# Challenge 5: It's All About the Payload, The Sequel

In this exercise ,you will understand the steps involved in making new data type for LLM interaction. You will learn how new data type gets added ,new fields, new filed types and what are the different place code changes required to support this data type.’

We use

* Semantic kernel
* Cognitive Search vector
* Change Feed in CosmosDB

**process**

Understanding the process of creating a new data type for embedding ,storing it in Cognitive search.

* Create model class
* Update ModelRegistry
* Apply Breakpoint in Cosmos DB change feed processor to study the flow.
* Add a new data type in Cosmos DB
* Create container and add documents
* Line by line debugging.

Objective :

* Understand the steps involved in making new data type ready for LLM.

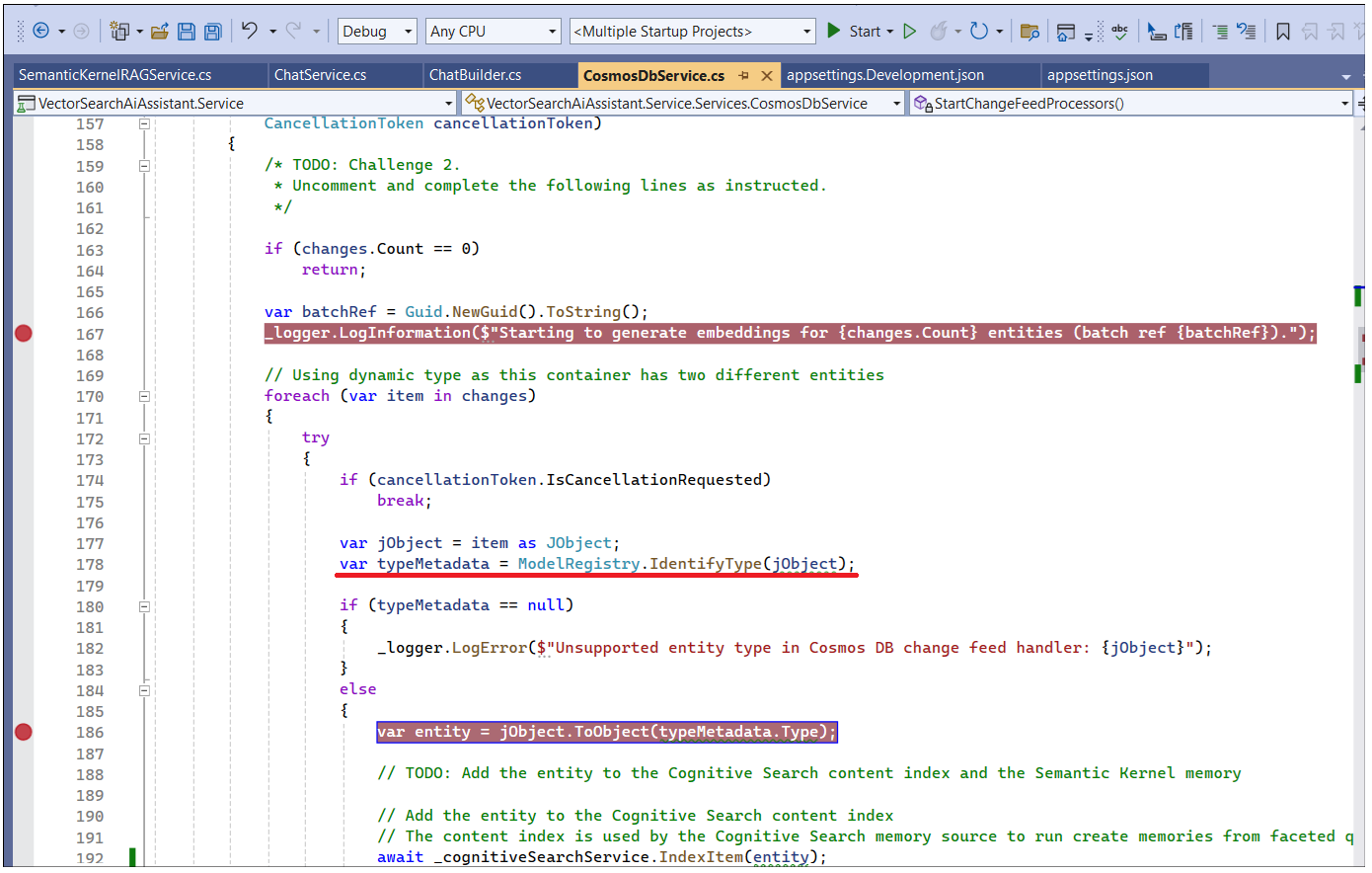
## **Task 1 : Apply Breakpoint in Cosmos DB change feed processor to study the flow**

1. Switch back to Visual Studio, open **CosmosDBServices.cs** under **VectorSearchAiAssistant.Service-> Services.** Put breakpoints at line #167 and 186.

