|  |
| --- |
| **Pattern and Guidelines: Cosmos DB**  This document covers the Patterns and Guidelines for Cosmos DB.  Contents  [1. Introduction 2](#_Toc19874609)  [2. References 2](#_Toc19874610)  [3. Overview 3](#_Toc19874611)  [4. Common Azure Cosmos DB use cases 3](#_Toc19874612)  [5. RBAC in Azure Cosmos DB 3](#_Toc19874613)  [6. Secure access to data in Azure Cosmos DB 5](#_Toc19874614)  [7. Ingesting and Managing data in Azure Cosmos DB 5](#_Toc19874615)  [8. Partitioning in Azure Cosmos DB 6](#_Toc19874616)  [9. Optimize cost in Azure Cosmos DB 6](#_Toc19874617)  [10. Cosmos DB Patterns 6](#_Toc19874618)  [11. Lambda Architecture with Azure Cosmos DB and HDInsight (Apache Spark) 10](#_Toc19874619)  [12. Guidelines on Cosmos DB supported data models 10](#_Toc19874620)  [13. Best practices for consistency level 13](#_Toc19874621)  [14. Best practices for partitioning 13](#_Toc19874622)  [15. Monitor and debug with metrics 13](#_Toc19874623)  [16. Connect Azure Cosmos DB using other Azure Services 13](#_Toc19874624)  [17. Online backup and on-demand data restore in Azure Cosmos DB 13](#_Toc19874625)  [18. Security in Azure Cosmos DB 13](#_Toc19874626)  [19. Advanced Threat Protection for Azure Cosmos DB 13](#_Toc19874627)  [20. Configure alerts 14](#_Toc19874628) |

|  |
| --- |
| Introduction |

Today’s applications are required to be highly responsive and always online. To achieve low latency and high availability, instances of these applications need to be deployed in datacenters that are close to their users. Applications need to respond in real time to large changes in usage at peak hours, store ever increasing volumes of data, and make this data available to users in milliseconds.

Azure Cosmos DB is Microsoft's globally distributed, multi-model database service. With a click of a button, Cosmos DB enables you to elastically and independently scale throughput and storage across any number of Azure regions worldwide. You can elastically scale throughput and storage, and take advantage of fast, single-digit-millisecond data access using your favorite API including SQL, MongoDB, Cassandra, Tables, or Gremlin. Cosmos DB provides comprehensive [service level agreements (SLAs)](https://azure.microsoft.com/en-us/support/legal/sla/cosmos-db/v1_3/) for throughput, latency, availability, and consistency guarantees, something no other database service offers.

Azure Cosmos DB is a fully managed database service with turnkey global distribution and transparent multi-master replication. You can run globally distributed, low-latency operational and analytics workloads and AI on transactional data within your database

The following is not exhaustive in terms of information and guidance on Cosmos DB but highlights the areas to which you should pay attention when using Cosmos DB and what will hurt you interms of cost and performance.

|  |
| --- |
| References |

It is advisable to read the following to familiarise yourself with what Cosmos DB and whats its strengths and benefits offer businesses. It is important also, to understand the costs impacts that misuse or misconfiguration of Cosmos could mean. Read content linked below for more background and indepth understanding.

1. [Azure Cosmos DB Documentations](https://docs.microsoft.com/en-us/azure/cosmos-db/)
2. [Azure Cosmos DB Introduction](https://docs.microsoft.com/en-us/azure/cosmos-db/introduction)
3. [Technical overview of Azure Cosmos DB](https://azure.microsoft.com/en-us/blog/a-technical-overview-of-azure-cosmos-db/)
4. [Regional presence with Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/regional-presence)
5. [Cosmos DB Playground](https://www.documentdb.com/sql/demo)
6. [Azure Cosmos DB query cheat sheets](https://docs.microsoft.com/en-gb/azure/cosmos-db/query-cheat-sheet)
7. Learning: Microsoft Courses: [Azure Cosmos DB](https://docs.microsoft.com/en-gb/learn/browse/?products=azure-cosmos-db)
8. Learning: Pluralsight : [Azure Cosmos DB](https://app.pluralsight.com/library/courses/azure-cosmos-db/table-of-contents)
9. Lambda Architecture : [Lambda Architecture with Azure Cosmos DB and HDInsight (Apache Spark)](https://github.com/Azure/azure-cosmosdb-spark/blob/master/samples/lambda/readme.md)
10. Azure Cosmos DB Use cases : [Common Azure Cosmos DB use cases](https://docs.microsoft.com/en-us/azure/cosmos-db/use-cases)
11. [Estimate workload cost on Azure Cosmos DB](https://cosmos.azure.com/capacitycalculator/)
12. [Understand cost of Azure Cosmos DB for simple write and read operations](https://docs.microsoft.com/en-gb/azure/cosmos-db/key-value-store-cost)
13. [Azure Cosmos DB partitioning design patterns – Part 1](https://azure.microsoft.com/en-us/blog/azure-cosmos-db-partitioning-design-patterns-part-1/)
14. [Understand how to manage an Azure Cosmos account](https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-manage-database-account)
15. [Monitor and debug Azure Cosmos DB metrics](https://docs.microsoft.com/en-us/azure/cosmos-db/cosmos-db-azure-monitor-metrics)
16. [Online backup and on-demand data restore in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/online-backup-and-restore)
17. [Security in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/database-security)
18. [Advanced Threat Protection for Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/cosmos-db-advanced-threat-protection)
19. [Copy data to or from Azure Cosmos DB (SQL API) by using Azure Data Factory](https://docs.microsoft.com/en-gb/azure/data-factory/connector-azure-cosmos-db?toc=https%3A%2F%2Fdocs.microsoft.com%2Fen-gb%2Fazure%2Fcosmos-db%2FTOC.json&bc=https%3A%2F%2Fdocs.microsoft.com%2Fen-gb%2Fazure%2Fbread%2Ftoc.json)

An Important area for knowledge are the Build 2019 sessions located [here](https://mybuild.techcommunity.microsoft.com/sessions?t=%257B%2522from%2522%253A%25222019-05-06T08%253A00%253A00-07%253A00%2522%252C%2522to%2522%253A%25222019-05-08T19%253A45%253A00-07%253A00%2522%257D&q=cosmos#top-anchor). A key session to view (we would call this mandatory viewing) is (specifically related to costs and performance):

1. [Best Practices for Azure Cosmos DB: Data modelling, Partitioning and RU’s](https://medius.studios.ms/Embed/Video-nc/B19-BRK3054?latestplayer=true)

## 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. Cosmos DB is sometimes referred to as a serverless database, and it is a highly available, highly reliable, and high throughput database. Cosmos DB is a superset of Azure Document DB and is available in all Azure regions.

With Cosmos DB you can distribute the data to any number of Azure regions, i.e., the data can be replicated to the geolocation from where your users are accessing, which helps in serving data quickly to users with low latency.

