



# DP-200T01: Building Globally Distributed Databases with Cosmos DB



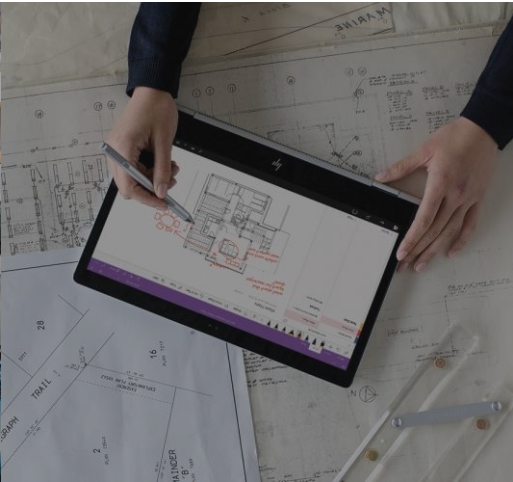
# Agenda

- L01 – Create an Azure Cosmos DB database built to scale
- L02 - Insert and query data in your Azure Cosmos DB database
- L03 - Build a .NET Core app for Azure Cosmos DB in Visual Studio Code
- L04 - Distribute your data globally with Azure Cosmos DB



# Lesson 01

## Create an Azure Cosmos DB Database Built to Scale



# Lesson Objectives

- What is Cosmos DB
- Create an Azure Cosmos DB account
- What is a Request Unit
- Choose a partition key
- Create a database and container for NoSQL data in Azure Cosmos DB

# What is Azure Cosmos DB



Scalability



Performance



Availability



Programming  
Models

# Create an Azure Cosmos DB account.

Home > New > Create Azure Cosmos DB Account

## Create Azure Cosmos DB Account

Basics Networking Tags Review + create

Azure Cosmos DB is a globally distributed, multi-model, fully managed database service. [Try it for free](#), for 30 days with unlimited renewals. Go to production starting at \$24/month per database, multiple containers included. [Learn more](#)

### Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Resource Group \*

[Create new](#)

### Instance Details

Account Name \*

API \*

Apache Spark

[Notebooks](#) [Notebooks with Apache Spark](#)

[None](#)

[Sign up for Apache Spark preview](#)

Location \*

Geo-Redundancy

Multi-region Writes

\*Up to 33% off multi-region writes is available to qualifying new accounts only. Accounts must be created between December 1, 2019 and February 29, 2020. Offer limited to accounts with both account locations and geo-redundancy, and applies only to multi-region writes in those same regions. Both Geo-Redundancy and Multi-region Writes must be enabled in account settings. Actual discount will vary based on number of qualifying regions selected.

# What are Request Units

Throughput is important to ensure you can handle the volume of transactions you need.

## Database throughput

Database throughput is the number of reads and writes that your database can perform in a single second

## What is a Request Unit

Azure Cosmos DB measures throughput using something called a request unit (RU). Request unit usage is measured per second, so the unit of measure is request units per second (RU/s). You must reserve the number of RU/s you want Azure Cosmos DB to provision in advance

## Exceeding throughput limits

If you don't reserve enough request units, and you attempt to read or write more data than your provisioned throughput allows, your request will be rate-limited

# Choosing a Partition Key

## Why have a Partition Strategy

Having a partition strategy ensures that when your database needs to grow, it can do so easily and continue to perform efficient queries and transactions.

## What is a Partition Key

A partition key is the value by which Azure organizes your data into logical divisions.

## Best Practice.

### Range of Values

The more values your partition key has, the more scalability you have.

### Review Queries

To determine the best partition key for a read-heavy workload, review the top three to five queries you plan on using.


### Transactional Workloads


For write-heavy workloads, you'll need to understand the transactional needs of your workload.




