Microsoft

AZ-204T00A

Learning Path 04: Develop solutions that use Azure Cosmos DB

Agenda

- Explore Azure Cosmos DB
- Work with Azure Cosmos DB

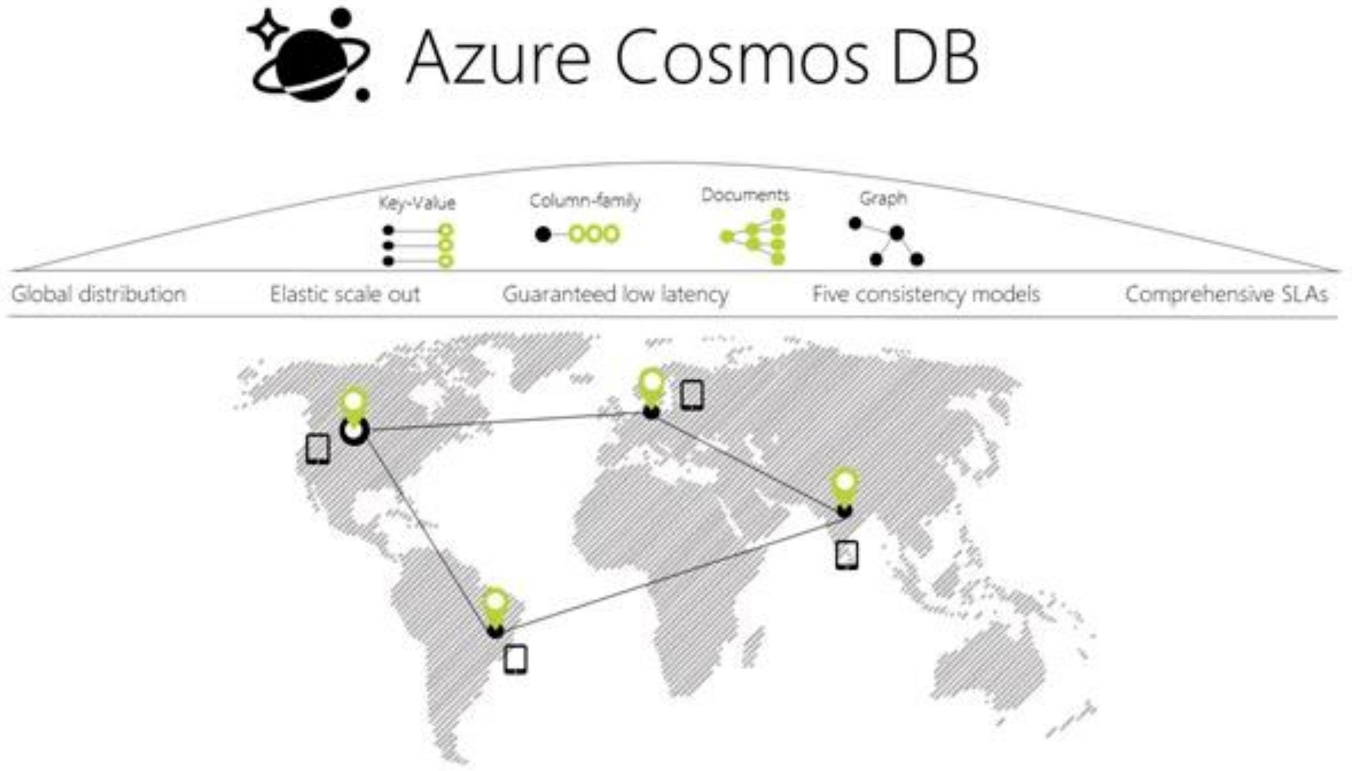
Module 1: Explore Azure Cosmos DB

Learning objectives

- Identify the key benefits provided by Azure Cosmos DB.
- Describe the elements in an Azure Cosmos DB account and how they are organized.
- Explain the different consistency levels and choose the correct one for your project.
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution.
- Describe how request units impact costs.
- Create Azure Cosmos DB resources by using the Azure portal.

Introduction

- Azure Cosmos DB is a fully managed NoSQL database
- Designed to provide low latency, elastic scalability of throughput
- Well-defined semantics for data consistency, and high availability



Identify key benefits of Azure Cosmos DB

- **Global replication:** Automatic and synchronous multi-region replication, supports automatic and manual failover
- **Varied consistency levels:** Offers five consistency models. Provides control over performance-consistency tradeoffs, backed by comprehensive SLAs
- **Low latency:** Serve <10 ms read and <10 ms write requests at the 99th percentile
- **Elastic scale-out:** Elastically scale throughput from 10 to 100s of millions of requests/sec across multiple regions

Explore the resource hierarchy (1 of 2)

Elements in an Azure Cosmos DB account

- The account is the fundamental unit of global distribution and high availability.
- Azure Cosmos DB account contains a unique DNS name

Azure Cosmos DB databases

- You can create one or multiple Azure Cosmos DB databases under your account.
- A database is analogous to a namespace.
- A database is the unit of management for a set of Azure Cosmos DB containers.