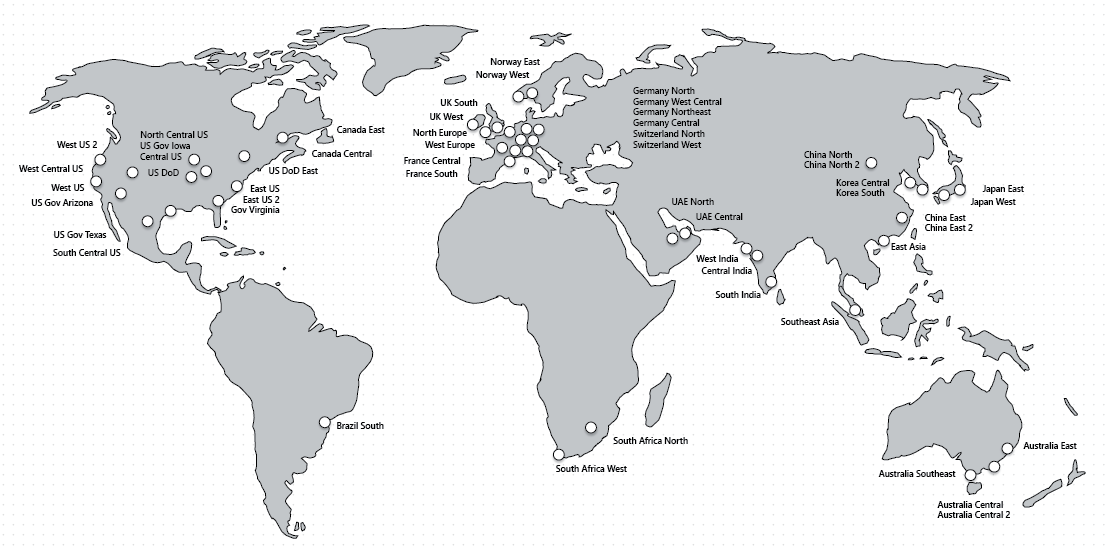


Figure : Regional Footprint

Please read the documentation in [this article](https://docs.microsoft.com/en-gb/azure/cosmos-db/key-value-store-cost) to understand what influences costs associated with Cosmos DB. Especially Request Units and the relationship with Reads and Writes.

|  |
| --- |
| Common Azure Cosmos DB use cases |

[This article](https://docs.microsoft.com/en-us/azure/cosmos-db/use-cases) provides an overview of several common use cases for Azure Cosmos DB. The recommendations in this article serve as a starting point as you develop your application with Cosmos DB.

|  |
| --- |
| RBAC in Azure Cosmos DB |

Azure Cosmos DB provides built-in role-based access control (RBAC) for common management scenarios in Azure Cosmos DB. An individual who has a profile in Azure Active Directory can assign these RBAC roles to users, groups, service principals, or managed identities to grant or deny access to resources and operations on Azure Cosmos DB resources.

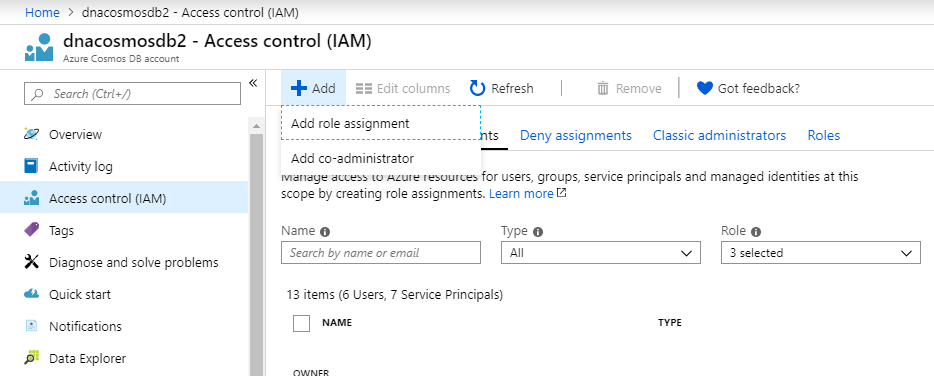
***Note****: Role assignments are scoped to control-plane access only, which includes access to Azure Cosmos accounts, databases, containers, and offers (throughput). Data plane operations are secured using* ***master keys or resource tokens****. To learn more, see* ***Secure access to data in Azure Cosmos DB*** *section below*

The following are the built-in roles supported by Azure Cosmos DB:

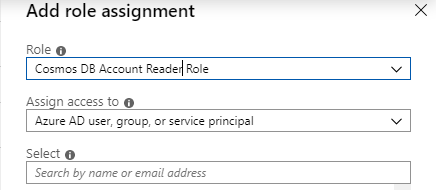
| **Built-in role** | **Description** |
| --- | --- |
| [DocumentDB Accounts Contributor](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#documentdb-account-contributor) | Can manage Azure Cosmos DB accounts. |
| [Cosmos DB Account Reader](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#cosmos-db-account-reader-role) | Can read Azure Cosmos DB account data. |
| [Cosmos Backup Operator](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#cosmosbackupoperator) | Can submit restore request for an Azure Cosmos database or a container. |
| [Cosmos DB Operator](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles#cosmos-db-operator) | Can provision Azure Cosmos accounts, databases, and containers but cannot access the keys that are required to access the data. |

Below are the basic steps to provide above role access to users. Below example demonstrate how to provide **Azure Cosmos DB account reader** access to user

1. Open the Azure portal, and select your Azure Cosmos DB account.
2. Click the **Access control (IAM)** tab, and then click **+ Add role assignment**.



1. In the **Add role assignment** pane, in the **Role** box, select **Cosmos DB Account Reader Role**.





1. In the **Assign access to box**, select **Azure AD user, group, or application**.
2. Select the user, group, or application in your directory to which you wish to grant access. You can search the directory by display name, email address, or object identifiers. The selected user, group, or application appears in the selected members list.
3. Click **Save**.

|  |
| --- |
| Secure access to data in Azure Cosmos DB |

Azure Cosmos DB uses two types of keys to authenticate users and provide access to its data and resources.

| **Key type** | **Resources** |
| --- | --- |
| [Master keys](https://docs.microsoft.com/en-us/azure/cosmos-db/secure-access-to-data#master-keys) | Used for administrative resources: database accounts, databases, users, and permissions |
| [Resource tokens](https://docs.microsoft.com/en-us/azure/cosmos-db/secure-access-to-data#resource-tokens) | Used for application resources: containers, documents, attachments, stored procedures, triggers, and UDFs |

Each account consists of two Master keys: a primary key and secondary key. The purpose of dual keys is so that you can regenerate, or roll keys, providing continuous access to your account and data.

In addition to the two master keys for the Cosmos DB account, there are two read-only keys. These read-only keys only allow read operations on the account. Read-only keys do not provide access to read permissions resources.

Refer [this](https://docs.microsoft.com/en-us/azure/cosmos-db/secure-access-to-data) article for detail understanding on securing access to data in Azure Cosmos DB

