

Design and implement a replication strategy for Azure Cosmos DB for NoSQL

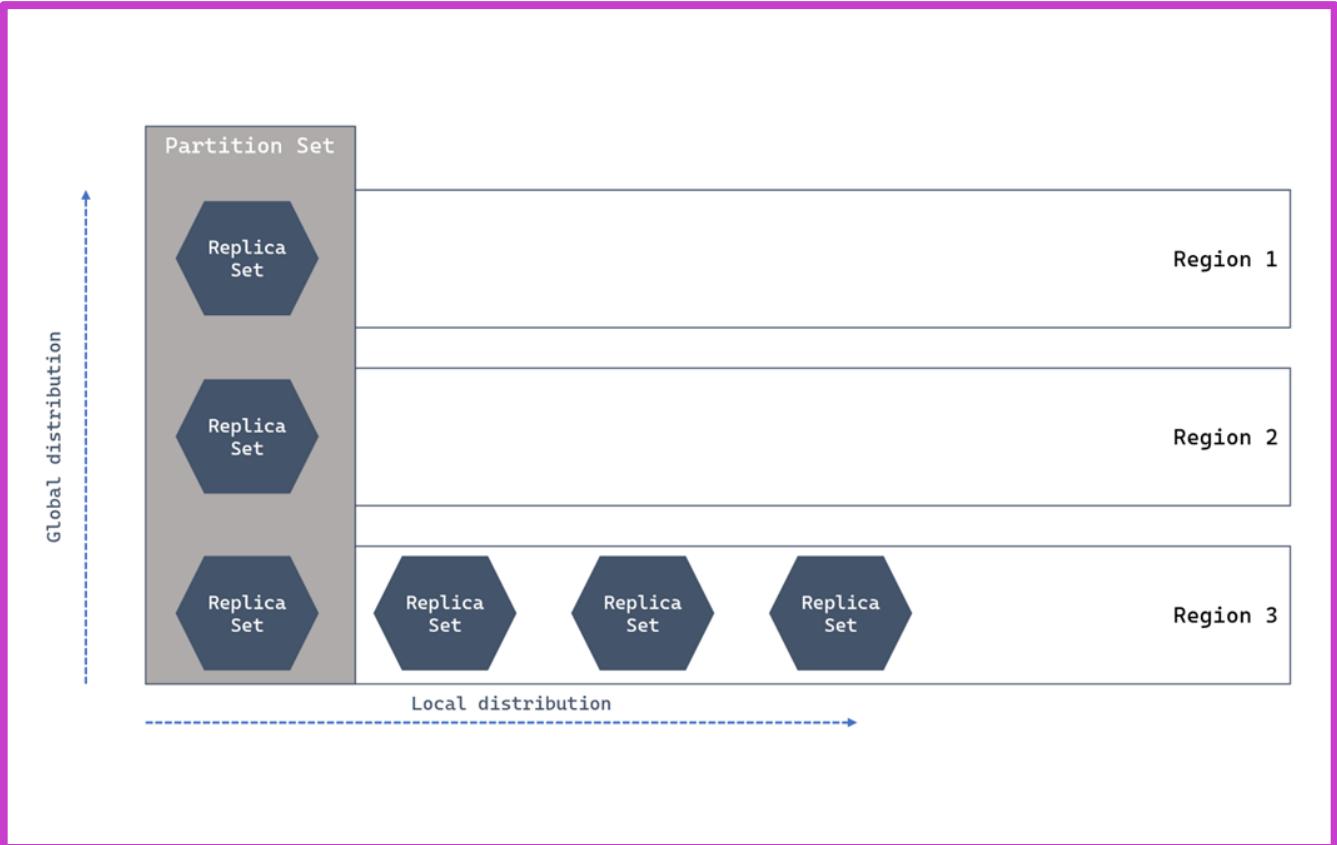
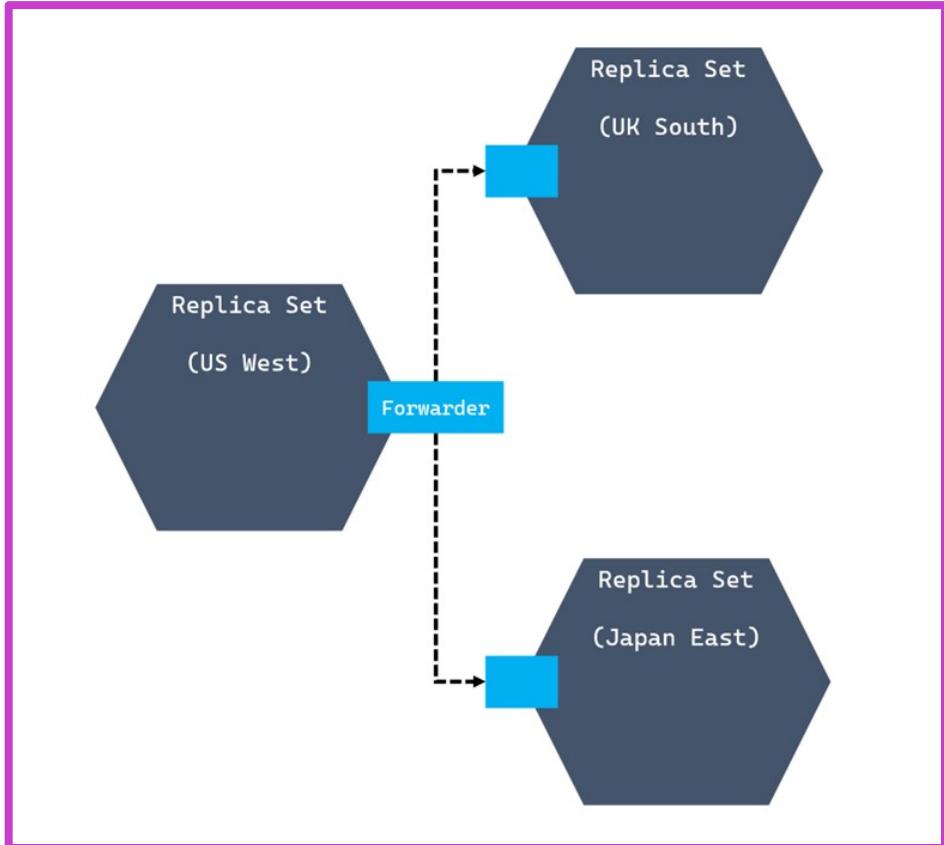