# Creating a Database and a Container in Cosmos DB


Add Container ✕

 Start at \$24/mo per database, multiple containers included  
[More details](#)

\* Database id 


Create new  Use existing


Provision database throughput 

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


Autopilot (preview)  Manual

Estimated spend (USD): **\$0.032 hourly / \$0.77 daily** (1 region, 400RU/s, \$0.00008/RU)

\* Container id 

\* Partition key 

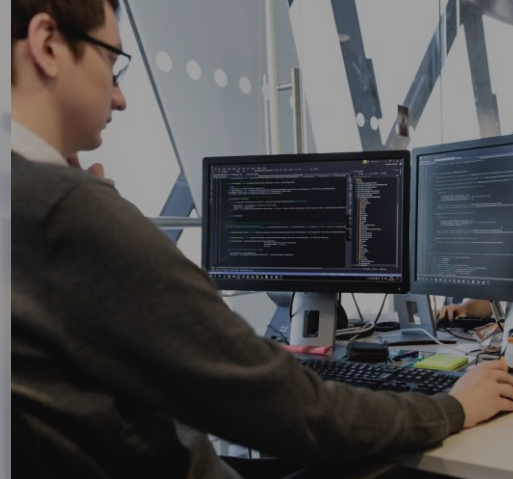
My partition key is larger than 100 bytes

Unique keys 

+ Add unique key

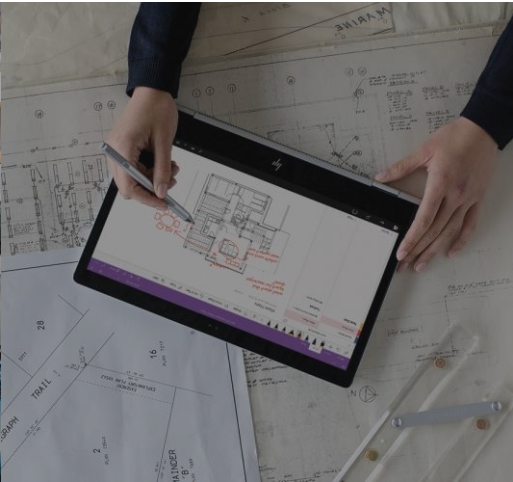
# Review Questions

- Q01 – You want to ensure that there is 99.999% availability for the reading and writing of all your data. How can this be achieved?
- A01 – By configuring reads and writes of data for multi-region accounts with multi region writes
  
- Q02 - What are the three main advantages to using Cosmos DB?
- Cosmos DB offers global distribution capabilities out of the box.
- Cosmos DB provides a minimum of 99.99% availability.
- Cosmos DB response times of read/write operations are typically in the order of 10s of milliseconds



# Lesson 02

## Insert and Query Data in your Azure Cosmos DB Database



# Lesson Objectives

- Create a product catalog document in the Data Explorer
  - Add data
- Perform Azure Cosmos DB queries
  - Query types
  - Run queries
- Running complex operations on your data
- Working with graph data

Create a product catalog documents in the Data Explorer.

The screenshot displays the Azure Cosmos DB Data Explorer interface for an account named 'awcdbstudcto'. The left sidebar contains navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quick start, Notifications, Data Explorer (selected), and Settings. The main area shows a tree view under 'SQL API' with 'Products' expanded to 'Clothing' and 'Items' selected. A table of 'Items' is shown with columns 'id' and '...' and three rows of data. A 'Load more' link is visible below the table. On the right, a JSON document is displayed, showing fields like 'id', 'productId', 'category', 'manufacturer', 'description', 'price', and 'shipping'.

awcdbstudcto - Data Explorer  
Azure Cosmos DB account

Search (Ctrl+/)

SQL API

- Products
  - Scale
  - Clothing
    - Items
    - Settings
    - Stored Procedures
    - User Defined Functions
    - Triggers
  - ToDoList
    - Items

Items

SELECT \* FROM c

id	/...
1	332...
2	332...
3	332...