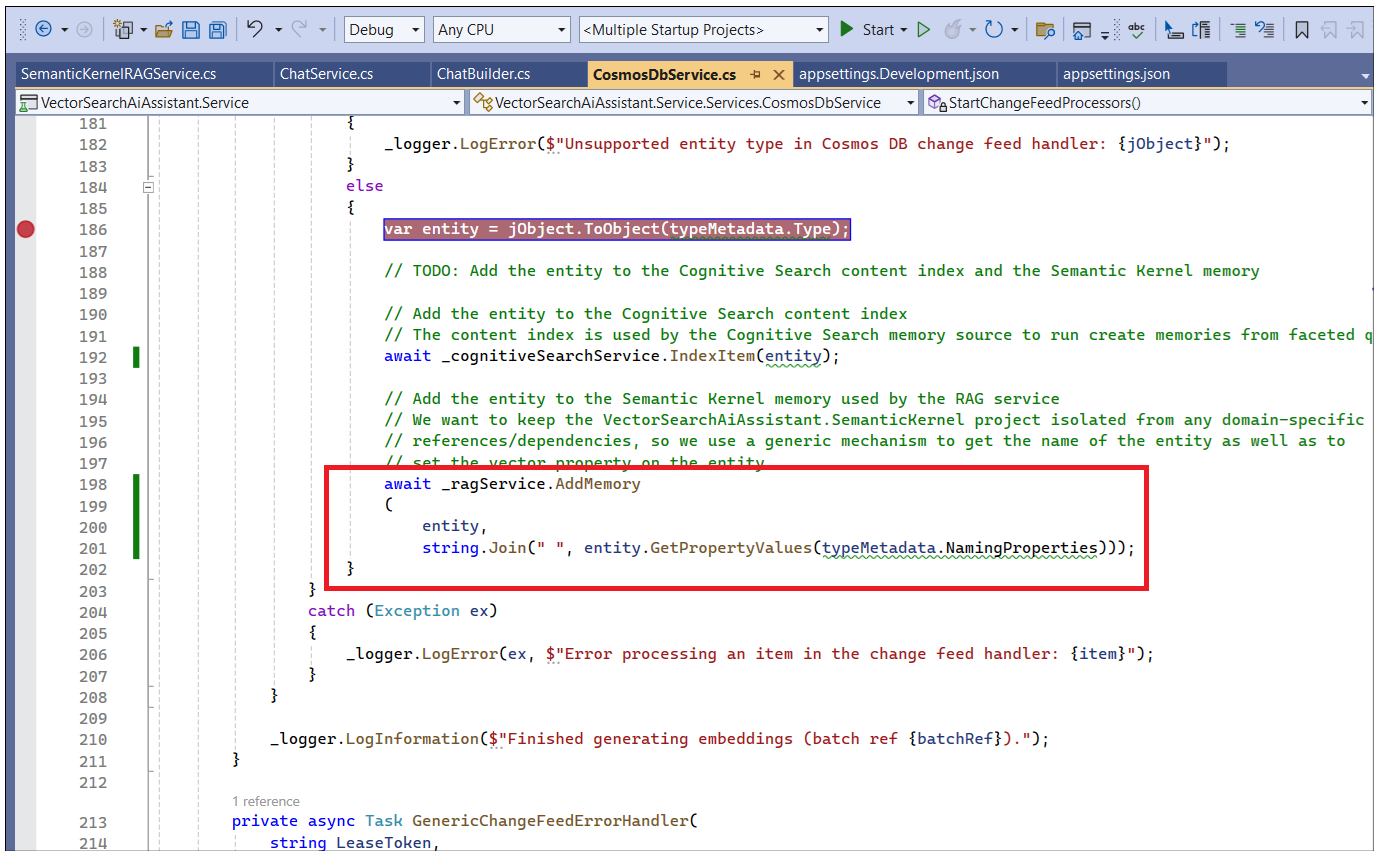
A computer screen shot of a computer screen

Description automatically generated

1. At line number 178, Change feed gets the document and that document passed to ModelRegistry to identify the type (IdentifyType(jObject,) of document and gets its Meatadata(typeMetadata)