Agenda

- Configure replication and manage failovers in Azure Cosmos DB
- Use consistency models in Azure Cosmos DB for NoSQL
- Configure multi-region write in Azure Cosmos DB for NoSQL

Configure replication and manage failovers in Azure Cosmos DB

Understand replication



Distribute data across regions

Configuring global distribution in Azure Cosmos DB is a turnkey operation that is performed when an account is created or afterward.

Configuring geo-redundancy for a new account

The screenshot shows the 'Create Azure Cosmos DB Account - Azure Cosmos DB for NoSQL' wizard. The 'Global Distribution' tab is selected. Under 'Geo-Redundancy', the 'Enable' radio button is selected. Under 'Multi-region Writes', the 'Disable' radio button is selected. Under 'Availability Zones', the 'Disable' radio button is selected. At the bottom, there are 'Review + create', 'Previous', and 'Next: Networking' buttons.

Configuring geo-redundancy for an existing account

The screenshot shows the 'osdiufiosdaosdo | Replicate data globally' page for an existing Azure Cosmos DB account. On the left, a sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Cost Management', 'Quick start', 'Notifications', 'Data Explorer', 'Settings', 'Features' (which has 'Replicate data globally' highlighted with a red box), 'Default consistency', and 'Backup & Restore'. On the right, a world map shows various regions with icons indicating their status. A sidebar on the right also shows 'Configure regions' (with 'Multi-region writes' set to 'Disable') and 'Configure the regions for reads, writes and leases' sections for 'Write Region' (West US 2) and 'Read Regions' (West Central US).

The cost of distributing data globally is **RU/s x # of regions**.

Azure Cosmos DB failovers

An automatic failover plan can transfer the write region to one of the read regions in the case of an outage.

Define automatic failover policies

Automatic Failover ...

Enable Automatic Failover ⓘ

ON OFF

Drag-and-drop read regions items to reorder the failover priorities.

Tip: Drag ⓘ on the left of the hovered row to reorder the list.

Write Region

West US 2

Read Regions	Priorities
West Central US	1
UK South	2
South Africa North	3

Perform manual failovers

Manual Failover ...

Select a Read Region to become the new Write Region.