Load more

```
{  
  "id": "1",  
  "productId": "332...",  
  "category": "W...",  
  "manufacturer": "...",  
  "description": "...",  
  "price": "14.99",  
  "shipping": {  
    "weight": 1,  
    "dimensions": {  
      "width": "1",  
      "height": "1",  
      "depth": "1"  
    }  
  },  
  "_rid": "P01tAM...",  
  "_self": "dbs/P...",  
  "_etag": "\"410..."  
}
```

# Perform Azure Cosmos DB Queries.

## **SELECT Query Basics**

```
SELECT <select_list>  
[FROM <optional_from_specification>]  
[WHERE <optional_filter_condition>]  
[ORDER BY <optional_sort_specification>]  
[JOIN <optional_join_specification>]
```

## **Examples**

```
SELECT *  
FROM Products p WHERE p.id ="1"
```

```
SELECT p.id, p.manufacturer, p.description  
FROM Products p WHERE p.id ="1"
```

```
SELECT p.price, p.description, p.productId  
FROM Products p ORDER BY p.price ASC
```

```
SELECT p.productId  
FROM Products p JOIN p.shipping
```

# Running complex operations on data

Multiple documents in your database frequently need to be updated at the same time. The way to perform these transactions in Azure Cosmos DB is by using stored procedures and user-defined functions (UDFs)

## Stored Procedures

Stored procedures perform complex transactions on documents and properties. Stored procedures are written in JavaScript and are stored in a collection on Azure Cosmos DB.

## User Defined Functions

User Defined Functions are used to extend the Azure Cosmos DB SQL query language grammar and implement custom business logic, such as calculations on properties and documents.

# Working with Graph Data

```
from gremlin_python.driver import client,
serializer
import sys, traceback

CLEANUP_GRAPH = "g.V().drop()"

INSERT_NATIONAL_PARK_VERTICES = [
    "g.addV('Park').property('id',
'p1').property('name',
'Yosemite').property('Feature', 'El Capitan'",
    "g.addV('Park').property('id',
'p2').property('name', 'Joshua
Tree').property('Feature', 'Yucca Brevifolia'",
    "g.addV('State').property('id',
's1').property('name',
'California').property('Location', 'USA'",
    "g.addV('Ecosystem').property('id',
'e1').property('name', 'Alpine'",
    "g.addV('Ecosystem').property('id',
'e2').property('name', 'Desert'",
    "g.addV('Ecosystem').property('id',
'e3').property('name', 'High Altitude'"
]

INSERT_NATIONAL_PARK_EDGES = [
    "g.V('p1').addE('is in').to(g.V('s1'))",
    "g.V('p2').addE('is in').to(g.V('s1'))",
    "g.V('p1').addE('has ecosystem
of').to(g.V('e1'))",
    "g.V('p2').addE('has ecosystem
of').to(g.V('e2'))",
    "g.V('p1').addE('has ecosystem
of').to(g.V('e3'))",
    "g.V('p2').addE('has ecosystem
of').to(g.V('e3'))"
]
```



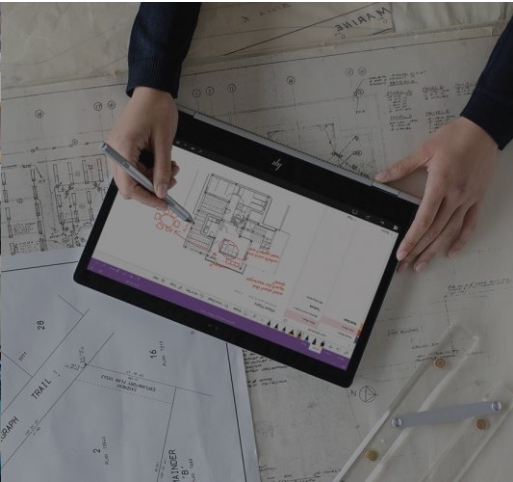
# Review Questions

- Q01 - You are a data engineer wanting to make the data that is currently stored in a Table Storage account located in the West US region available globally. Which Cosmos DB model should you migrate to?
- A01 – Table API
- Q02 - What type of data model provides a traversal language that enables connections and traversals across connected data?
- A02 – Gremlin API



# Lesson 03

## Build a .NET Core App for Azure Cosmos DB in VS Code



# Lesson Objectives

- Create an Azure Cosmos DB account, database, and container in Visual Studio Code using the Azure Cosmos DB extension
- Create an application to store and query data in Azure Cosmos DB
- Use the Terminal in Visual Studio Code to quickly create a console application
- Add Azure Cosmos DB functionality with the help of the Azure Cosmos DB extension for Visual Studio Code