|  |
| --- |
| Ingesting and Managing data in Azure Cosmos DB |

* There are many ways to ingest data in Azure Cosmos DB

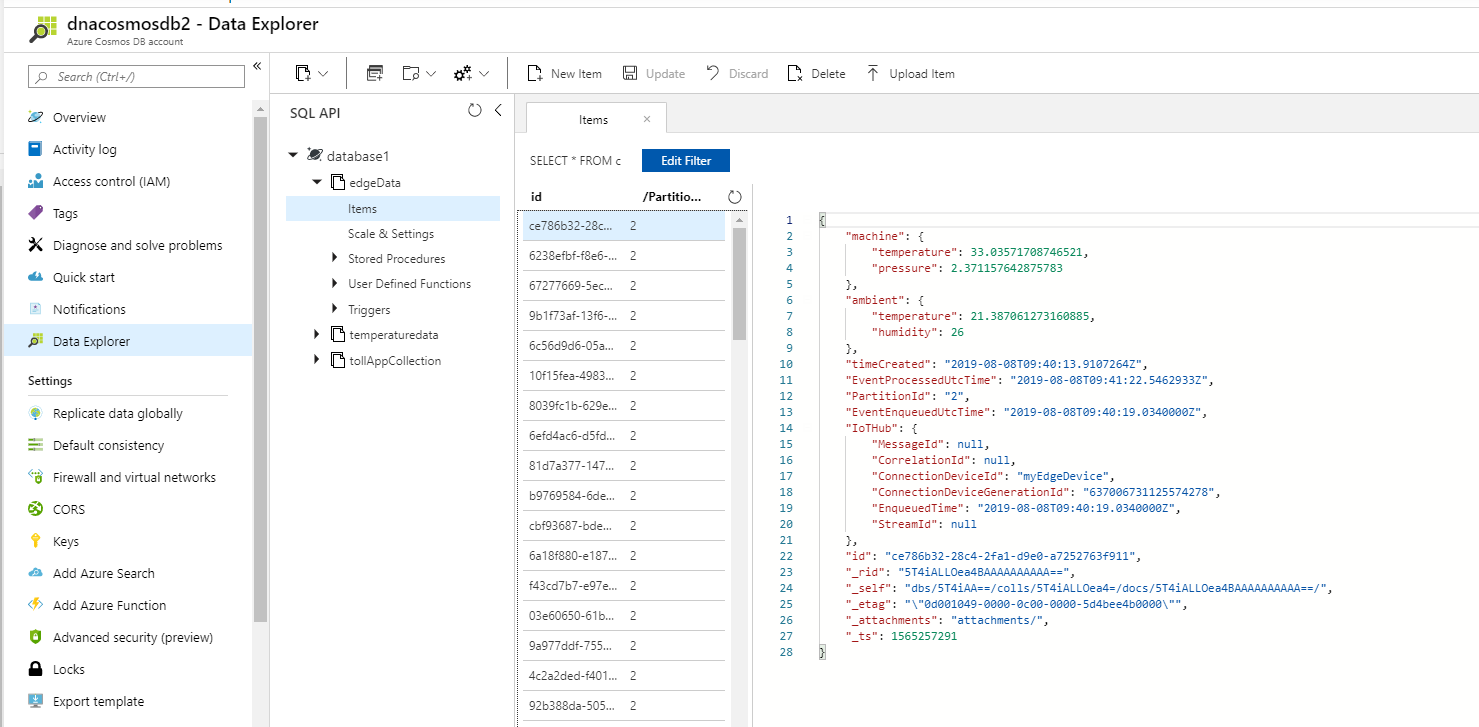
1. Ingest streaming data in Azure Cosmos DB using Azure Stream Analytics – Refer [this](https://eu001-sp.shell.com/:w:/r/sites/AAFAA5019/ITSO%20Enterprise%20Service%20Portfolios%20Public%20Library/DnA%20Programme/01.%20Projects/08.%20Project%204.4b%20Stream%20Analytics/03.%20Execute/02.%20SOP/ASOM%20-%20SOP/Stream%20Analytics/Azure%20Stream%20Analytics%20-%20Setup%20Input,%20Output%20and%20Query.docx?d=w5879073f0bb24d62b62da10611b8e6c4&csf=1&e=nuugzJ) document for understanding how to setup output in stream analytics

2. Ingest data using Azure Data Factory – refer [this](https://docs.microsoft.com/en-us/azure/data-factory/connector-azure-cosmos-db?toc=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fazure%2Fcosmos-db%2FTOC.json&bc=https%3A%2F%2Fdocs.microsoft.com%2Fen-us%2Fazure%2Fbread%2Ftoc.json) article for more understanding

* There are various ways to manage Data in Azure Cosmos DB

1. Refer [Managing data in Azure Cosmos DB using Storage Explorer](https://eu001-sp.shell.com/:w:/r/sites/AAFAA5019/ITSO%20Enterprise%20Service%20Portfolios%20Public%20Library/DnA%20Programme/01.%20Projects/08.%20Project%204.4b%20Stream%20Analytics/03.%20Execute/02.%20SOP/ASOM%20-%20SOP/Cosmos%20DB/Azure%20Cosmos%20DB%20-%20Connecting%20using%20Microsoft%20Azure%20Storage%20Explorer.docx?d=wea4d57cd937045228e049bdf7d9e23cb&csf=1&e=iul9cW)

2. Manage data using Data Explorer (see below)



|  |
| --- |
| Partitioning in Azure Cosmos DB |

Azure Cosmos DB uses partitioning to scale individual containers in a database to meet the performance needs of your application. In partitioning, the items in a container are divided into distinct subsets called logical partitions. Logical partitions are formed based on the value of a partition key that is associated with each item in a container. All items in a logical partition have the same partition key value.

For example, a container holds items. Each item has a unique value for the UserID property. If UserID serves as the partition key for the items in the container and there are 1,000 unique UserID values, 1,000 logical partitions are created for the container.

In addition to a partition key that determines the item’s logical partition, each item in a container has an item ID (unique within a logical partition). Combining the partition key and the item ID creates the item's index, which uniquely identifies the item.

Refer below articles on Patitioning in Azure Cosmos DB

* [Managing logical partitions and Choosing a partition key](https://docs.microsoft.com/en-us/azure/cosmos-db/partitioning-overview)
* [Partitioning and horizontal scaling in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/partition-data)
* [Create a synthetic partition key](https://docs.microsoft.com/en-us/azure/cosmos-db/synthetic-partition-keys)

|  |
| --- |
| Optimize cost in Azure Cosmos DB |

With Azure Cosmos DB, you pay for the throughput provisioned and the storage that you consume. Understand pricing model and how to optimize various cost in Azure Cosmos DB

* [Understand Pricing model in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/how-pricing-works)
* [Optimize provisioned throughput cost in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-throughput)
* [Optimize query cost in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-queries)
* [Optimize storage cost in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-storage)
* [Optimize reads and writes cost in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-reads-writes)
* [Optimize multi-region cost in Azure Cosmos DB](https://docs.microsoft.com/en-us/azure/cosmos-db/optimize-cost-regions)

|  |
| --- |
| Cosmos DB Patterns |

1. **Azure Cosmos DB partitioning design patterns**

In this document, you will learn how to use partition keys to efficiently distribute data, improve application performance, and enable faster look-up. The pre-requisites of this article are general knowledge of Azure Cosmos DB and a good understanding of [change feed](https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed), [request unit](https://docs.microsoft.com/en-us/azure/cosmos-db/request-units) (RU), and [Azure Functions](https://azure.microsoft.com/en-us/services/functions/).

Imagine you have data which you would like to insert with high throughput and query on two or more different keys. In this scenario, suppose you work for an airline company and need to store user reservation information in a collection. User data is defined as:

{

UserId: user@email.com,

FirstName: John,

LastName: Doe,

PNR: 2345423,

CityOfOrigin: Seattle,

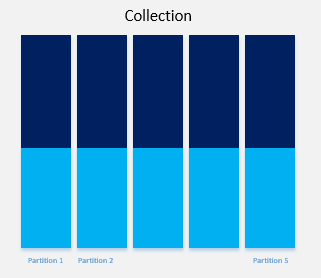
CityOfDestination: London,

DOB: 12.2.76,

other details ….

}

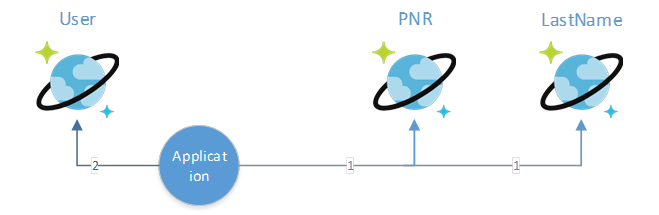
Out of many possible values, you choose “UserId” (user email address) as the partition key. This is a good choice for a partition key because “UserId” is unique for every user, ensuring your data remains well distributed. Your data is distributed evenly among all partitions, as shown in Figure 1. However, when you are querying the data you do not always have a “UserId”. Sometimes you want to query the data by user last name or user Passenger Name Record (PNR) number.