Tip: Identify all dependent services leveraging this account and ensure that triggering a failover will not jeopardize your production application.

Write Region

West US 2

Read Regions

West Central US

Configure SDK region

Use the *ApplicationRegion* or *ApplicationPreferredRegions* properties to configure preferred regions.

Setting an application region

```
CosmosClientOptions options = new () { ApplicationRegion = Regions.UKSouth };
CosmosClient client = new (connectionString, options);

// Or using the CosmosClientBuilder class
CosmosClient client = new CosmosClientBuilder(connectionString)
    .WithApplicationRegion(Regions.UKSouth)
    .Build();
```

Setting a list of preferred application regions

```
List<string> regions = new() { "East Asia", "South Africa North", "West US" };
CosmosClientOptions options = new () { ApplicationPreferredRegions = regions };
CosmosClient client = new (connectionString, options);

// Or using the CosmosClientBuilder class
CosmosClient client = new CosmosClientBuilder(connectionString)
    .WithApplicationPreferredRegions( new List<string> { Regions.EastAsia, Regions.SouthAfricaNorth,
Regions.WestUS } ) .Build();
```

Lab – Connect to different regions with the Azure Cosmos DB for NoSQL SDK

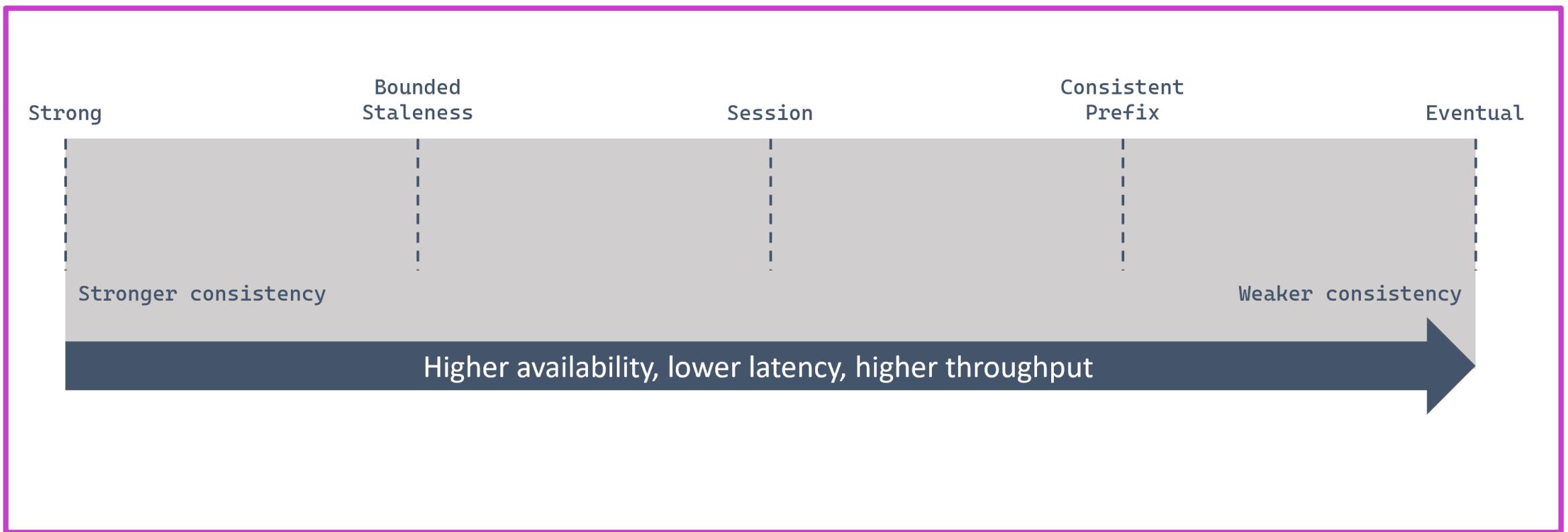


- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- Connect to the Azure Cosmos DB for NoSQL account from the SDK
- Configure the .NET SDK with a preferred region list