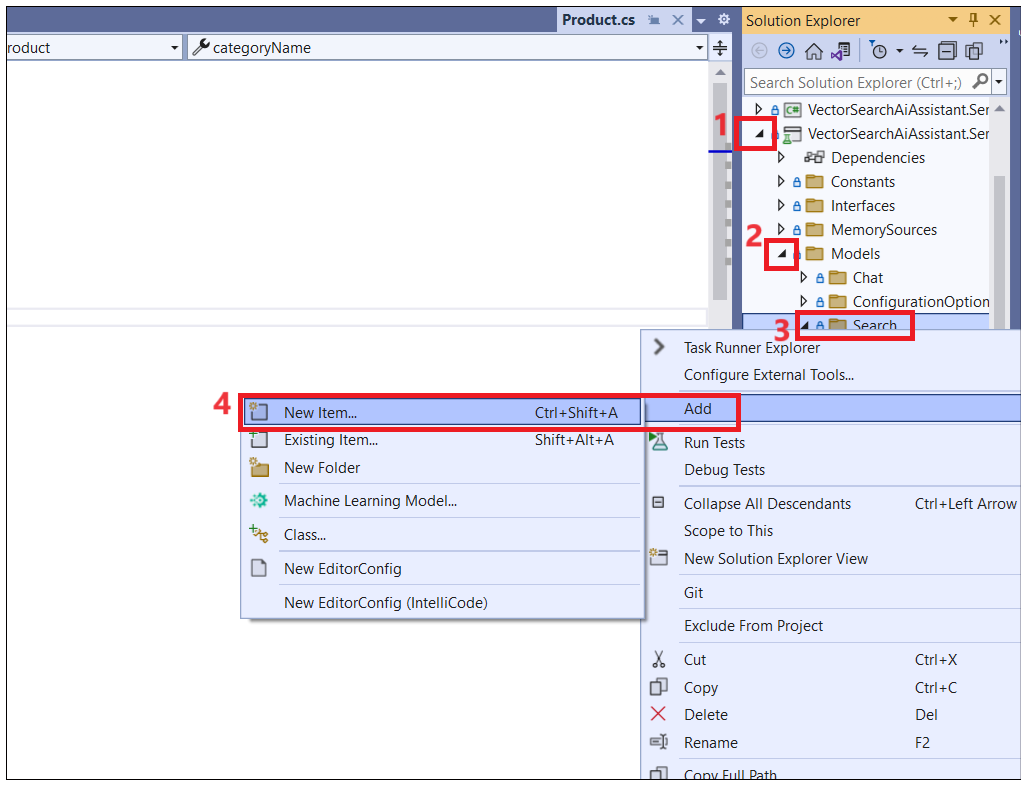


1. Based on the metada,RAG service use properties to generate embeddings.

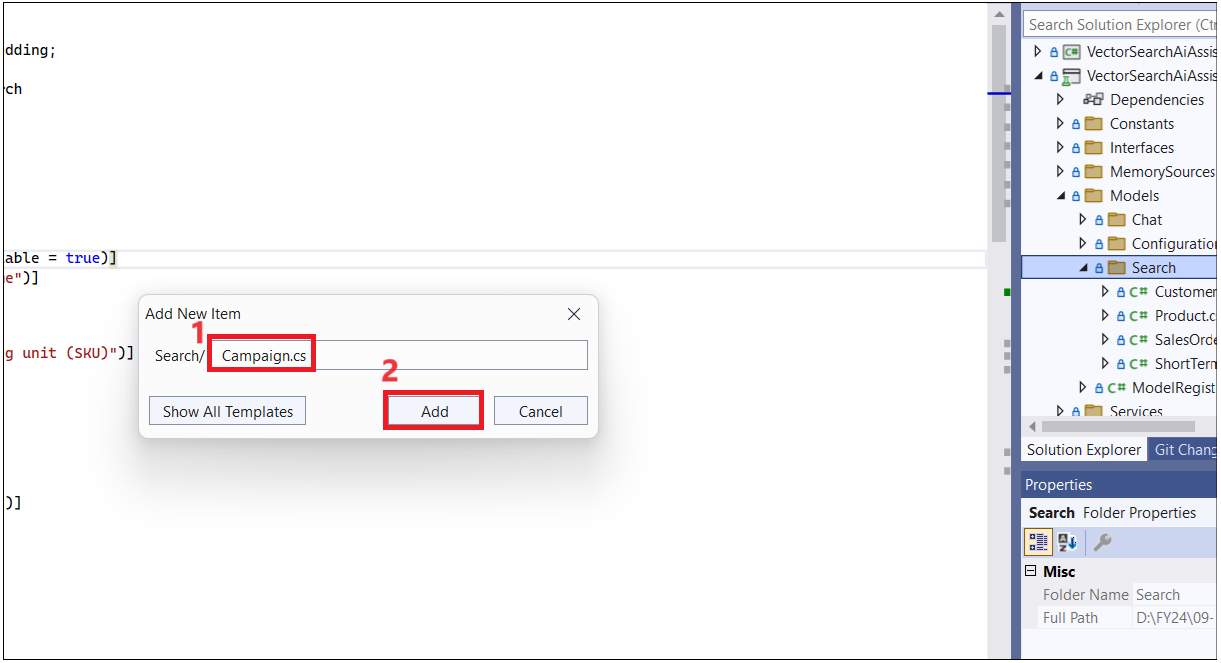


## **Task 2 : Create the campaign container class in Cosmos DB**

1. **ModelRegistry** should be aware of the type of new datatype. To create a new data type, first create a class. Go to **VectorSearchAiAssistant.Service ->Models-> Search** and **right-click -> Add -> New Item.**



1. Enter the name of the new Item as ++**Campaign.cs++** and then click on **Add.**



1. Replace the code with the below code and save the file. You can see some attributes to know searchable fields and embedded files for the vector search to be able to index and place to store the vector.

using Azure.Search.Documents.Indexes;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection.Emit;

using System.Text;

using System.Threading.Tasks;

using VectorSearchAiAssistant.SemanticKernel.Models;

using VectorSearchAiAssistant.SemanticKernel.TextEmbedding;

namespace VectorSearchAiAssistant.Service.Models.Search

{

public class Campaign : EmbeddedEntity

{

[SearchableField(IsFilterable = true)]

public string campaignId { get; set; }

[SimpleField]

[EmbeddingField(Label = "Campaign name")]

public string campaignName { get; set; }

[SimpleField]

[EmbeddingField(Label = "Campaign description")]

public string campaignDescription { get; set; }

}

}

A screenshot of a computer

Description automatically generated

## **Task 3 : Update ModelRegistry**

1. Now, we want the **Campaign.cs** class to be available in ModelRegistry.To add an entry into the **ModelRegistry**, go to **ModelRegistry** and see all the datatypes available. You need to add an entry for the Campaign class. Add the below code under the product in as shown in the below image.
2. Add the below entity to the model registry in **ModelRegistry.cs** in the **VectorSearchAiAssistant.Service** project, under the **Models** folder