[](https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/59bc2d78-5964-4bd1-ad72-a58dd800c68e.png)

*Figure 1: Data distributed evenly across partitions*

Azure Cosmos DB indexes all data by default. If you try to query the data by "LastName", you will get the result, but it will cost you more [request units](https://docs.microsoft.com/en-us/azure/cosmos-db/request-units) (RU/s) because queries without partition key become fan-out queries. Fan-out queries check all partitions, which will cost you extra RU/s and may affect the performance of your application. If you have a small number of partitions with less data, you may not perceive any significant side effects of fan-out queries, but when you start getting in high numbers of partitions and large amounts of data, fan-out queries can be detrimental to your applications. Infrequent cross partition query is fine, but if it is a frequent query, then what is the solution?

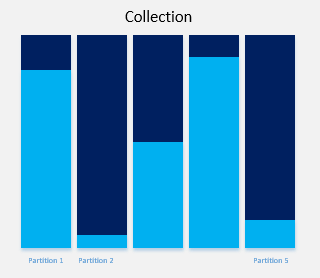
One option is to have two more lookup collections PNR and “LastName” for the mapping of PNR to “UserId”, and “LastName” to “UserId”. The PNR collection will have PNR as the partition key and row key and “UserId” as the value.

[](https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/9d2f02b1-e0d0-447f-a4cb-58eb5ec6e98b.png)

These different lookup collections will make your application more efficient. To lookup detail by PNR, first query the PNR collection to get the “UserId”. Then use “UserId” to query user collection for all details. These two calls can complete within a few milliseconds and will consume fewer RU/s than a single fan-out query. Most of the point lookup can complete within one to two milliseconds. Even after two lookups, you can be done within 10 milliseconds for most of the queries.

You may not want a few extra milliseconds added to calls and instead may decide to duplicate the data in the PNR and “LastName” collections. While this will enable fast look-up, it isn’t recommended because it may add complexity and costs when data is updated. Ultimately, you must balance requirements, performance, and complexity. Starting with a simple solution is often the best approach.

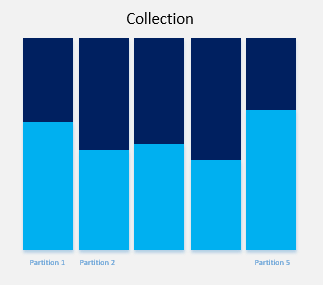
Now let’s look at the distribution of your data in different collections. For example, if you look at the “LastName” collection you will find that the data is not equally distributed because there are more people with the last name “Smith” than “Zubrkee”. In this case, the data will look very much like Figure 2.

[](https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/02535643-5f76-4e7a-9483-9b3d70a7de97.png)

*Figure 2: Data unevenly distributed across partitions*

The data in this scenario is distributed unevenly, with some full partitions and others greatly underused. This type of data distribution creates many problems.

* The total RU/s for a collection is divided amongst all partitions. This means that 1000 RU/s will be distributed across five partitions as 200 RU/s for each partition. If you try to write more than 200 RU/s to any of these partitions, calls will begin to fail as you cross the threshold. If a developer finds they’re getting throttled on 300 RU/s despite having 1000 RU/s allocated at the collection level, the issue is often that a bad partition key is hitting only one partition.
* A partition key can currently have 10 GB of data at maximum (this may change in future), making it important to use a partition key which fills all partitions efficiently. In the example of “LastName”, a more granular partition key is needed to evenly distribute data. Since the data also includes “Origin City”, a new partition key can be made from “LastName” and “Origin City” and the results will look like Figure 3 below.

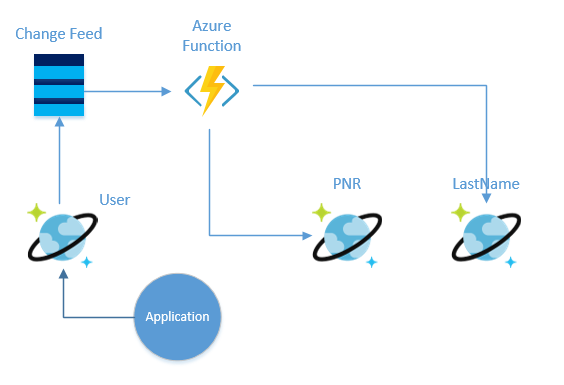
[](https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/68082817-1ab1-4a3c-9d23-b6b4fb6bf7cd.png)

*Figure 3: Data distribution after applying granular partition key*

This looks much better. The data is distributed more evenly, and travelers can easily and quickly look-up their reservations by simply entering their last name and city of origin.

Now that you’ve distributed your data evenly, how do you populate the other collection? For this, we need to understand change feed. [Change feed](https://docs.microsoft.com/en-us/azure/cosmos-db/change-feed) exposes all the internal changes happening inside a collection. Change feed support in Azure Cosmos DB works by listening to an Azure Cosmos DB collection for any changes. It then outputs the sorted list of documents that were changed in the order in which they were modified. The changes are persisted, can be processed asynchronously and incrementally, and the output can be distributed across one or more consumers for parallel processing.

The change feed is available for each partition key range within the document collection and thus can be distributed across one or more consumers for parallel processing. Whenever a record is inserted into the user collection it will appear in change feed. The easiest way to consume change feed is Azure Function. Azure Functions is a [serverless](https://azure.microsoft.com/solutions/serverless/) compute service that enables you to run code-on-demand without having to explicitly provision or manage infrastructure. Use Azure Functions to run a script or piece of code in response to a variety of events.

[](https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/c05eafc5-3823-4fdb-93de-8f4ab499fe4b.png)

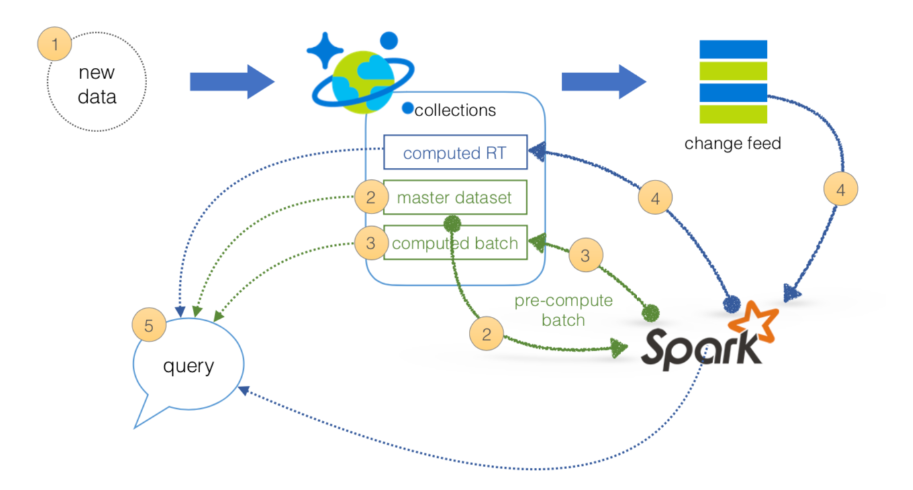
When you consume change feed through Azure function, all the inserted/changed documents come in your function as a parameter of your function.

public static async Task Run(IReadOnlyList<Document> input, TraceWriter log)

Once you have the whole document in your function, you can update the PNR and “LastName” collection accordingly.

|  |
| --- |
| Lambda Architecture with Azure Cosmos DB and HDInsight (Apache Spark) |

we can simplify Lambda Architecture by using Azure Cosmos DB, the Cosmos DB Change Feed to avoid the need to multi-cast data between the batch and speed layers, Apache Spark on HDInsight, and the Spark Connector for Azure Cosmos DB.