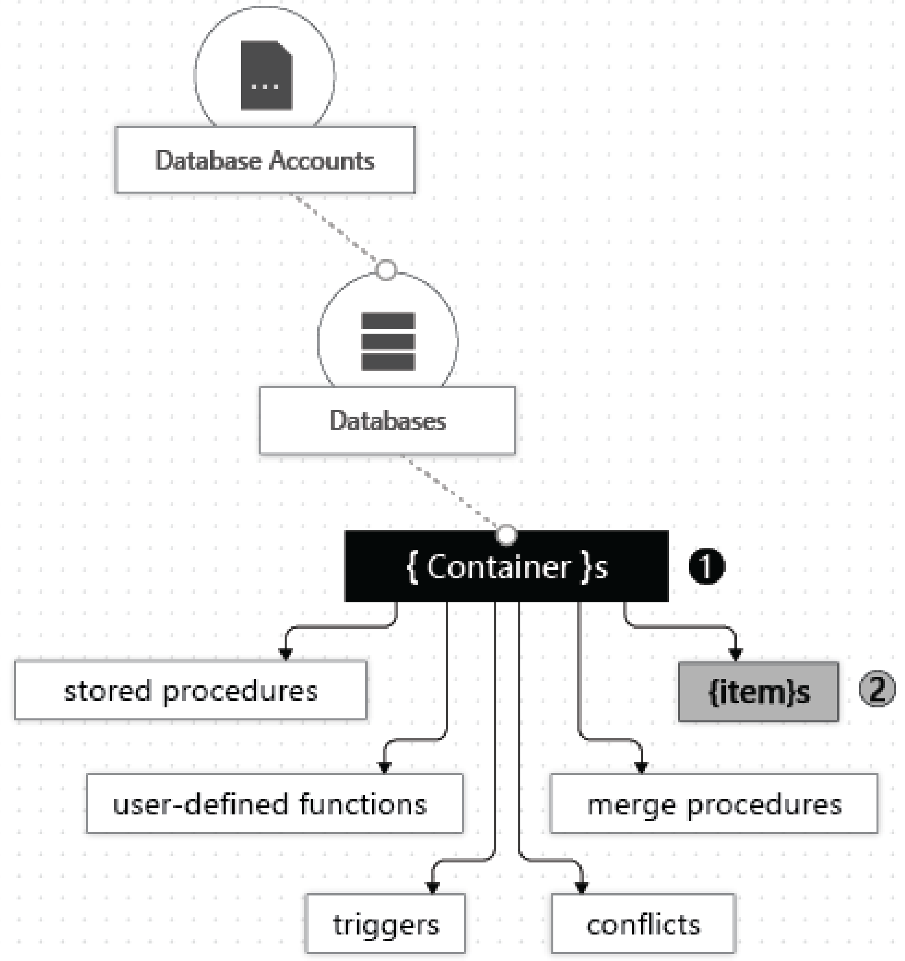
Azure Cosmos DB containers

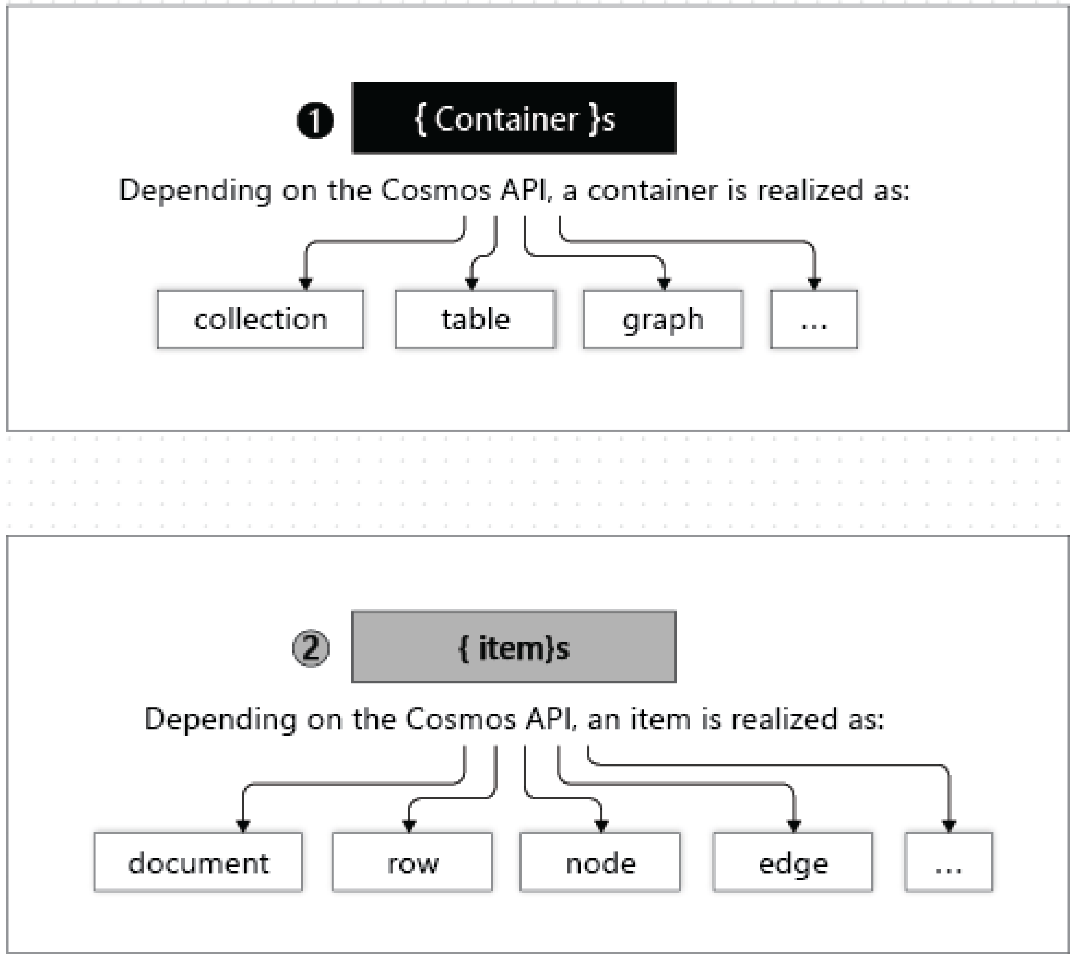
- An Azure Cosmos DB container is the unit of scalability both for provisioned throughput and storage.
- A container is horizontally partitioned and then replicated across multiple regions.