{

nameof(Campaign),

new ModelRegistryEntry

{

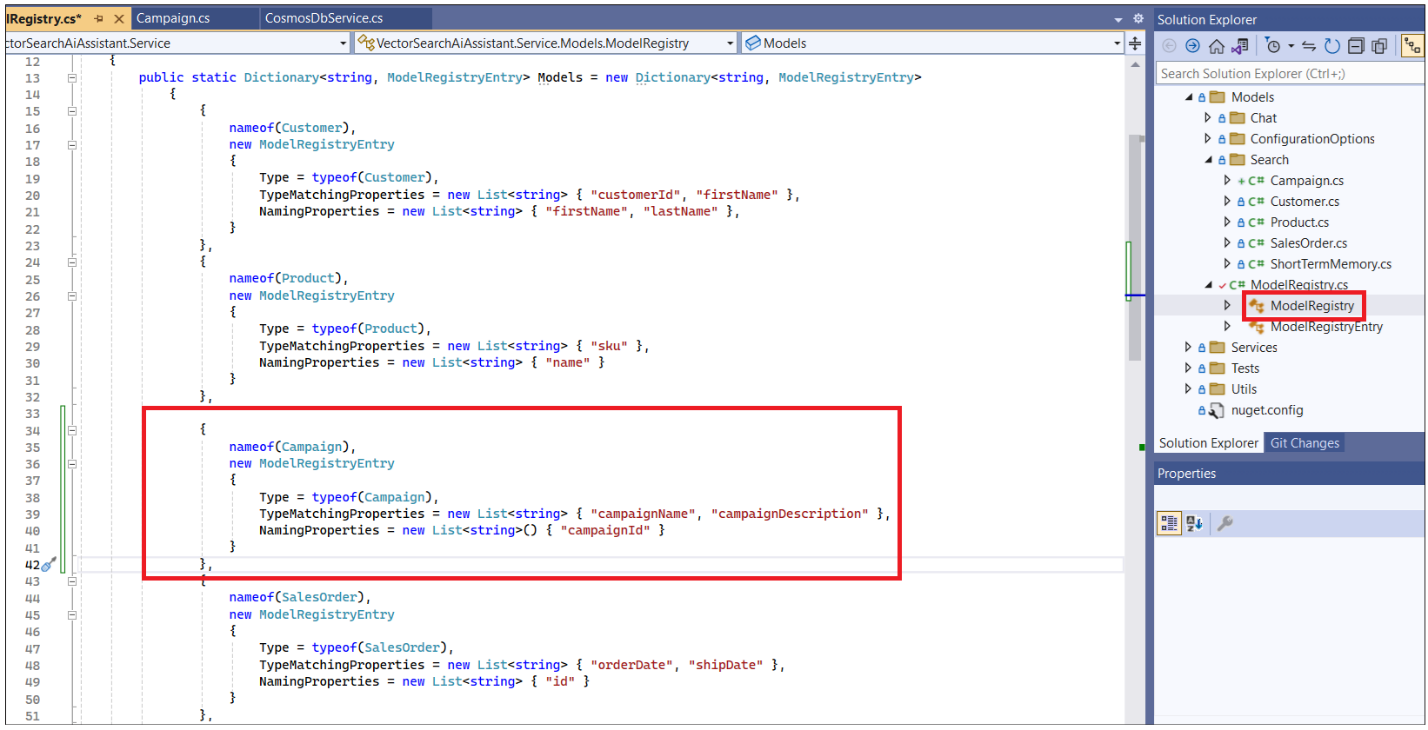
Type = typeof(Campaign),

TypeMatchingProperties = new List<string> { "campaignName", "campaignDescription" },