With this design, we need only two managed services that together will address the batch, serving, and speed layers of Lambda Architecture simplifying not only the operations but also the data flow.

1. All data pushed into Cosmos DB layer for processing
2. The batch layer has a master dataset (immutable, append-only set of raw data) and pre-compute the batch views
3. The serving layer has batch views so data for fast queries.
4. The speed layer compensates for processing time (to serving layer) and deals with recent data only.
5. All queries can be answered by merging results from batch views and real-time views.

For more details – refer [this](https://github.com/Azure/azure-cosmosdb-spark/blob/master/samples/lambda/readme.md) article

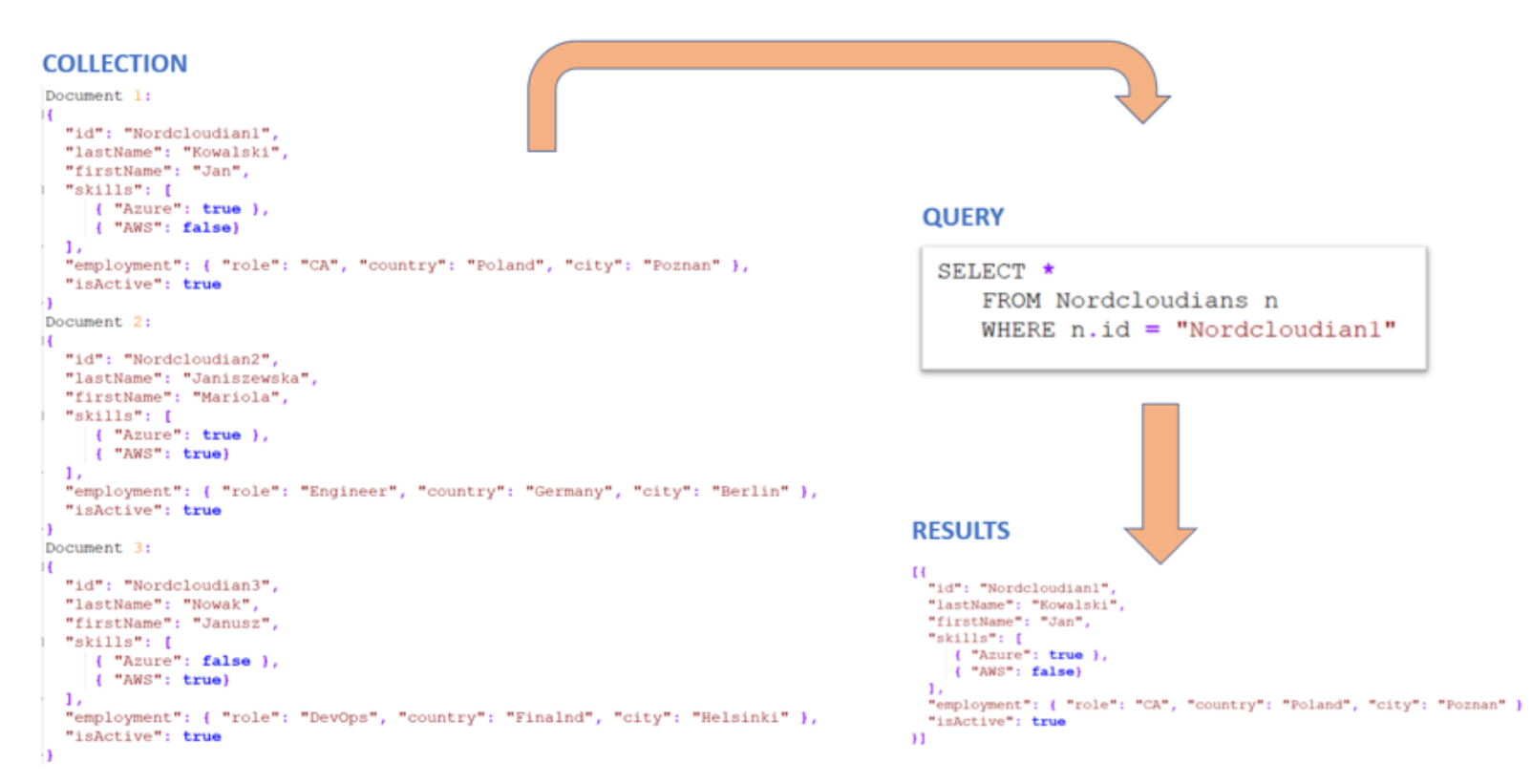
|  |
| --- |
| Guidelines on Cosmos DB supported data models |

Below are the supported data models in Cosmos DB. All of them use main, cool features provided by Azure Cosmos DB.

1. **SQL API**

Azure Cosmos DB SQL API accounts support querying items using Structured Query Language (SQL) as a JSON query language. The design goals of the Azure Cosmos DB query language are to:

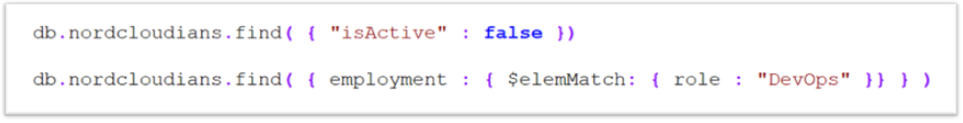
* Support SQL, one of the most familiar and popular query languages, instead of inventing a new query language. SQL provides a formal programming model for rich queries over JSON items.
* Use JavaScript's programming model as the foundation for the query language. JavaScript's type system, expression evaluation, and function invocation are the roots of the SQL API. These roots provide a natural programming model for features like relational projections, hierarchical navigation across JSON items, self-joins, spatial queries, and invocation of user-defined functions (UDFs) written entirely in JavaScript.



1. **MONGODB API**

Existing instances of MongoDB can be migrated to Azure Cosmos DB without huge effort for this activities. Both standards are compatible. If new environment is created, change between native MongoDB instance and Cosmos DB instance (by MongoDB API) comes to change a connection string in application. Existing drivers for application written for MongoDB are fully supported. By design all properties within documents are automatically indexed.

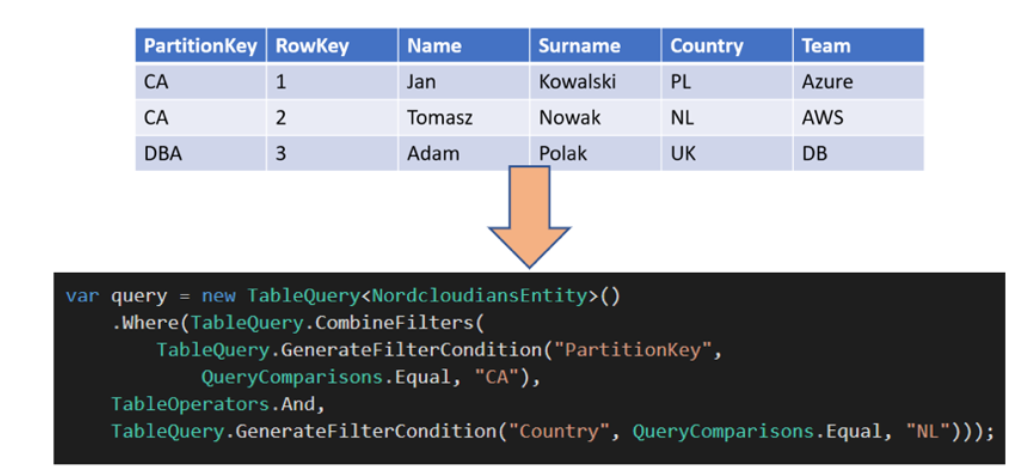
Let’s check, how simple queries for identical documents collection as used in previous point will look like:



As a result, specified sub-JSON contains data will be returned. If query doesn’t return results, empty object will be send as a response to query.