Azure Cosmos DB items

- Depending on which API you use, an Azure Cosmos DB item can represent either a document in a collection, a row in a table, or a node or edge in a graph.

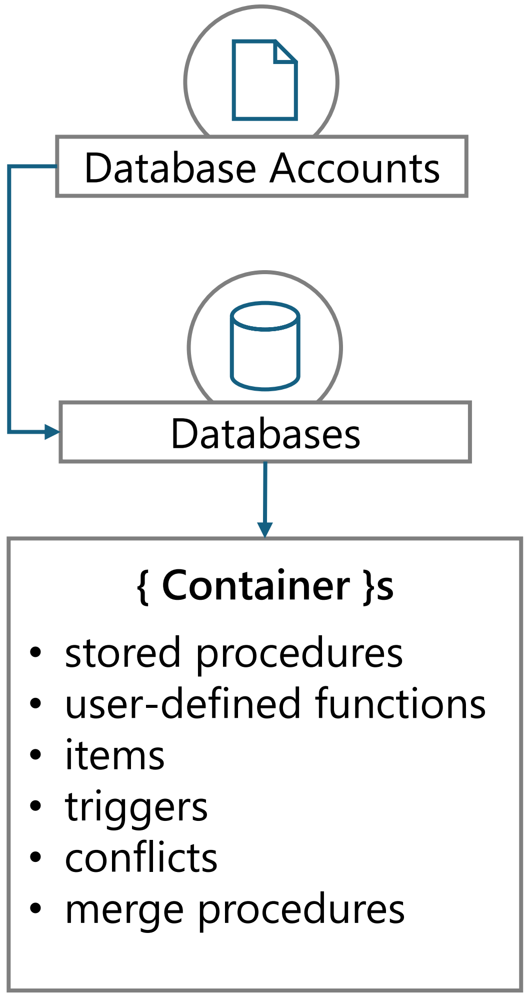
Explore the resource hierarchy (2 of 3)





1
{ Container }s
Depending on the Cosmos API, a container is realized as:
collection
table
graph
...
2
{ item}s
Depending on the Cosmos API, an item is realized as:
document
row
node
edge
...

Explore the resource hierarchy (3 of 3)



{ Container }s

- stored procedures
- user-defined functions
- items
- triggers
- conflicts
- merge procedures

Depending on the Cosmos API, a container is realized as:

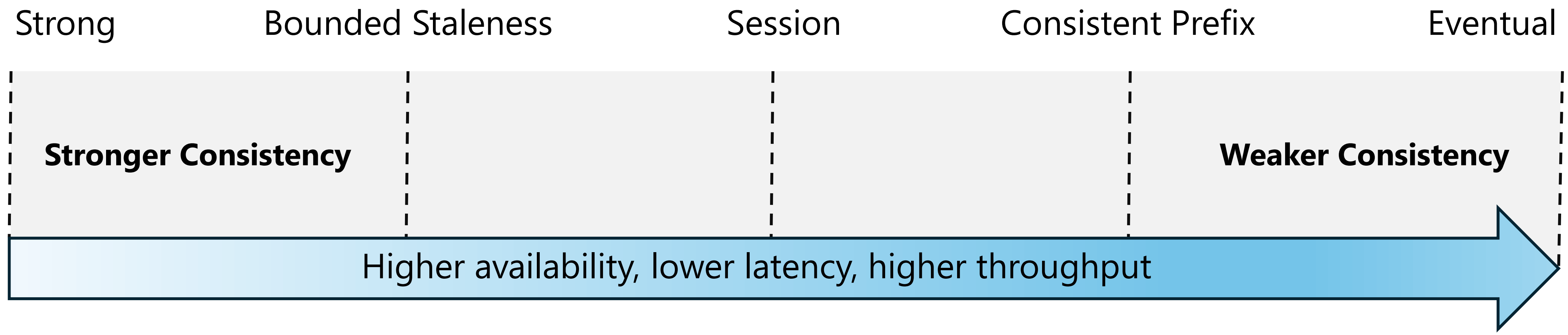
- collection
- table
- graph
- …

Depending on the Cosmos API, an item is realized as:

- document
- row
- node
- edge
- …

Explore consistency levels (1 of 2)

Azure Cosmos DB approaches data consistency as a spectrum of choices instead of two extremes.



Explore consistency levels (2 of 2)

Consistency Level	Description
Strong	When writes are performed on your primary database, the operation is replicated to the replica instances. The write operation is committed (and visible) on the primary only after it has been committed and confirmed by all replicas.
Bounded Staleness	Like the Strong level with the difference that you can configure how stale documents can be within replicas. Staleness is the quantity of time (or the version count) a replica document can be behind the primary document.
Session	This level guarantees that all read and write operations are consistent within a user session. Within the user session, all reads and writes are monotonic and guaranteed to be consistent across primary and replica instances.
Consistent Prefix	This level has loose consistency but guarantees that when updates show up in replicas, they will show up in the correct order (that is, as prefixes of other updates) without any gaps.
Eventual	This level has the loosest consistency and essentially commits any write operation against the primary immediately. Replica transactions are asynchronously handled and will eventually (over time) be consistent with the primary. This tier has the best performance, because the primary database does not need to wait for replicas to commit to finalize its transactions.

Choose the right consistency level

Azure Cosmos DB for NoSQL Azure Cosmos DB for Table	Azure Cosmos DB for Cassandra Azure Cosmos DB for MongoDB Azure Cosmos DB for Apache Gremlin	Consistency guarantees in practice
For many real-world scenarios, session consistency is optimal and it's the recommended option.	Azure Cosmos DB provides native support for wire protocol-compatible APIs for popular databases.	Consistency guarantees for a read operation correspond to the freshness and ordering of the database state that you request.
If your application requires strong consistency, it is recommended that you use bounded staleness consistency level.	These include API for MongoDB, API for Apache Cassandra, and API for Apache Gremlin.	Read-consistency is tied to the ordering and propagation of the write/update operations.
If you need the highest availability and the lowest latency, then use eventual consistency level.	When using API for Apache Gremlin the default consistency level configured on the Azure Cosmos account is used.	The Probabilistically Bounded Staleness metric provides an insight into how often you can get a stronger consistency than the configured level.

Explore supported APIs

- **API for NoSQL:** This API stores data in document format. It offers the best end-to-end experience as we have full control over the interface, service, and the SDK client libraries.
- **API for MongoDB:** This API stores data in a document structure, via BSON format. It is compatible with MongoDB wire protocol.
- **API for Apache Cassandra:** This API stores data in column-oriented schema and is wire protocol compatible with Apache Cassandra.
- **API for Table:** This API stores data in key/value format.
- **API for Apache Gremlin:** This API allows users to make graph queries and stores data as edges and vertices.
- **API for PostgreSQL:** Stores data either on a single node, or distributed in a multi-node configuration

