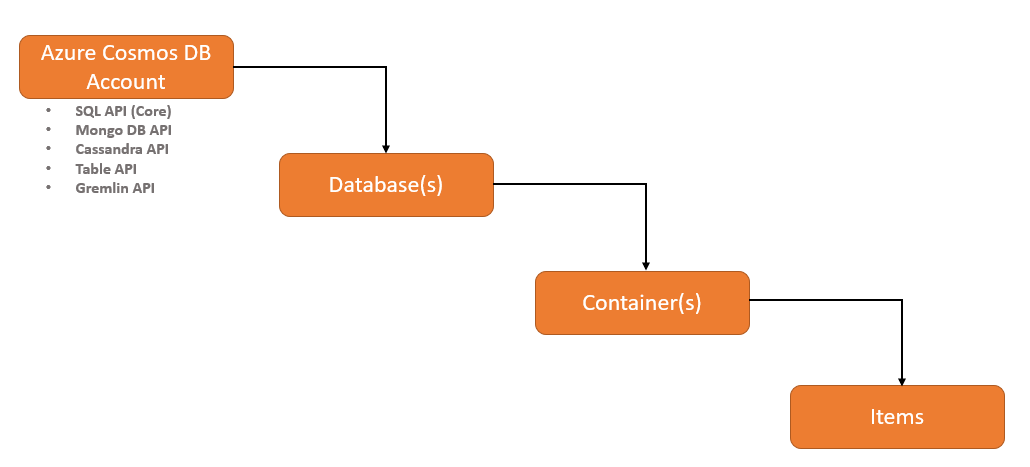
# Measurement Unit:

Since Azure Cosmos DB supports many APIs, each API has its own database operations such as read, write, query, update and delete. Each database operation consumes system resources based on the complexity of the operation. The cost of all database operations is measured in terms of ***Request Units (RUs)***. It abstracts system resources such as CPU, memory and IOPS that required to perform all the database operations. Finally the total cost is billed for the provisioned throughput(RUs) and the consumed storage on a hourly basis.



# Azure Cosmos DB Structure:

Azure Cosmos DB account consists of multiple database. Each cosmos DB database is a unit of management for a set of schema-agnostic containers. A container is horizontally partitioned and replicated across multiple regions. Based on the partition key, items added in the container and provisioned throughput are distributed automatically across a set of logical partitions. Azure Cosmos DB allows you to set TTL (Time-To-Live) for each item in the container or set the same TTL for all the items in the container. Once the TTL is expired, those items will be deleted from the container, and the query on this container will not return the expired items.



# Use Cases:

* Real time telemetry for IOT and AI
  + IOT – Smart Cars, Smart cities and Smart building
  + Track digital action such as clicks and app events
* Real time recommendations
  + Recommendation to enhance user experience (e.g.) Ecommerce product suggestion, Netflix video recommendations
* Mission critical or Geo distributed application
  + Deliver consistent real time, always on user experience anywhere in the world
* NoSQL Migration