# Creating Azure Cosmos DB in Visual Studio Code

The screenshot shows the Visual Studio Code interface with the Azure extension. The left sidebar displays the 'AZURE' tree view, which is expanded to show the 'COSMOS DB' section. Under 'COSMOS DB', there are several databases: 'adventbikes (MongoDB)', 'ctocdb (SQL)', 'Products', 'Stored Procedures', and 'ToDoList'. The 'Products' database is expanded to show 'Clothing' and 'Documents'. A search box above the tree view contains the text 'Retail'. The main editor area shows a JSON document with the following content:

```
1 {  
2   "productId": "33218896",  
3   "category": "Women's Clothing",  
4   "manufacturer": "Contoso Sport",  
5   "description": "Quick dry crew neck t-shirt",  
6   "price": "14.99",  
7   "shipping": {  
8     "weight": 1,  
9     "dimensions": {  
10      "width": 6,  
11      "height": 8,  
12      "depth": 1  
13    }  
14  },  
15  "_rid": "QS19ALDRxXUBAAAAAAAAAA==",  
16  "_self": "dbs/QS19AA==/colls/QS19ALDRxXU=/documents/1-1",  
17  "_etag": "\"13000b6a-0000-0700-0000-5c9b619c\"",  
18  "_attachments": "attachments/",  
19  "_ts": 1553686941  
20 }  
21
```

# Working with documents programmatically

**CreateDocument  
Async**

**ReadDocument  
Async**

**ReplaceDocument  
Async**

**UpsertDocument  
Async**

**DeleteDocument  
Async**

# Querying document programmatically

```
{
// Set some common query options
FeedOptions queryOptions = new FeedOptions { MaxItemCount = -1,
EnableCrossPartitionQuery = true };

// Here we find nelapin via their LastName
IQueryable<User> userQuery =
this.client.CreateDocumentQuery<User>(
UriFactory.CreateDocumentCollectionUri(databaseName,
collectionName), queryOptions)
.Where(u => u.LastName == "Pindakova");

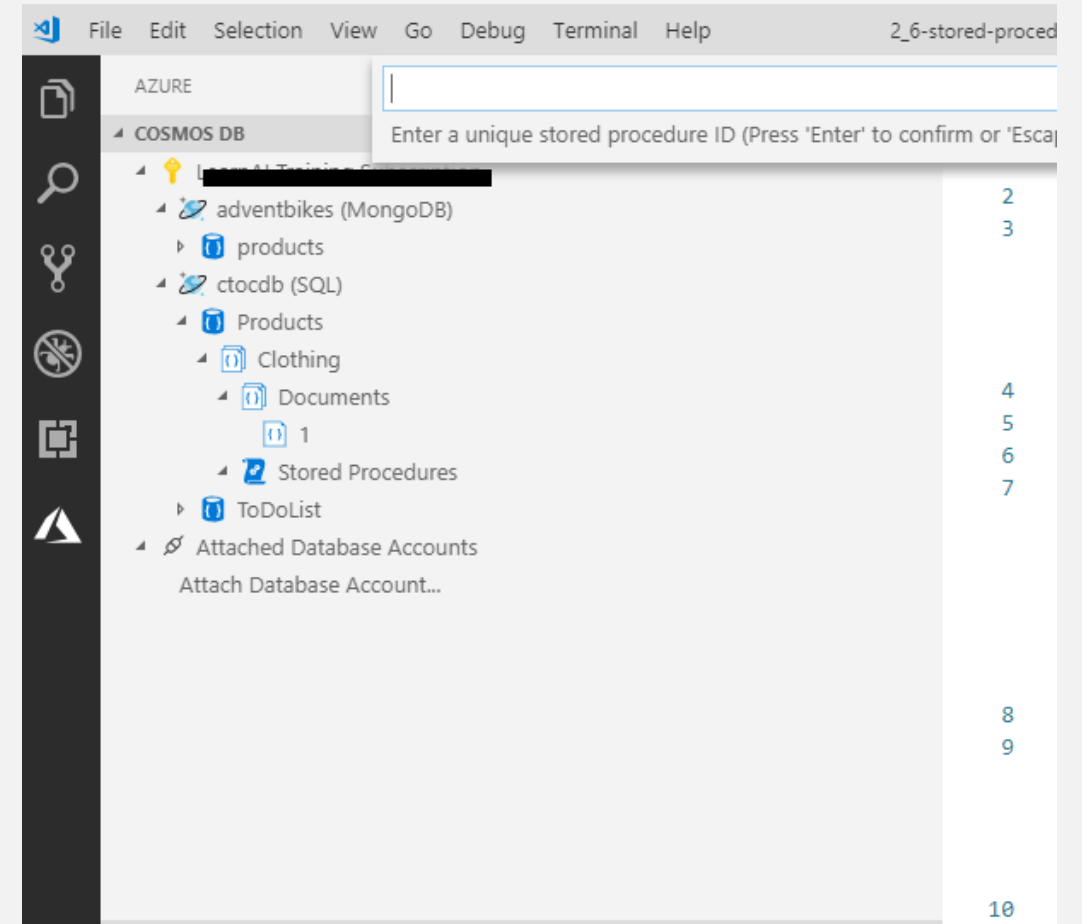
// The query is executed synchronously here, but can also be
executed asynchronously via the IDocumentQuery<T> interface
Console.WriteLine("Running LINQ query...");
foreach (User in userQuery)
{
Console.WriteLine("\tRead {0}", user);
}

// Now execute the same query via direct SQL
IQueryable<User> userQueryInSql =
this.client.CreateDocumentQuery<User>(
UriFactory.CreateDocumentCollectionUri(databaseName,
collectionName),
"SELECT * FROM User WHERE User.lastName = 'Pindakova'",
queryOptions );

Console.WriteLine("Running direct SQL query...");
foreach (User in userQueryInSql)
{
Console.WriteLine("\tRead {0}", user);
}

Console.WriteLine("Press any key to continue ...");
Console.ReadKey();
}
```