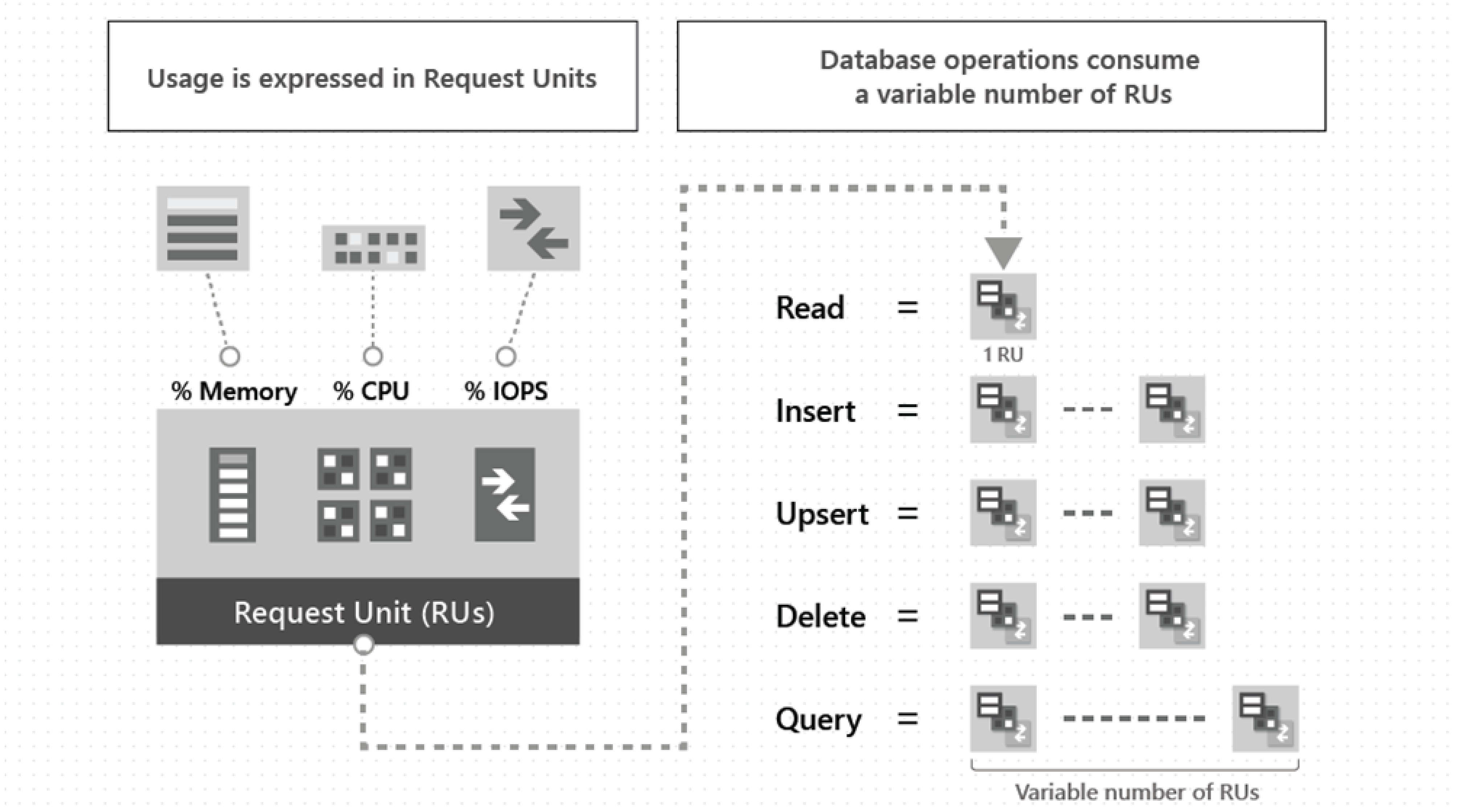
Discover request units

Account types and Request Units

The type of Azure Cosmos DB account created determines the way RUs are charged, there are three modes of account creation:

- **Provisioned throughput:** You manage the number of RUs for your app in 100 RUs per second increments.
- **Serverless:** Billed for the number of RUs consumed by your database operations.
- **Autoscale:** Automatically and instantly scale RUs based on usage.

Discover request units



Exercise: Create Azure Cosmos DB resources by using the Azure portal

In this exercise you learn how to create an Azure Cosmos DB account and add data.

Objectives

- Create an Azure Cosmos DB account
- Add a database and a container
- Add data to your database
- Clean up resources

Summary and knowledge check

In this module, you learned how to:

- Identify the key benefits provided by Azure Cosmos DB.
- Describe the elements in an Azure Cosmos DB account and how they are organized.
- Explain the different consistency levels and choose the correct one for your project.
- Explore the APIs supported in Azure Cosmos DB and choose the appropriate API for your solution.
- Describe how request units impact costs.
- Create Azure Cosmos DB resources by using the Azure portal.

1. What consistency level offers the greatest throughput?

2. What are request units (RUs) in Azure Cosmos DB?

Module 2: Work with Azure Cosmos DB

Learning objectives

- Identify classes and methods used to create resources.
- Create resources by using the Azure Cosmos DB .NET v3 SDK.
- Write stored procedures, triggers, and user-defined functions by using JavaScript.
- Implement change feed notifications

Introduction

- This module focuses on Azure Cosmos DB .NET SDK v3 for API for NoSQL.
- Because Azure Cosmos DB supports multiple API models, version 3 of the .NET SDK uses the generic terms "container" and "item".
- This module also covers JavaScript when discussing stored procedures, triggers, and user-defined functions.