NamingProperties = new List<string>() { "campaignId" }

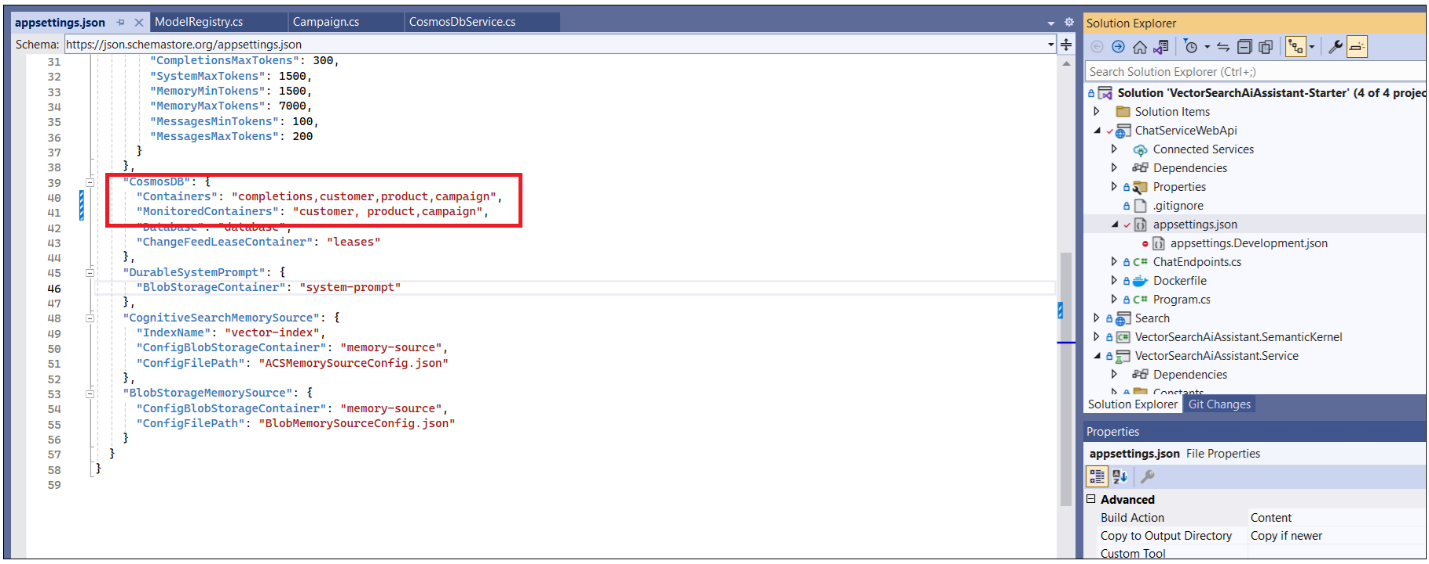
}

},



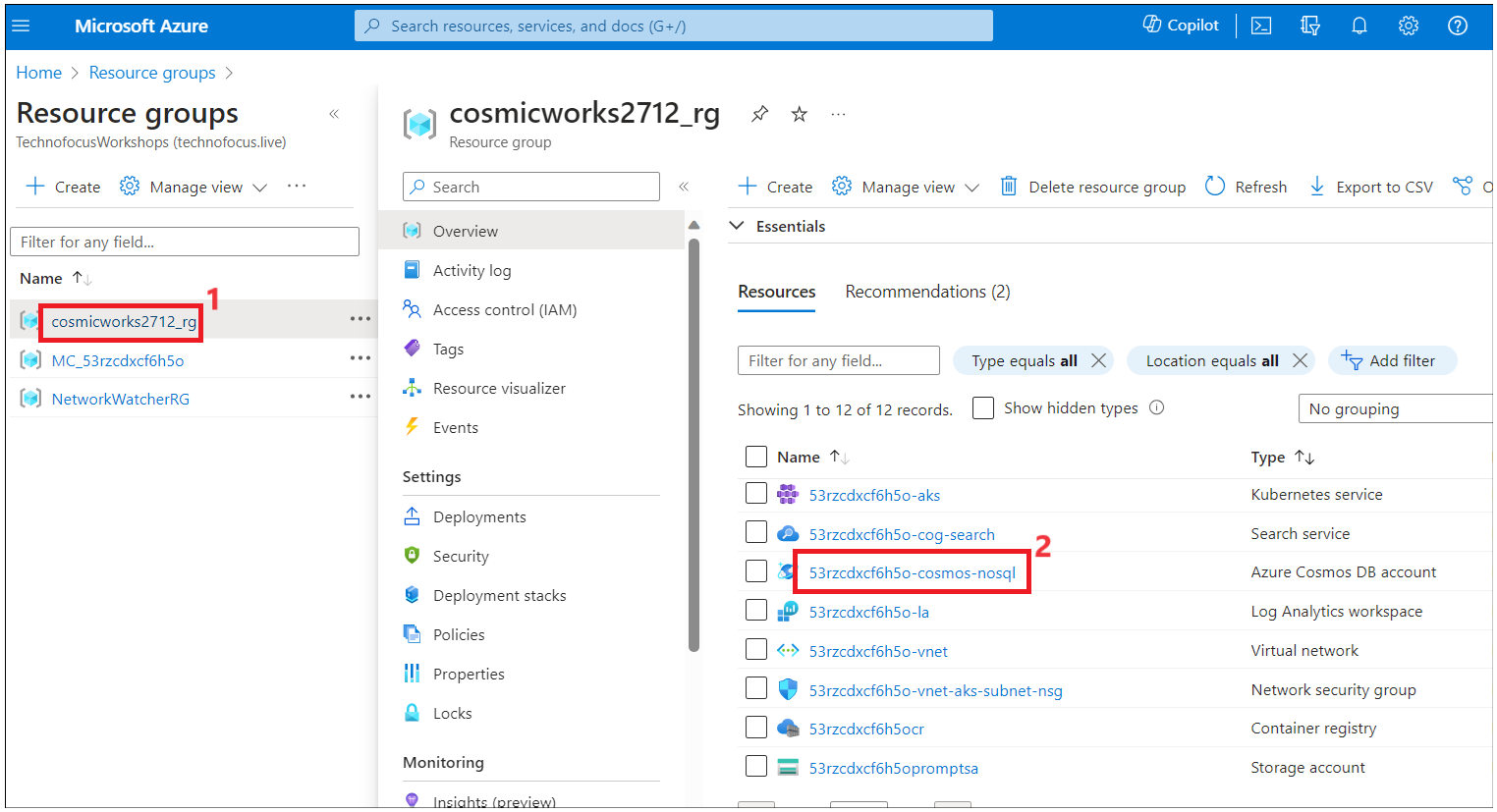
IMPORTANT: ModelREgistry is a dictionary with the name and registry of the datatype. In the above code, MatchingProperties are campaignName and campaignDescription. NamingProperties is the campaignId. These attributes are used by embedding logic to figure out which fields to vectorize and store in the Cognitive search.

1. Go to **ChatServiceWebApi -> appsettings.json** and add a ++**campaign++** class to the list of containers and **MonitoredContainers list is monitored by the change feed processor.**