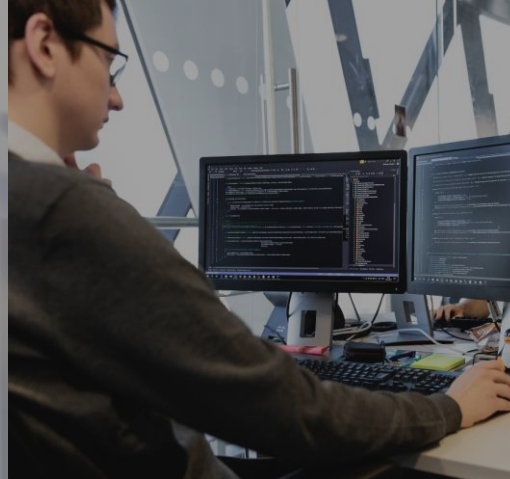
# Perform Azure Cosmos DB Queries.



# Review Questions

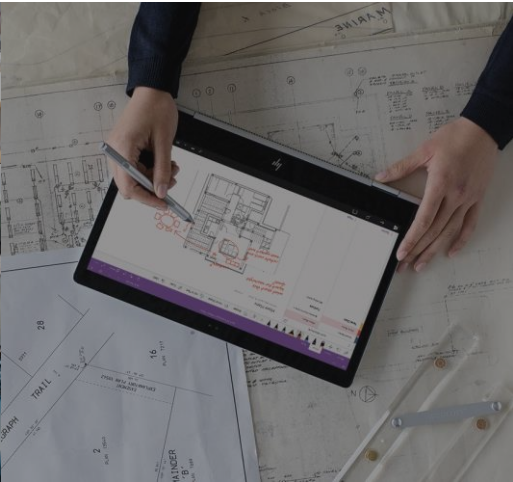
- Q01 – Suppose you are using Visual Studio Code to develop a .NET Core application that accesses Azure Cosmos DB. You need to include the connection string for your database in your application configuration. What is the most convenient way to get this information into your project?
- A01 – Directly from Visual Studio Code
- Q02 - When working with Azure Cosmos DB's SQL API, which of these can be used to perform CRUD operations?
- A02 – LINQ
- Q03 - When working with the Azure Cosmos DB Client SDK's DocumentClient class, you use a NOSQL model. How would you use this class to change the FirstName field of a Person Document from 'Ann' to 'Fran'?
- A03 – Call ReplaceDocumentAsync with an updated Person object