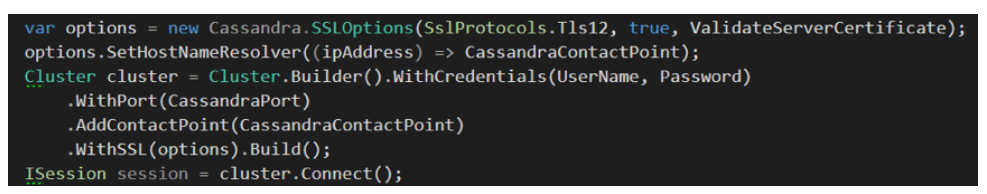
1. **TABLE API**

This kind of API can be used by applications prepared natively for close working with Azure Storage tables. Of course Cosmos DB provides some premium capabilities comparing to Storage tables e.g. high availability or global distribution. Migration to new DB source for application doesn’t require changes in code. User can query data in a few ways. Also lot of SDKs are provided by design. Below sample will show how to query data by .NET SDK with LINQ. During execution LINQ query will be translated to ODATA query expression.



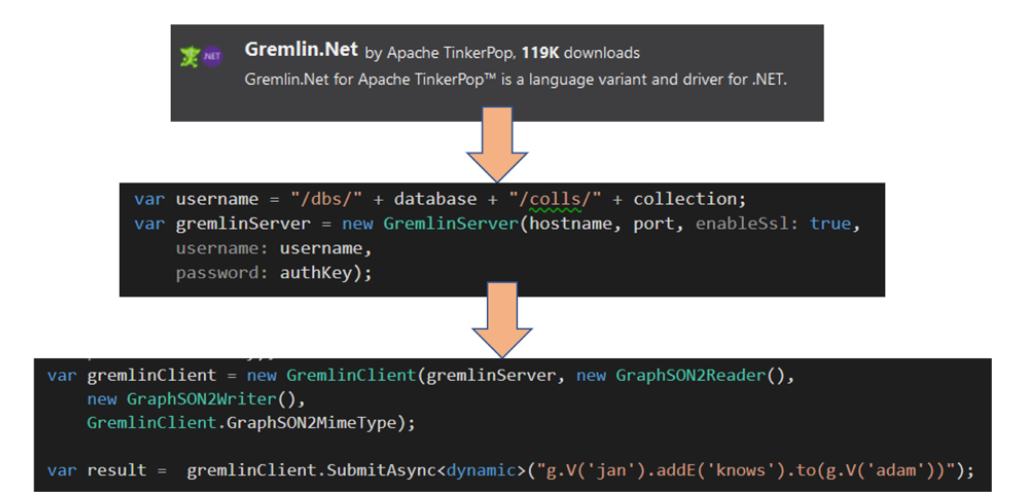
1. **CASSANDRA API**

Azure Cosmos DB Cassandra API is dedicated data store for applications created for Apache Cassandra. User is able to interact with data via CQL (Cassandra Query Language). In many cases action for changing DB source from Apache Cassandra to Azure Cosmos DB ‘s Cassandra API is just changing a connection string in application. From code perspective integration with Cassandra is realized via dedicated SDK (NuGet -> Install-Package CassandraCSharpDriver). Sample code for connecting to Cassandra cluster from .NET application is presented below.



1. **GREMLIN API**

The last API provided by Azure Cosmos DB is Gremlin API. This kind of interface can be used for storing and operation on graph data. API supports natively possibilities to graph modeling and traversing. We can query the graphs with millisecond latency and evolve the graph structure and schema in easy way. For querying activities we can use Gremlin or Apache TinkerPop languages. Step by step process from NuGet package installation to run first query is has been shown below.



|  |
| --- |
| Best practices for consistency level |

Refer [Cosmos DB - Best Practices](https://eu001-sp.shell.com/:w:/r/sites/AAFAA5019/ITSO%20Enterprise%20Service%20Portfolios%20Public%20Library/DnA%20Programme/01.%20Projects/08.%20Project%204.4b%20Stream%20Analytics/03.%20Execute/02.%20SOP/ASOM%20-%20SOP/Cosmos%20DB/Cosmos%20DB%20-%20Best%20Practices.docx?d=w45c125474d5741f5a35c0ce826920c71&csf=1&e=JmxZfJ) for more information

|  |
| --- |
| Best practices for partitioning |

Refer [Cosmos DB - Best Practices](https://eu001-sp.shell.com/:w:/r/sites/AAFAA5019/ITSO%20Enterprise%20Service%20Portfolios%20Public%20Library/DnA%20Programme/01.%20Projects/08.%20Project%204.4b%20Stream%20Analytics/03.%20Execute/02.%20SOP/ASOM%20-%20SOP/Cosmos%20DB/Cosmos%20DB%20-%20Best%20Practices.docx?d=w45c125474d5741f5a35c0ce826920c71&csf=1&e=JmxZfJ) for more information

|  |
| --- |
| Monitor and debug with metrics |

Azure Cosmos DB provides metrics for throughput, storage, consistency, availability, and latency. Go through Microsoft’s detail article on monitoring with ComosDB [here](https://docs.microsoft.com/en-us/azure/cosmos-db/use-metrics)

|  |
| --- |
| Connect Azure Cosmos DB using other Azure Services |

Refer [Connect Azure Cosmos DB using Power BI and other services](https://eu001-sp.shell.com/:w:/r/sites/AAFAA5019/ITSO%20Enterprise%20Service%20Portfolios%20Public%20Library/DnA%20Programme/01.%20Projects/08.%20Project%204.4b%20Stream%20Analytics/03.%20Execute/02.%20SOP/ASOM%20-%20SOP/Cosmos%20DB/Connect%20Azure%20Cosmos%20DB%20using%20Power%20BI%20and%20other%20services.docx?d=w755032e140bc417fab2bb9ae95186b5a&csf=1&e=egqDXm) for more information

|  |
| --- |
| Online backup and on-demand data restore in Azure Cosmos DB |

Azure Cosmos DB automatically takes backups of your data at regular intervals. The automatic backups are taken without affecting the performance or availability of the database operations. All the backups are stored separately in a storage service, and those backups are globally replicated for resiliency against regional disasters. The automatic backups are helpful in scenarios when you accidentally delete or update your Azure Cosmos account, database, or container and later require the data recovery.

Refer detail information on this features [here](https://docs.microsoft.com/en-us/azure/cosmos-db/online-backup-and-restore)

|  |
| --- |
| Security in Azure Cosmos DB |

This [article](https://docs.microsoft.com/en-us/azure/cosmos-db/database-security) discusses database security best practices and key features offered by Azure Cosmos DB to help prevent, detect, and respond to database breaches

|  |
| --- |
| Advanced Threat Protection for Azure Cosmos DB |

Advanced Threat Protection for Azure Cosmos DB provides an additional layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit Azure Cosmos DB accounts. This layer of protection allows to address threats, even without being a security expert, and integrate them with central security monitoring systems.