Use consistency models in Azure Cosmos DB for NoSQL

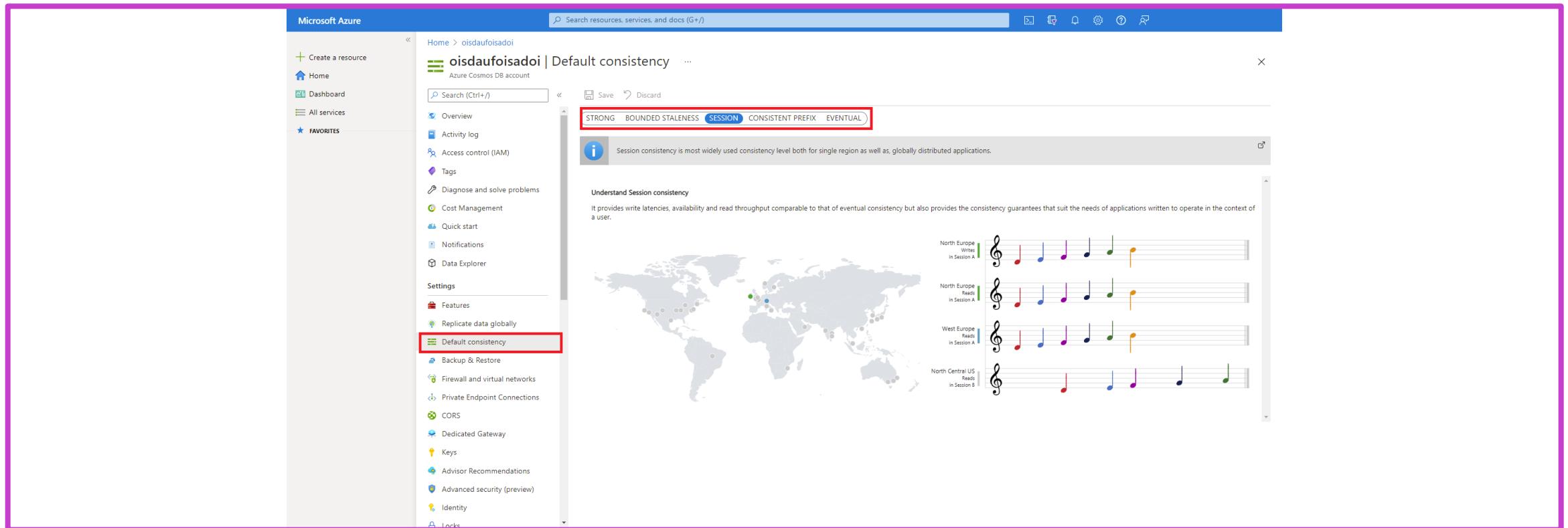
Understand consistency models

In a distributed database system, tradeoffs are often made between highly consistent data with extended latency and speedy data operations that may not be consistent immediately.



Configure default consistency model in the portal

In the Azure portal, the Default consistency pane is used to configure a new default consistency level for the entire account.



Change consistency model with the SDK

Using the *ItemRequestOptions* class, you can relax the current default consistency level to a weaker one.

```
// Set the item request Consistency Level option

ItemRequestOptions options = new()
{
    ConsistencyLevel = ConsistencyLevel.Eventual
};

// Assign the option to the create item operation
var item = new Product
{
    id = $"{Guid.NewGuid()}",
    categoryId = $"{Guid.NewGuid()}",
    name = "Reflective Handlebar"
};

await container.CreateItemAsync<Product>(item, requestOptions: options);
```

Use session tokens

Using the .NET SDK classes, the session token can be manually extracted and passed back to the Azure Cosmos DB resource.

```
// Create an item with an Item Response
ItemResponse<Product> response = await container.CreateItemAsync<Product>(item);

// Get the session token from the item response
string token = response.Headers.Session;

// Set the item request option session token to the previous token
ItemRequestOptions options = new()
{
    SessionToken = token
};

// Use the token on the new request
ItemResponse<Product> readResponse = container.ReadItemAsync<Product>(id, partitionKey,
requestOptions: options);
```

Lab – Configure consistency models in the portal and the Azure Cosmos DB for NoSQL SDK



- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- Connect to the Azure Cosmos DB for NoSQL account from the SDK
- Configure consistency level for a point operation

Configure multi-region write in Azure Cosmos DB for NoSQL

Understand multi-region write

With Azure Cosmos DB, every region supports both writes and reads.