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (1 of 3)

Creating items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);

// create anonymous type in .NET
Product orangeSoda = new Product {
    id = "7cc3212d-0e2c-4a13-b348-f2d879c43342",
    name = "Orange Soda", group = "Beverages",
    diet = false, price = 1.50m, quantity = 2000
};

// Upload item
Product item = await container.CreateItemAsync(orangeSoda);
Product item = await container.UpsertItemAsync(orangeSoda);
```

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (2 of 3)

Reading items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);


// Get unique fields
string id = "7cc3212d-0e2c-4a13-b348-f2d879c43342";
PartitionKey partitionKey = new PartitionKey("Beverages");


// Read item using unique id
ItemResponse<Product> response = await container.ReadItemAsync<Product>(
          id,
          partitionKey );


// Serialize response
Product item = response.Resource;
```

Explore Microsoft .NET SDK v3 for Azure Cosmos DB (2 of 3)

Query items

```
// Get container reference
CosmosClient client = new CosmosClient(endpoint, key);
Container container = client.GetContainer(databaseName, collectionName);

// Use SQL query language
FeedIterator<Product> iterator = container.GetItemQueryIterator<Product>(
    "SELECT * FROM products p WHERE p.diet = false"
);

// Iterate over results
while (iterator.HasMoreResults)
{
    FeedResponse<Product> batch = await iterator.ReadNextAsync();
    foreach(Product item in batch)  { }
}
```

Exercise : Create resources by using the Microsoft .NET SDK v3

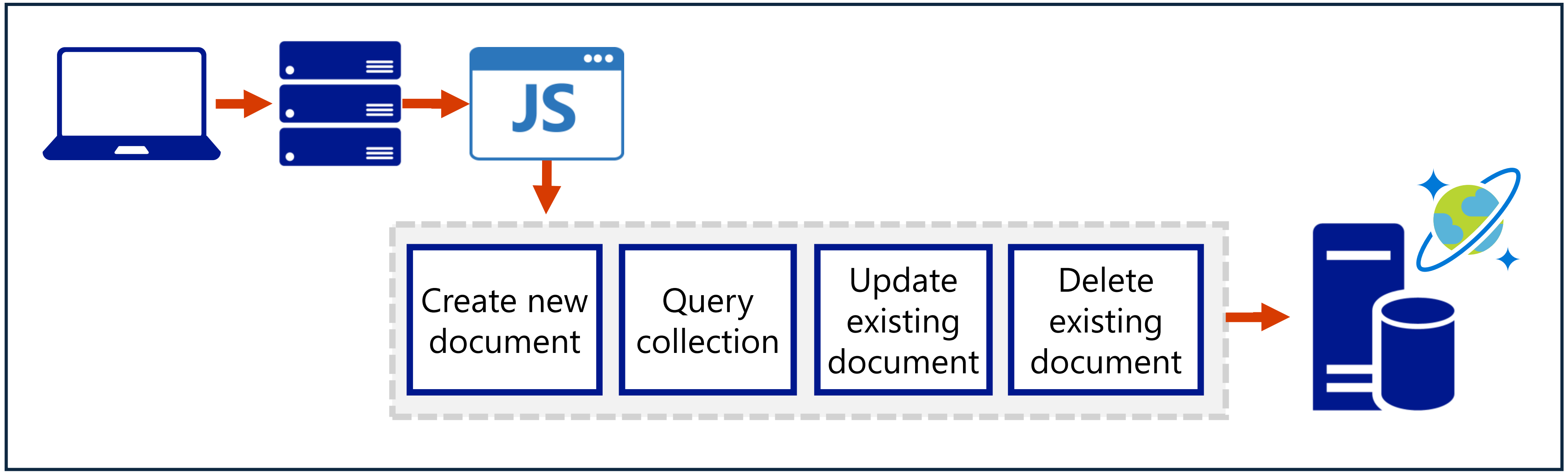
In this exercise you create an Azure Cosmos DB account and then perform operations on the account through a console app.

Objectives

- Create resources in Azure
- Set up and build the console application
- Add code to connect to an Azure Cosmos DB account
- Create a database
- Create a container

Create stored procedures (1 of 4)

- In Azure Cosmos DB, JavaScript is hosted in the same memory space as the database
- Requests made within stored procedures and triggers run in the same scope of a database session



Create stored procedures (2 of 4)