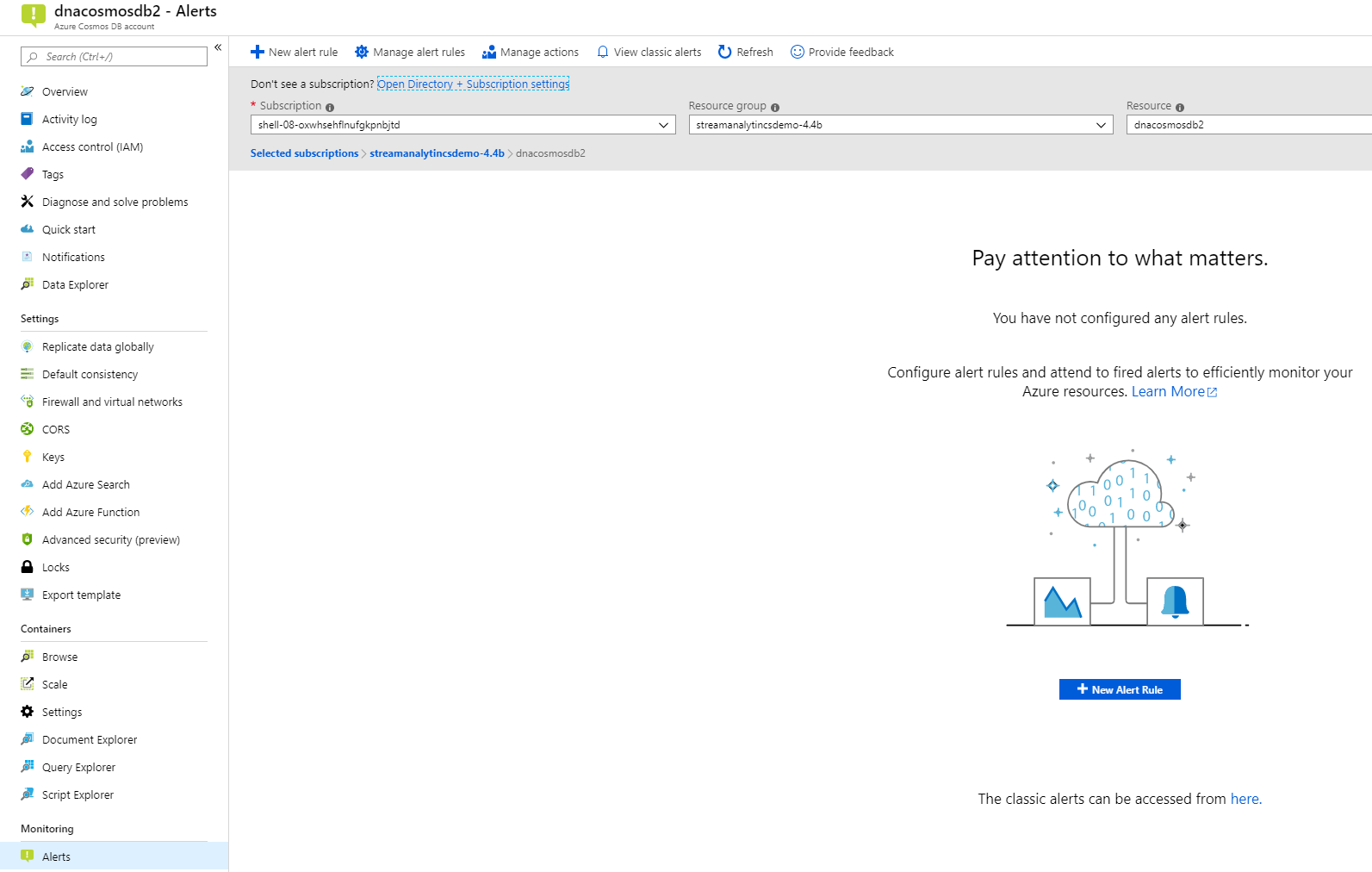
Security alerts are triggered when anomalies in activity occur. These security alerts are integrated with [Azure Security Center](https://azure.microsoft.com/en-us/services/security-center/), and are also sent via email to subscription administrators, with details of the suspicious activity and recommendations on how to investigate and remediate the threats.

Refer [this](https://docs.microsoft.com/en-us/azure/cosmos-db/cosmos-db-advanced-threat-protection) arcticle to understand how to enable this feature

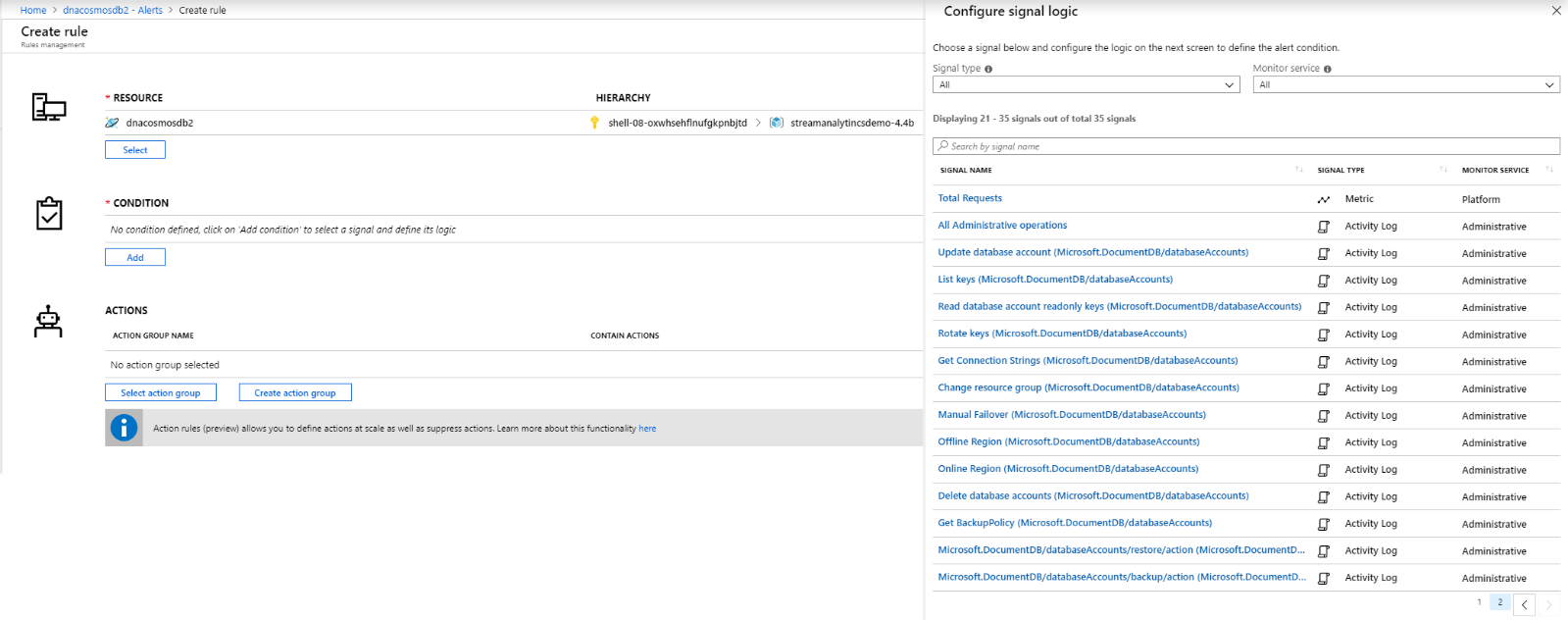
|  |
| --- |
| Configure alerts |

It's important to monitor Azure Cosmos DB’s throughput, storage, availability, latency, and consistency. these steps describe how to set up alerts for common scenarios that should be monitored.