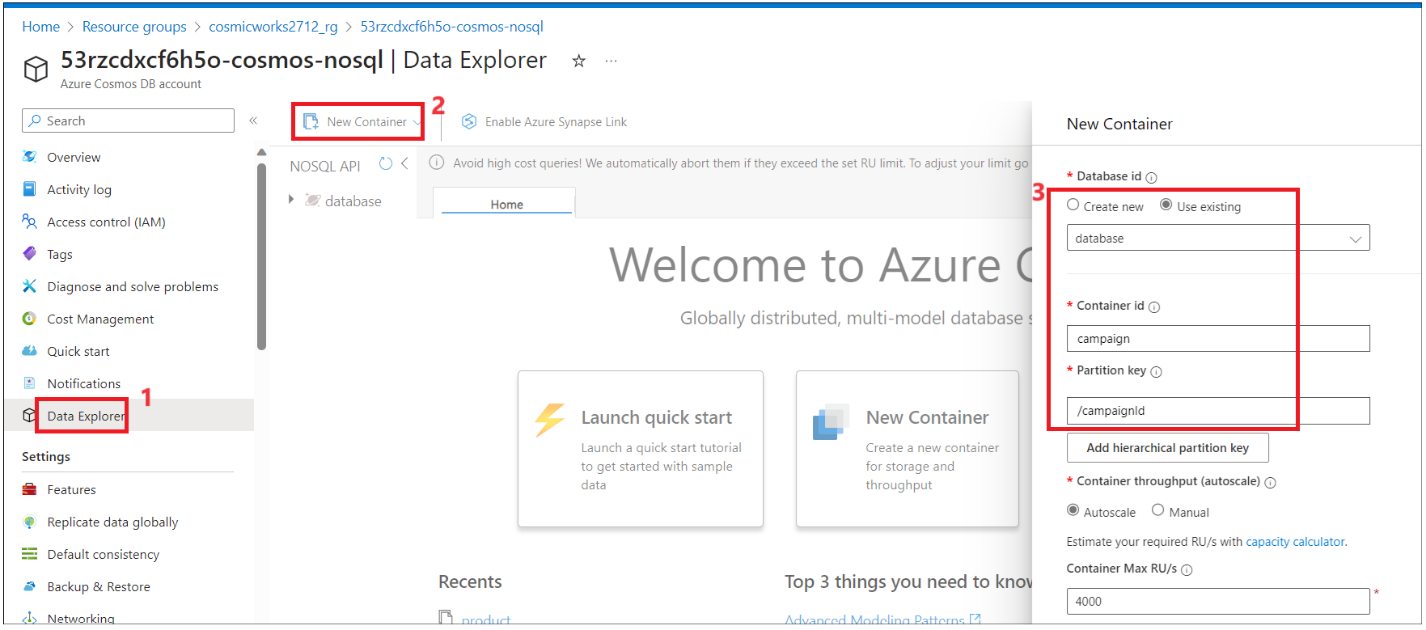
****

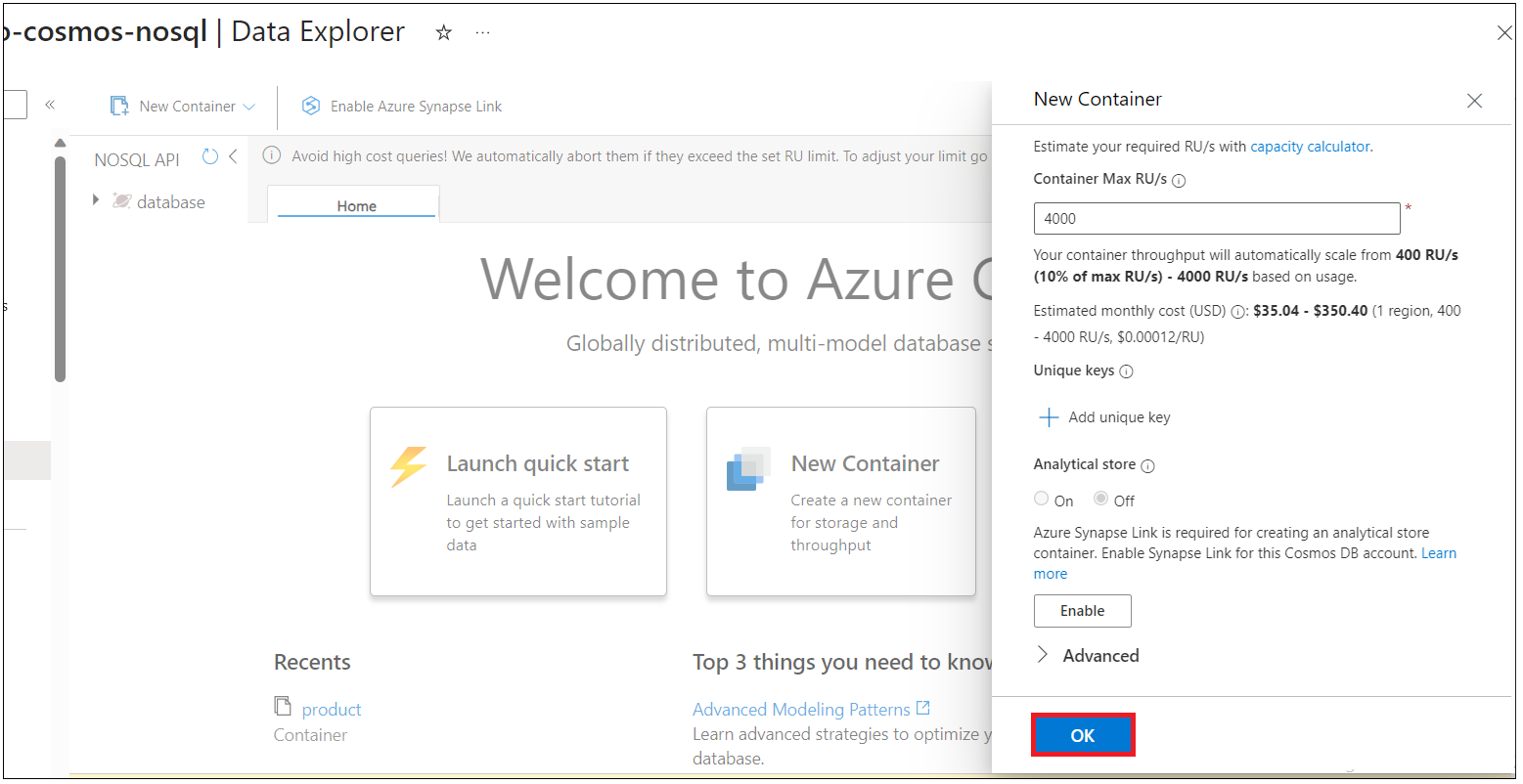
## **Task 4 : Create a** campaign container in CosmosDB and add documents

1. Switch back to the **Azure portal**. Click on Resource group and then click on **Azure Cosmos DB account.**



1. Click on **Data Explorer -> New Container** . Enter the below values scroll down and then click on **OK**.
   1. Database ID- Existing – **database**
   2. Container id - ++campaign++
   3. partition key - ++/campaignId++