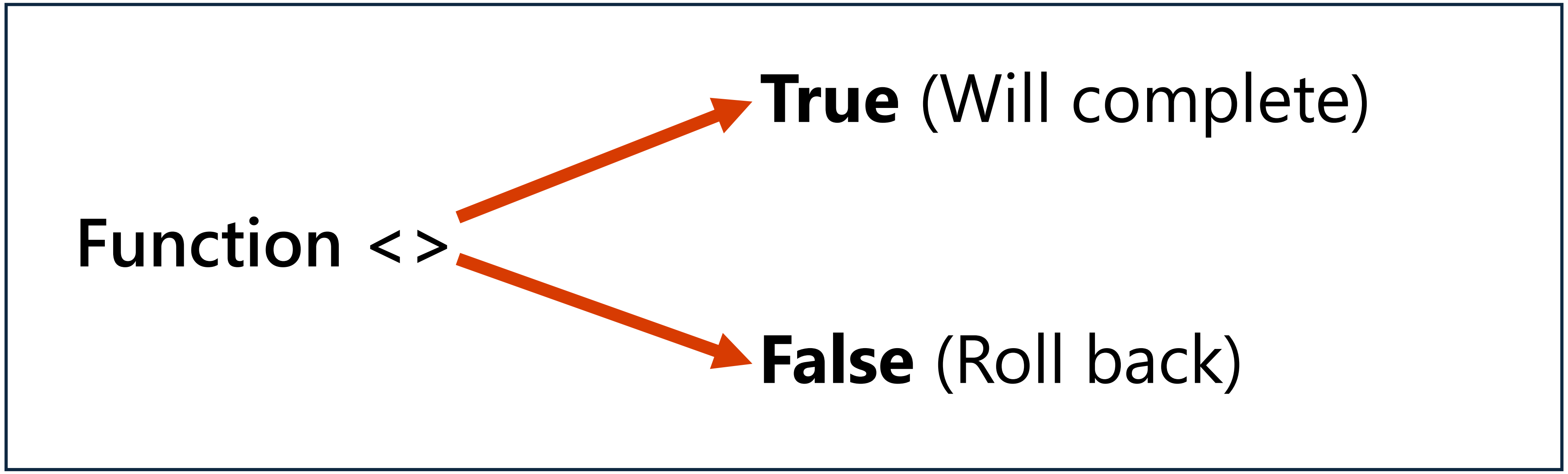
Example code

```
function createSampleDocument(documentToCreate) {
    var context = getContext();
    var collection = context.getCollection();
    var accepted = collection.createDocument(
        collection.getSelfLink(),
        documentToCreate,
        function (error, documentCreated) {
            context.getResponse().setBody(documentCreated.id)
        }
    );
    if (!accepted) return;
}
```

Create stored procedures (3 of 4)

Bounded execution

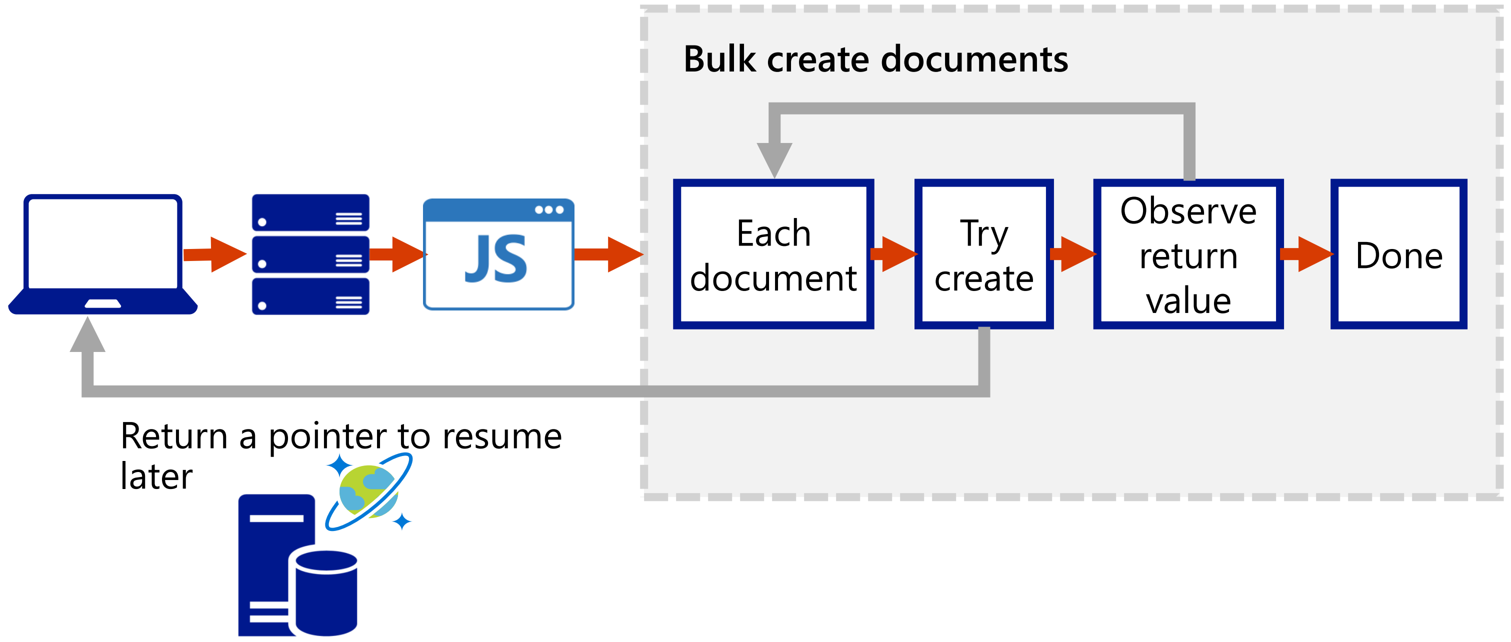
- All Azure Cosmos DB operations must complete within a limited amount of time



