Programowanie usług w chmurze

Sprawozdanie z laboratorium 1

Bartosz Sieracki 304480

Niestety na samym końcu zorientowałem się, że sugerowany format sprawozdania był inny – przy następnej okazji zrobię lepiej.

Krok 1: Utworzenie konta w Azure

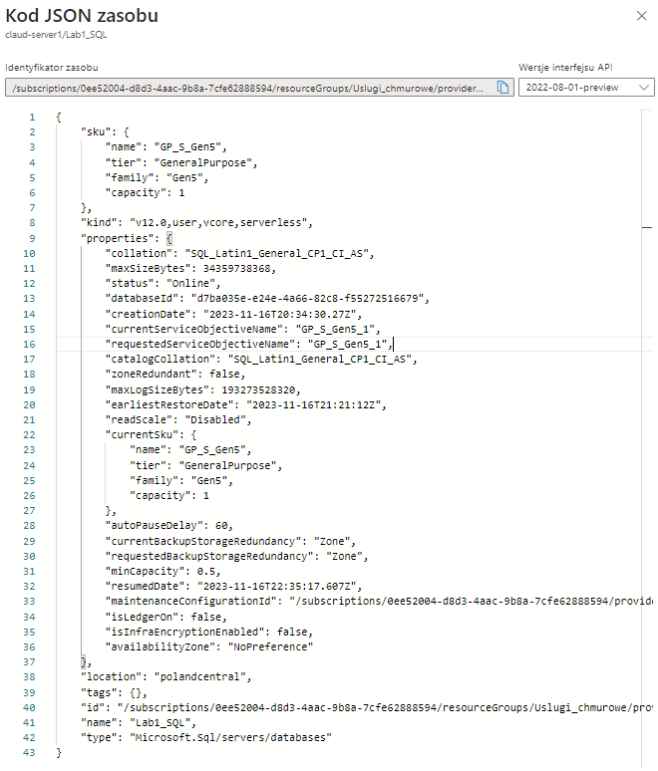
Na samym początku należało utworzyć konto Azure z domeny Politechniki.

Krok 2: Utworzenie instancji Azure SQL Database

Pierwszą czynnością w ramach technologii Azure było utworzenie grupy zasobów. Poniżej pokazano kody JSON utworzonych wszystkich zasobów prowadzących do utworzenia bazy danych, zaczynając od grupy zasobów.

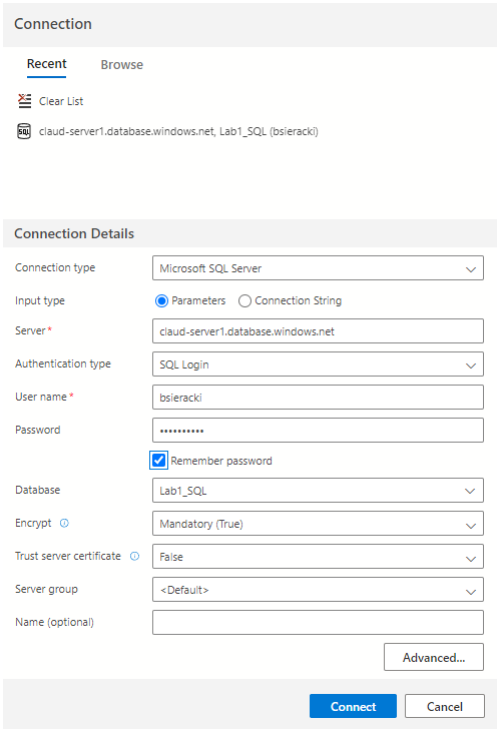
Następnie utworzono serwer o podanych niżej parametrach.

Utworzono bazę danych z przykładowymi danymi zaproponowanymi przez Azure.