# Lesson 04

## Distribute your data globally with Azure Cosmos DB



# Lesson Objectives





- Learn about the benefits of writing and replicating data to multiple regions around the world
- Cosmos DB multi-master replication
- Cosmos DB failover management
- Change the consistency setting for your database

# Benefits of writing and replicating data to multiple regions

[Home](#) > [Resource groups](#) > [cto\\_rg](#) > [ctocdb](#) > Replicate data globally

## Replicate data globally

ctocdb

 Save  Discard  Manual Failover  Automatic Failover





Click on a location to add or remove regions from your Azure Cosmos DB account.

\* Each region is billable based on the throughput and storage for the account. [Learn more](#)

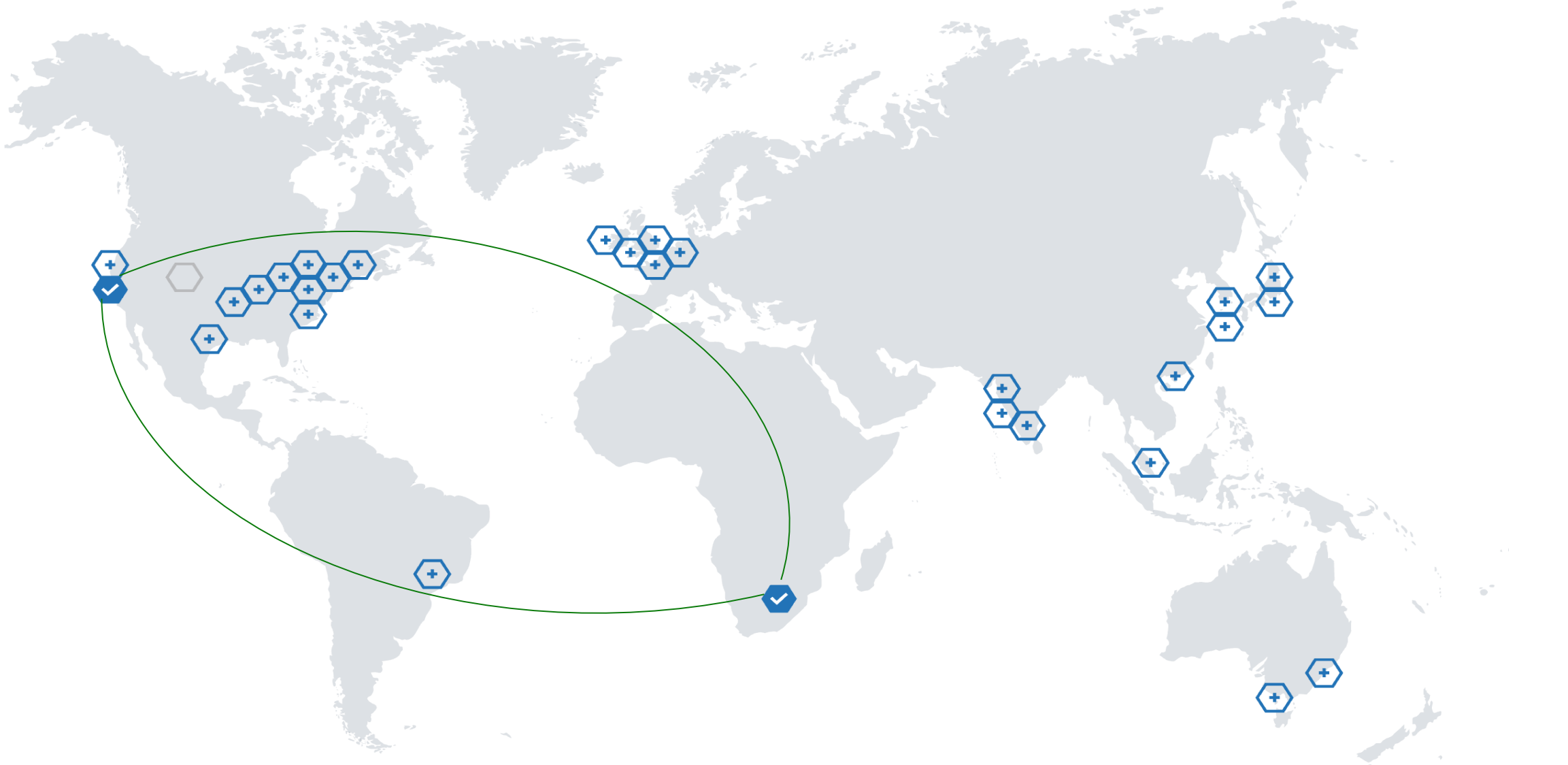


### Configure regions

Configure the regions available for reads and writes. [+ Add region](#)

REGIONS	READS ENABLED	WRITES ENABLED	
West US	✓	✓	
UK South	✓	✓	
Japan West	✓	✓	
South Africa North	✓	✓	

# Cosmos DB multi-master replication



# Cosmos DB failover management

Automated fail-over is a feature that comes into play when there's a disaster or other event that takes one of your read or write regions offline, and it redirects requests from the offline region to the next most prioritized region.