1. To Create two new campaign documents in Cosmos DB , expand **campaign->Items -> New item**. Replace the code with the below code and then click on **Save.**

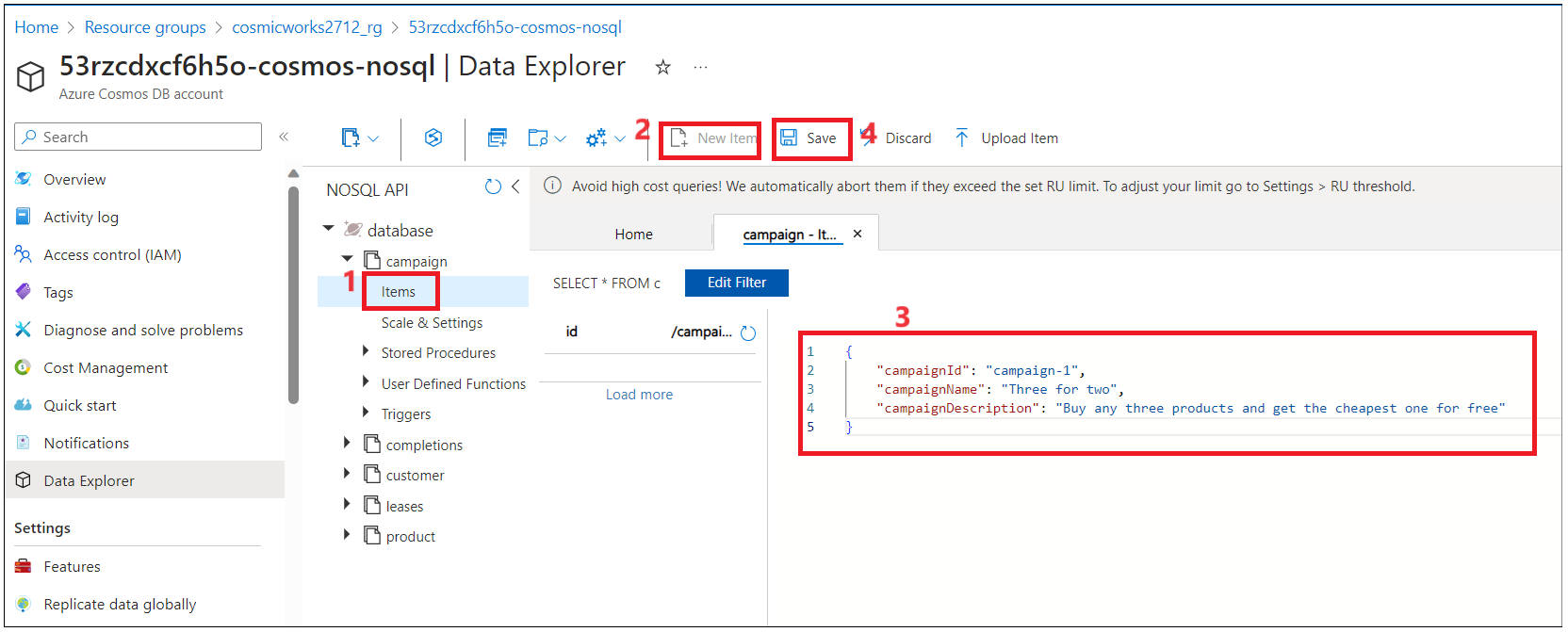
{

"campaignId": "campaign-1",

"campaignName": "Three for two",

"campaignDescription": "Buy any three products and get the cheapest one for free"