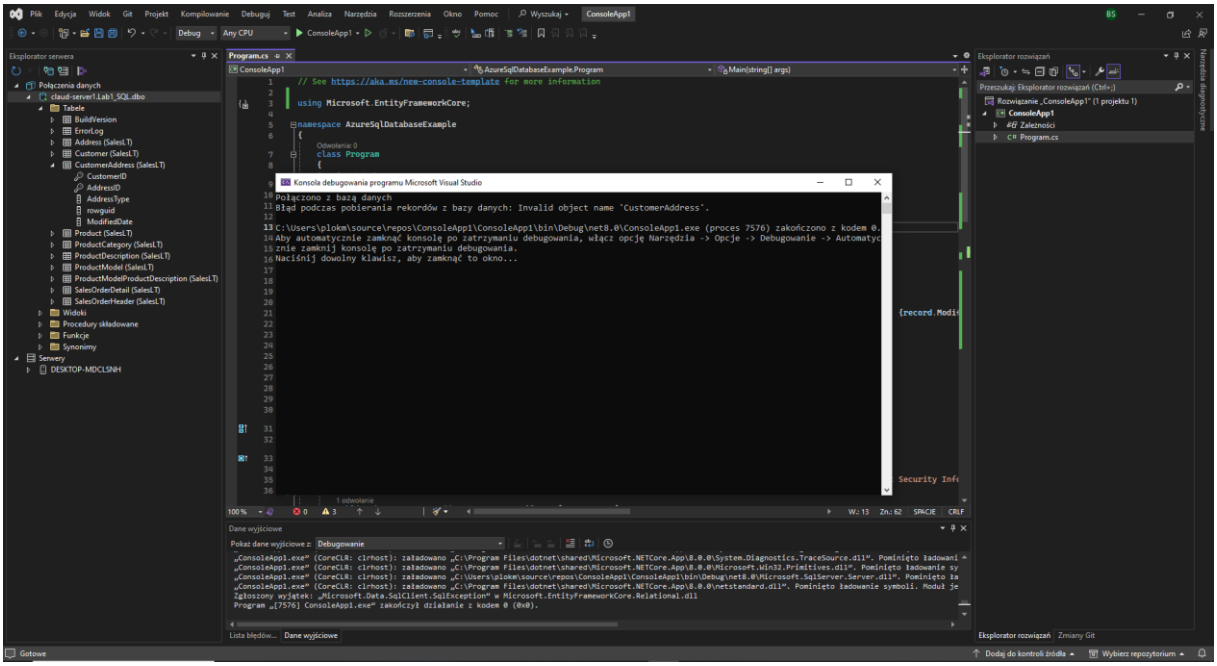
Krok 4: Połączenie z bazą danych

Połączenie z bazą danych odbyło się przy pomocy Azure Data Studio. Panel łączenia się z bazą danych przy pomocy tego oprogramowania przebiegał następująco.



Krok 5: Tworzenie aplikacji

Utworzono program w technologii .NET w celu połączenia się z bazą danych. Funkcjonalność pobierania i wyświetlania danych jest bardzo blisko celu, jednak napotkałem błąd związany z poruszaniem się po tej domyślnej bazie danych, którą podczas tworzenia zaproponował Azure.



Poniżej dołączam kod tego programu.

```
using Microsoft.EntityFrameworkCore;

namespace AzureSqlDatabaseExample
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            using (var dbContext = new YourDbContext())
            {
                Console.WriteLine("Połączono z bazą danych");
                try
                {
                    var records = dbContext.CustomerAddress.Take(5).ToList();
                    Console.WriteLine("Rekordy w tabeli:");

                    foreach (var record in records)
                    {
                        Console.WriteLine($"{record.CustomerID}: {record.AddressID},
{record.AddressType}, {record.rowguid}, {record.ModifiedDate}");
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Błąd podczas pobierania rekordów z bazy
danych: {ex.Message}");
                }
            }
        }
    }
    public class YourDbContext : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=tcp:claud-
server1.database.windows.net,1433;Initial Catalog=Lab1_SQL;Persist Security
Info=False;User
ID=bsieracki;Password=Azuresql1@;MultipleActiveResultSets=False;Encrypt=True;TrustServ
erCertificate=False;Connection Timeout=30;");
        }
        public DbSet<CustomerAddress> CustomerAddress { get; set; }
    }
    [Keyless]
    public class CustomerAddress
    {
        public int CustomerID { get; set; }
        public int AddressID { get; set; }
        public string AddressType { get; set; }
        public string rowguid { get; set; }
        public string ModifiedDate { get; set; }
    }

}
```

Krok 6: Konfiguracja maszyny wirtualnej

Następnie stworzono maszynę wirtualną, której opcje wyborów pokazano poniżej.