Create stored procedures (4 of 4)

Transaction continuation

- JavaScript functions can implement a continuation-based model to batch or resume execution
  - The continuation value can be any value of your choice
  - Your applications can then use this value to resume a transaction from a new starting point



Create triggers and user-defined functions (1 of 2)

- **Pre-triggers:** Executed before modifying, or creating, a database item.
  - Can be used to validate the properties of an Azure Cosmos item that is being created, for example.
  - Pre-triggers cannot have any input parameters. The request object in the trigger is used to manipulate the request message associated with the operation.
- **Post-triggers:** Executed after modifying a database item
  - Can be used to query for the metadata item and updates it with details about the newly created item.
- **User-defined function:** User-defined functions extend the Azure Cosmos DB SQL API's query language grammar and implement custom business logic.
  - Can only be called from inside queries. They don't have access to the context object and are meant to be used as compute-only code.

Create triggers and user-defined functions (2 of 2)

Pre-trigger example code

```javascript
function validateToDoItemTimestamp() {
    var context = getContext();
    var request = context.getRequest();

    // item to be created in the current operation
    var itemToCreate = request.getBody();

    // validate properties
    if (!("timestamp" in itemToCreate)) {
        var ts = new Date();
        itemToCreate["timestamp"] = ts.getTime();
    }

    // update the item that will be created
    request.setBody(itemToCreate);
}
```

Explore change feed in Azure Cosmos DB (1 of 3)

- Change feed in Azure Cosmos DB is a persistent record of changes to a container in the order they occur.
- Work with Azure Cosmos DB change feed using either a push model or a pull model.
- Recommended to use the push model – no need to worry about polling the change feed for future changes.
- Two ways to read from the change feed with a push model:
  - Azure Functions Azure Cosmos DB triggers
  - The change feed processor library

Explore change feed in Azure Cosmos DB (2 of 3)

- Azure Functions
  - Automatically triggered on each new event in your Azure Cosmos DB container's change feed.
  - With the *Azure Functions trigger for Azure Cosmos DB*, you can use the Change Feed Processor's scaling and reliable event detection functionality without the need to maintain any worker infrastructure.
- Change feed processor
  - Part of the Azure Cosmos DB .NET V3 and Java V4 SDKs.
  - Simplifies the process of reading the change feed and distributes the event processing across multiple consumers effectively.
  - Has four main components: the monitored container, the lease container, the compute instance, and the delegate.

Explore change feed in Azure Cosmos DB (3 of 3)

Change feed processor example code

```
private static async Task<ChangeFeedProcessor> StartChangeFeedProcessorAsync(
    CosmosClient cosmosClient,
    IConfiguration configuration)
{
    string databaseName = configuration["SourceDatabaseName"];
    string sourceContainerName = configuration["SourceContainerName"];
    string leaseContainerName = configuration["LeasesContainerName"];

    Container leaseContainer = cosmosClient.GetContainer(databaseName, leaseContainerName);
    ChangeFeedProcessor changeFeedProcessor = cosmosClient.GetContainer(databaseName, sourceContainerName)
        .GetChangeFeedProcessorBuilder<ToDoItem>(processorName: "changeFeedSample", onChangesDelegate:
         HandleChangesAsync)
            .WithInstanceName("consoleHost")
            .WithLeaseContainer(leaseContainer)
            .Build();

    Console.WriteLine("Starting Change Feed Processor...");
    await changeFeedProcessor.StartAsync();
    Console.WriteLine("Change Feed Processor started.");
    return changeFeedProcessor;
}
```

Summary and knowledge check

In this module, you learned how to:

- Identify classes and methods used to create resources
- Create resources by using the Azure Cosmos DB .NET v3 SDK
- Write stored procedures, triggers, and user-defined functions by using JavaScript
- Implement change feed notifications

1. When defining a stored procedure in the Azure portal input parameters are always sent as what type to the stored procedure?
2. What would one use to validate properties of an item being created?

Discussion and lab

Group discussion questions

- What are the benefits of using Azure Cosmos DB? What types of apps would benefit most?
- Describe the five consistency levels. Can you give an example of an application that matches the characteristics of each consistency level?

Lab 04: Construct a polyglot data solution

In this lab, you will create an Azure Cosmos DB resource and a storage account resource. Using C# and .NET, you will access the Cosmos DB resource and upload data into it.

Additionally, as Contoso may want to access the data in Cosmos DB through a user-friendly interface, you will implement a .NET solution that accesses and displays the data from Cosmos DB in a web browser.

Finally, you will set the consistency level for your Cosmos DB instance and implement an Azure function for change feed notifications.

http://aka.ms/az204labs

- Exercise 1: Creating data store resources in Azure
- Exercise 2: Review and upload data
- Exercise 3: Configure a .NET web application

End of presentation