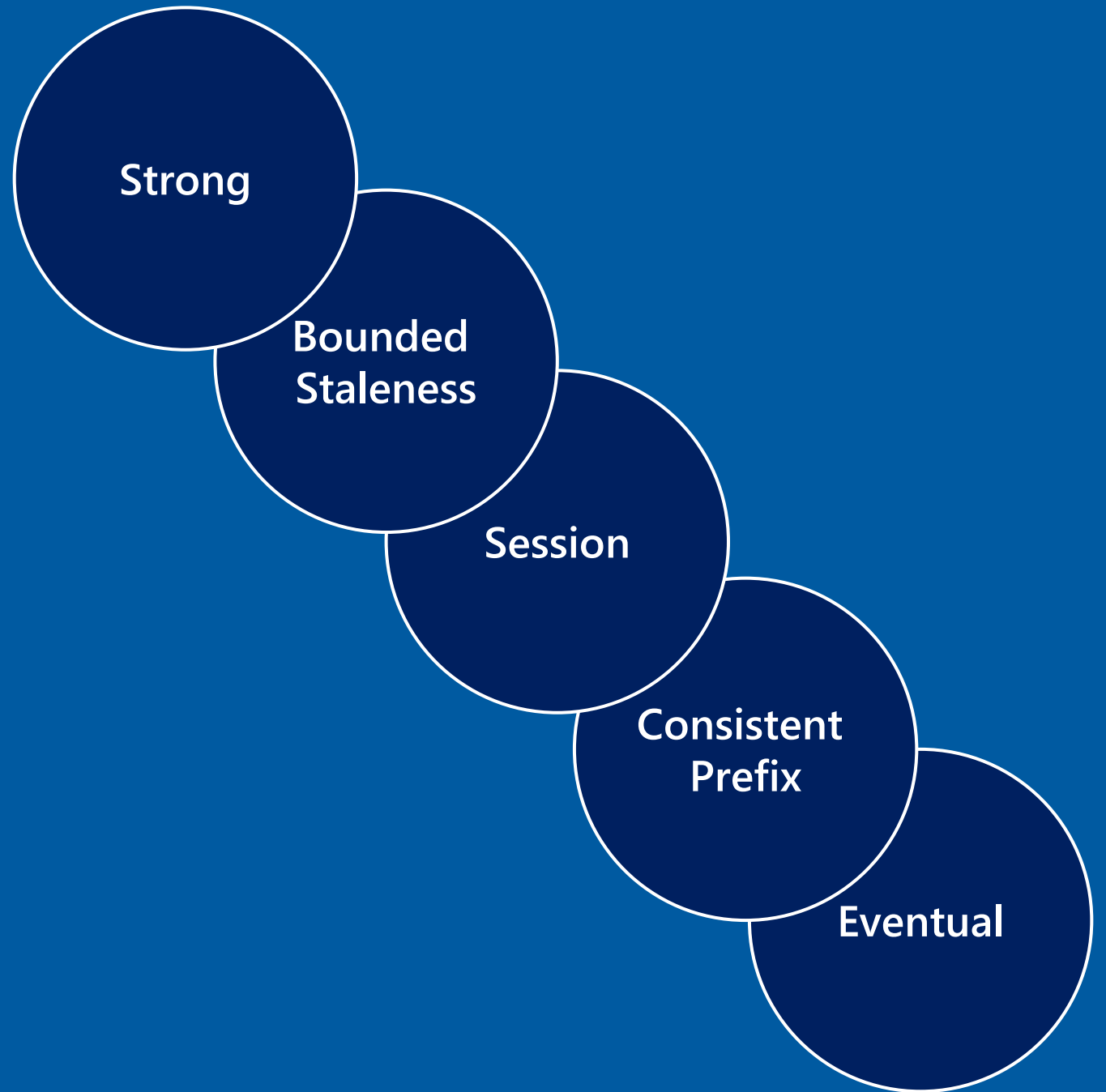
## Read region outage

Azure Cosmos DB accounts with a read region in one of the affected regions are automatically disconnected from their write region and marked offline

## Write region outage

If the affected region is the current write region and automatic fail-over is enabled, then the region is automatically marked as offline. Then, an alternative region is promoted as the write region

# Cosmos DB Consistency Levels



# Review Questions

- Q01 – You want to maximize the data integrity of data that is stored in a Cosmos DB. Which consistency level should you choose?
- A01 – Strong

# Lab: Building Globally Distributed Databases with Cosmos DB





# Lab overview

In this lab, the students will be able to describe and demonstrate the capabilities that Azure Cosmos DB can bring to an organization. They will be able to create a Cosmos DB instance and show how to upload and query data through a portal and through a .Net application. They will then be able to demonstrate how to enable global scale of the Cosmos DB database.

## Lab objectives

After completing this lab, you will be able to:

1. Create an Azure Cosmos DB database built to scale
2. Insert and query data in your Azure Cosmos DB database
3. Distribute your data globally with Azure Cosmos DB
4. (Optional) Build a .NET Core app for Azure Cosmos DB in Visual Studio Code