The screenshot shows the Azure Cosmos DB account settings page. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Cost Management, Quick start, Notifications, Data Explorer, Settings, Features (which has 'Replicate data globally' highlighted with a red box), Default consistency, and Backup & Restore. The main area features a world map with numerous blue hexagonal icons representing data locations. A callout box highlights the 'Configure regions' section, which includes a 'Multi-region writes' toggle switch set to 'Enable' (also highlighted with a red box). Below this, a table lists five regions: North Europe, Japan East, Australia East, East US 2, and South Africa North, all with 'Reads Enabled' and 'Writes Enabled' checked. A note at the bottom says 'Configure the regions for reads, writes and availability zone (supported in selected regions and can only be configured when a new region is added). + Add region'.

Regions	Reads Enabled	Writes Enabled	Availability Zone
North Europe	✓	✓	
Japan East	✓	✓	
Australia East	✓	✓	
East US 2	✓	✓	
South Africa North	✓	✓	

Understand conflict resolution policies

Azure Cosmos DB's multi-region write feature has automatic conflict management built in. The default policy is known as Last Write Wins that uses the `_ts` property by default.

Replace the default `_ts` property configuring any numeric property as a *conflict resolution path* on the .NET SDK

```
Database database = client.GetDatabase("cosmicworks");

// Define a default custom conflict resolution path as /metadata/sortableTimestamp.
ContainerProperties properties = new("products", "/categoryId")
{
    ConflictResolutionPolicy = new ConflictResolutionPolicy()
    {
        Mode = ConflictResolutionMode.LastWriterWins,
        ResolutionPath = "/metadata/sortableTimestamp",
    }
};

// Note: You can only set a conflict resolution policy on newly created containers.
Container container = database.CreateContainerIfNotExistsAsync(properties);
```

Create a custom conflict resolution policy step 1: JavaScript SP

Create a custom conflict resolution policy when you wish to write your own logic to resolve conflicts between items. Let's first define the custom JavaScript SP that resolves the conflicts.

```
function <function-name>(incomingItem, existingItem, isTombstone, conflictingItems)
```

Custom conflict resolution JavaScript stored procedure example that uses the `/metadata/sortableTimestamp` path.

```
function resolveConflicts(incomingItem, existingItem, isTombstone, conflictingItems)
{
    if (!incomingItem) { if (existingItem) { __.deleteDocument(existingItem._self, {}); } }
    else if (isTombstone) { }
    else { if (existingItem) { if (incomingItem.metadata.sortableTimestamp > existingItem.metadata.sortableTimestamp) { return; } }
        var i;
        for (i = 0; i < conflictingItems.length; i++)
        { if (incomingItem.metadata.sortableTimestamp > conflictingItems[i].metadata.sortableTimestamp) { return; } }
        delete (conflictingItems, incomingItem, existingItem);
    }
    function delete (documents, incoming, existing)
    {
        if (documents.length > 0)
        { __.deleteDocument(documents[0]._self, {}, function (err, responseOptions) { documents.shift();
            delete (documents, incoming, existing); });
        else if (existing)
        { __.replaceDocument(existing._self, incoming); }
        else { __.createDocument(collection.getSelfLink(), incoming); }
    }
}
```

Create a custom conflict resolution policy step 2: .NET SDK

Once the JavaScript SP code has been defined, let's assume it was written to the *resolver.js* file.

```
Database database = client.GetDatabase("cosmicworks");

// Define the custom conflict resolution path as /metadata/sortableTimestamp.
ContainerProperties properties = new("products", "/categoryId")
{
    ConflictResolutionPolicy = new ConflictResolutionPolicy()
    {
        Mode = ConflictResolutionMode.Custom,
        ResolutionProcedure = string.Format("dbs/{0}/colls/{1}/sprocs/{2}",
                                              this.databaseName,
                                              this.udpCollectionName,
                                              "resolver")
    }
};

// Note: You can only set a conflict resolution policy on newly created containers.
Container container = database.CreateContainerIfNotExistsAsync(properties);

StoredProcedure ResolverSP = container.Scripts.CreateStoredProcedureAsync(
    new StoredProcedureProperties("resolver", File.ReadAllText(@"resolver.js")))
);
```

Lab – Connect to a multi-region write account with the Azure Cosmos DB for NoSQL SDK



- Prepare your development environment
- Create an Azure Cosmos DB for NoSQL account
- Connect to the Azure Cosmos DB for NoSQL account from the SDK
- Configure write region for the SDK