}



1. Clicm on **New item** again and add below code to add another document.

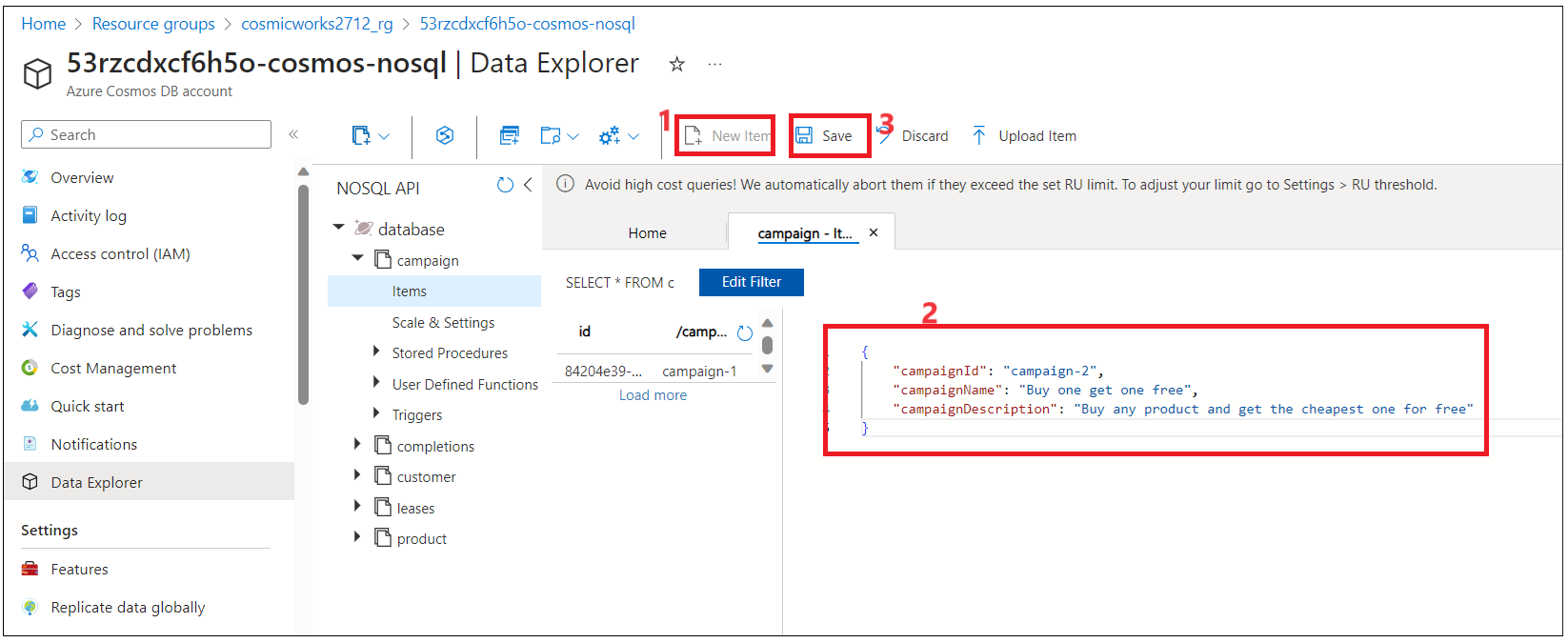
{

"campaignId": "campaign-2",

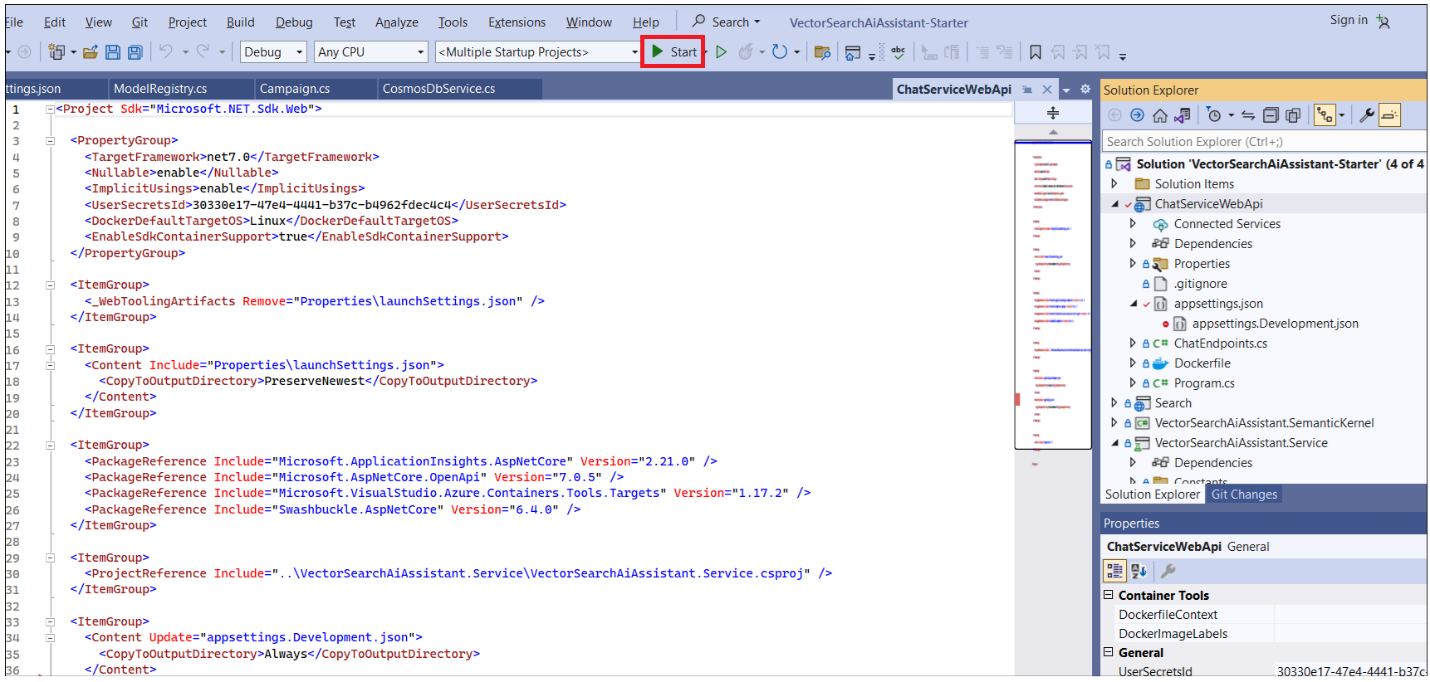
"campaignName": "Buy one get one free",

"campaignDescription": "Buy any product and get the cheapest one for free"

}



1. Switch back to Visual Studio. Run the ChatWebServiceApi project in debug and validate that the Cognitive Search index was expanded with the new entities (both the definition and the content).



1. Check CosmosDBService.cs , the new campaign document is added and change feed process is processed the document .Click **Debug ->Continue(F5).**

A screenshot of a computer

Description automatically generated

IMPORTANAT : ModelRegistry identify the type of object and returns.Once the type is provided , ragService.AddMemory adds the document to Azure Cognitive search based on the NamingProperties and indexed to use in Model.

## **Task 5 : Ask the interface about campaigns**

1. Go to the interface, click on **Create a New chat,** and ask questions about the newly vectorized campaigns, for example:

**++What are available campaigns?++**

**++Are you currently running any three for two campaigns?++**

A screenshot of a computer

Description automatically generated

1. Go back to Visual Studio and stop the solution.

**Summary**

• Chat history showing how it responded using the new data as context.

• Able to locate the new product or customer data you loaded in the Cognitive Search Index and in Cosmos DB.