# Lab scenario

The developers and Information Services department at AdventureWorks are aware that a new service known as Cosmos DB recently released on Azure can provide planetary scale access to data in near real-time. They want to understand the capability that the service can offer and how it can bring value to AdventureWorks, and in what circumstances.

The Information Services department want to understand how the service can be setup and how data can be uploaded. The developers would like to see an example of an application that can be used to upload data to the Cosmos. Both would like to understand how the claim of planetary scale can be met.

At the end of this lab, you will have:

1. Create an Azure Cosmos DB database built to scale
2. Insert and query data in your Azure Cosmos DB database
3. Distribute your data globally with Azure Cosmos DB
4. (Optional) Build a .NET Core app for Azure Cosmos DB in Visual Studio Code

# Lab review

- Exercise 1 – Can you think how Cosmos DB can help your organization?
- Exercise 2 – Name the five programming models that is supported by Cosmos DB?
- Exercise 3 – What NuGet packages is used to interact with Cosmos DB?
- Exercise 4 – How could the distribution of data in Cosmos DB benefit your organization?

# Module Summary >

## In this module, you have learned about:

- Create an Azure Cosmos DB database built to scale.
- Insert and query data in your Azure Cosmos DB database.
- Build a .NET Core app for Azure Cosmos DB in Visual Studio Code.
- Distribute your data globally with Azure Cosmos DB.

## Next steps >

After the course, consider visiting [the Microsoft Cosmos DB Whitepapers site](#) to explore Azure Cosmos DB concepts at a deeper level.