Podstawowe

Subskrypcja	Azure for Students
Grupa zasobów	Uslugi_chmurowe
Nazwa maszyny wirtualnej	virtual-machine
Region	Norway East
Opcje dostępności	Nie jest wymagana żadna nadmiarowość infrastruktury
Typ zabezpieczeń	Standardowe
Obraz	Ubuntu Server 22.04 LTS — Gen2
Architektura maszyny wirtualnej	x64
Rozmiar	Standard DS1 v2 (vcpu: 1, 3.5 GiB pamięci)
Typ uwierzytelniania	Hasło
Nazwa użytkownika	bsieracki
Publiczne porty ruchu przychodzącego	HTTP, HTTPS, SSH
Azure Spot	Nie

Dyski

Rozmiar dysku systemu operacyjnego	Domyślne ustawienia obrazu
Typ dysku systemu operacyjnego	SSD w warstwie Premium — LRS
Użyj dysków zarządzanych	Tak
Usuń dysk systemu operacyjnego z maszyną wirtualną	Włączone
Efemeryczny dysk systemu operacyjnego	Nie

Sieć

Sieć wirtualna	(nowe) virtual-machine-vnet
Podsieć	(nowy) default (10.0.0.0/24)
Publiczny adres IP	(nowe) virtual-machine-ip
Przyspieszona sieć	Wyłączone
Umieścić tę maszynę wirtualną za istniejącym rozwiązaniem do równoważenia obciążenia?	Nie
Usuwanie publicznego adresu IP i karty sieciowej po usunięciu maszyny wirtualnej	Wyłączone

Zarządzanie

Microsoft Defender dla Chmury	Podstawowa (bezpłatnie)
Tożsamość zarządzana przypisana przez system	Wyłączone
Zaloguj się przy użyciu usługi Microsoft Azure AD	Wyłączone
Automatyczne zamykanie	Wyłączone
Kopia zapasowa	Wyłączone
Włącz poprawkę na gorąco	Wyłączone
Opcje orkiestracji poprawek	Domyślne ustawienia obrazu

Monitorowanie

Alerty	Wyłączone
Diagnostyka rozruchu	Włączone
Włącz diagnostykę gościa systemu operacyjnego	Wyłączone

Zaawansowane

Rozszerzenia	Brak
Aplikacje maszyny wirtualnej	Brak
Cloud init	Nie
Dane użytkownika	Nie
Typ kontrolera dysku	SCSI
Grupa umieszczania w pobliżu	Brak
Grupa rezerwacji pojemności	Brak

Krok 7: Praca z Azure Table Storage

Stworzono Storage account oraz utworzono tabelę, do której dodano ręcznie przykładowe dane należące do dwóch kategorii – owoc i warzywo (PartitionKey) oraz posiadające atrybuty (nazwa, kolor, liczba sztuk).

Podstawowe

Subskrypcja	Azure for Students
Grupa zasobów	NetworkWatcherRG
Lokalizacja	polandcentral
Nazwa konta magazynu	nazwakontamagazynu
Model wdrożenia	Resource Manager
Wydajność	Standard
Replikacja	Magazyn geograficznie nadmiarowy z dostępem do odczytu (RA-GRS)

Zaawansowane

Włącz hierarchiczną przestrzeń nazw	Wyłączone
Włącz system plików sieciowych w wersji 3	Wyłączone
Zezwalaj na replikację między dzierżawami	Wyłączone
Warstwa dostępu	Hot
Włącz protokół SFTP	Wyłączone
Duże udziały plików	Disabled

Sieć

Łączność sieciowa	Publiczny punkt końcowy (wszystkie sieci)
Domyślna warstwa routingu	Routing sieci firmy Microsoft
Typ punktu końcowego	Standardowy

Zabezpieczenia

Bezpieczny transfer	Wyłączone
Zezwalaj na dostęp do klucza konta magazynu	Włączone
Domyślnie w celu autoryzacji rozwiązania Microsoft Entra w witrynie Azure Portal	Wyłączone
Dostęp anonimowy do obiektu blob	Wyłączone
Minimalna wersja protokołu TLS	Wersja 1.2
Dozwolony zakres operacji kopiowania (wersja zapoznawcza)	Z dowolnego konta magazynu

Ochrona danych

Przywracanie do punktu w czasie	Wyłączone
Usuwanie nietrwałe obiektów blob	Włączone
Okres przechowywania obiektów blob w dniach	7
Usuwanie nietrwałe kontenerów	Wyłączone
Usuwanie nietrwałe udziałów plików	Włączone
Okres przechowywania udziału plików w dniach	7
Przechowywanie wersji	Wyłączone
Zestawienie zmian obiektu blob	Wyłączone
Obsługa niezmienności na poziomie wersji	Wyłączone