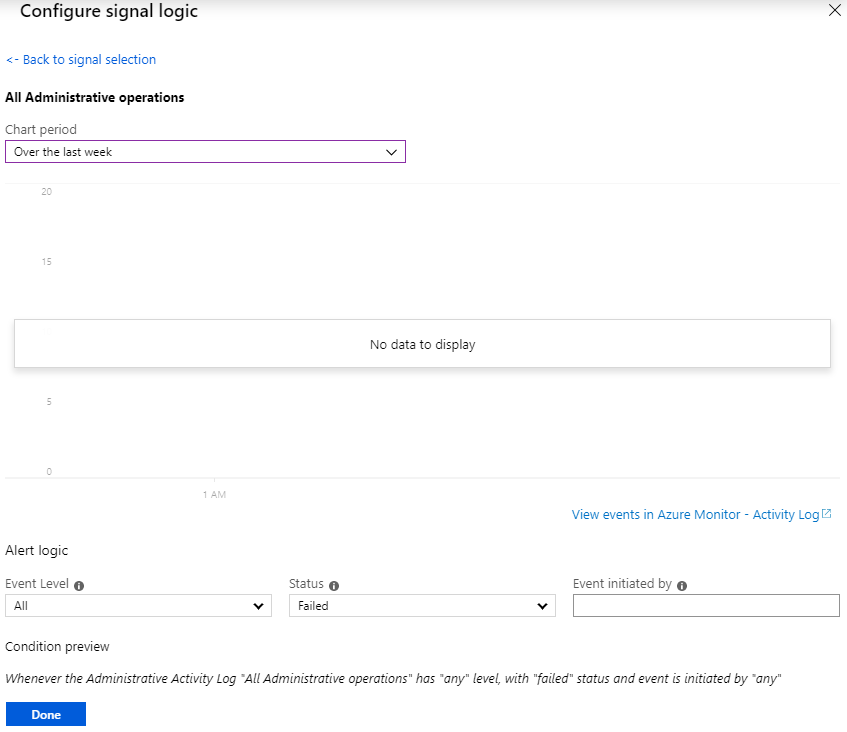
* In the Azure portal, open the Azure Comos DB you want to create an alert for.
* On the **Account** page, navigate to the **Monitoring** section.
* Select **Alert**, and then New **alert rule**.



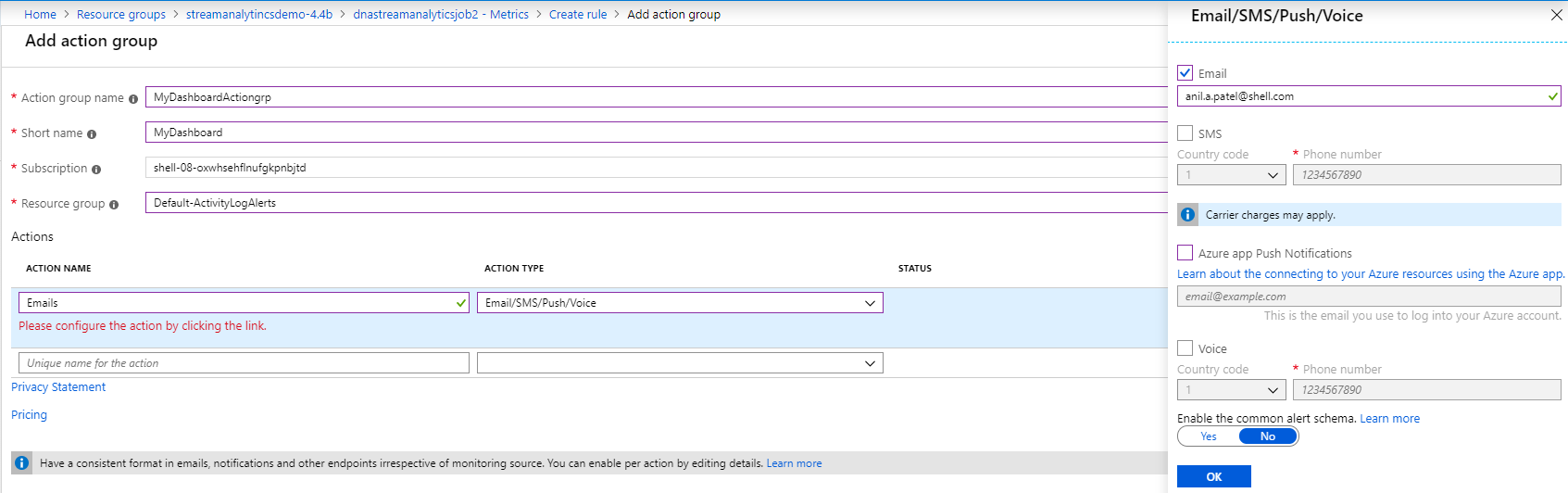
* Your Cosmos DB account name should automatically appear under **RESOURCE**. If not then select Cosmos DB account. Click **Add condition**, and select **All Administrative operations**.



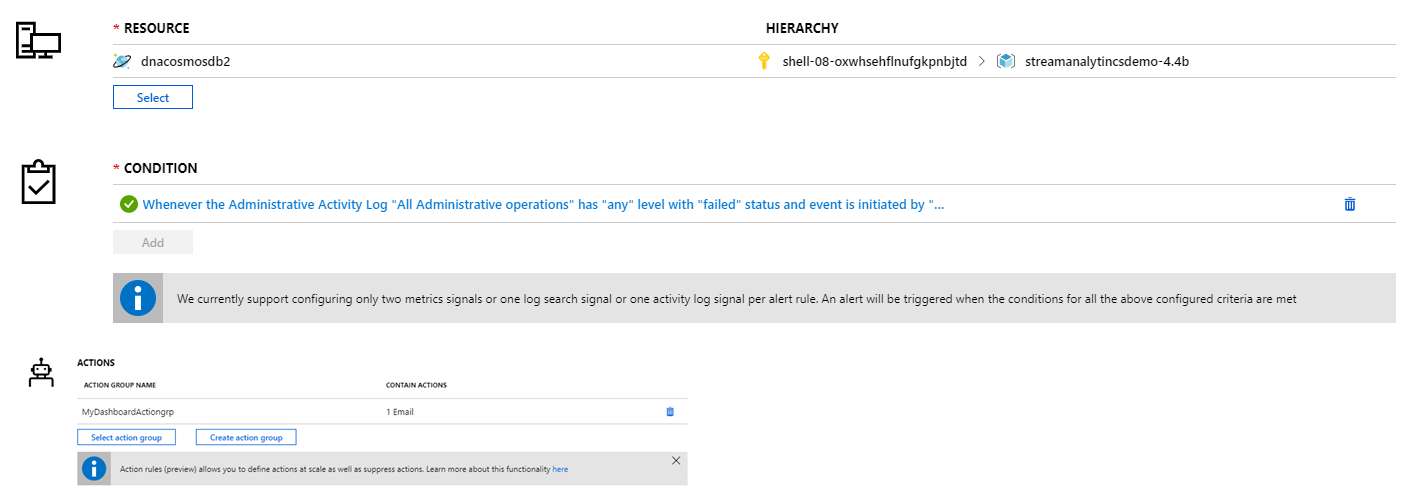
* Under **Configure signal logic**, change **Event Level** to **All** and change **Status** to **Failed**. Leave **Event initiated by** blank and select **Done**.



* Select an existing action group or create a new group. In this example, a new action group called **MyDashboardActiongrp** was created with an **Emails** action that sends an email to users.



* The **RESOURCE**, **CONDITION**, and **ACTION GROUPS** should each have an entry. Note that in order for the alerts to fire, the conditions defined need to be met. For example, you can measure a metric's average value of over the last 15 minutes, every 5 minutes.



* Add an **Alert rule name**, **Description**, and your **Resource Group** to the **ALERT DETAILS** and click **Create alert rule** to create the rule for your Stream Analytics job.