Szyfrowanie

Typ szyfrowania	Klucze zarządzane przez firmę Microsoft
Włącz obsługę kluczy zarządzanych przez klienta	Tylko obiekty blob i pliki
Włącz szyfrowanie infrastruktury	Wyłączone

Nazwa właściwości	Typ	Wartość
PartitionKey	String	Owoc
RowKey	String	1
Timestamp	DateTi...	2023-11-17T00:50:01.0792634Z
Kolor	String	czerwony
Liczba_sztuk	Int32	5
Nazwa	String	truskawka

Dodaj właściwość

Krok 8. Konfiguracja Firewalla Azure SQL Database

Ustawione zostały parametry firewalla, adres IP, który ma dostęp do bazy.

Czas utworzenia

Szacowany czas tworzenia konta (w minutach)	2

Szacowany czas tworzenia jest obliczany na podstawie wybranej lokalizacji

Podstawy

Subskrypcja	Azure for Students
Grupa zasobów	Uslugi_chmurowe
Lokalizacja	Israel Central
Nazwa konta	(nowy) konto-cosmos
API	Podstawowe (SQL)
Tryb wydajności	Aprowizowana przepływność
Nadmiarowość geograficzna	Włącz
Zapisy w wielu regionach	Włącz
Strefy dostępności	Wyłącz

Zasady kopii zapasowych

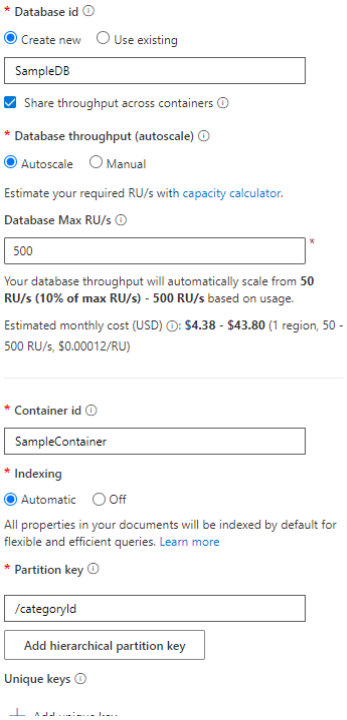
Zasady kopii zapasowych	Periodic
Nadmiarowość magazynu kopii zapasowych	Strefowo nadmiarowy magazyn kopii zapasowych

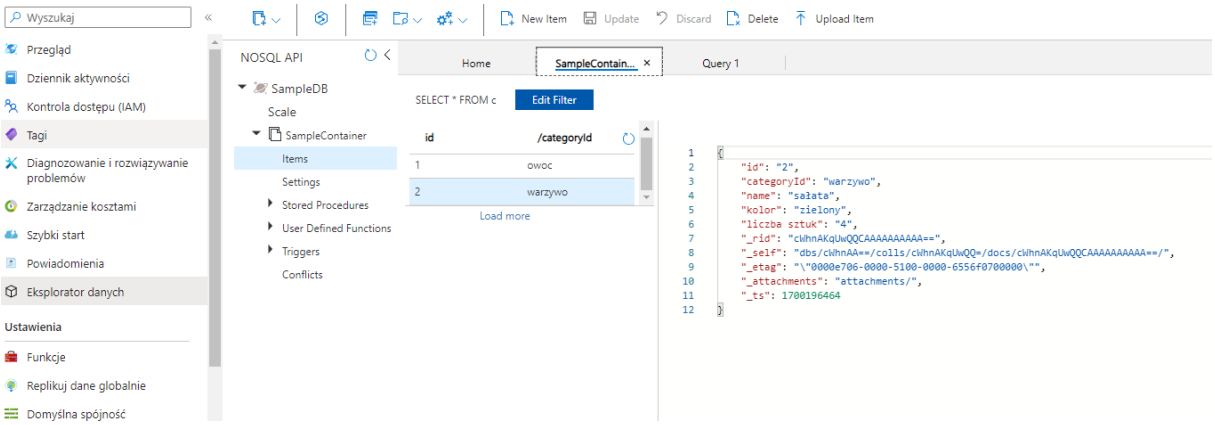
Sieć

Metoda łączności	Wszystkie sieci
Minimum TLS Protocol	TLS 1.2

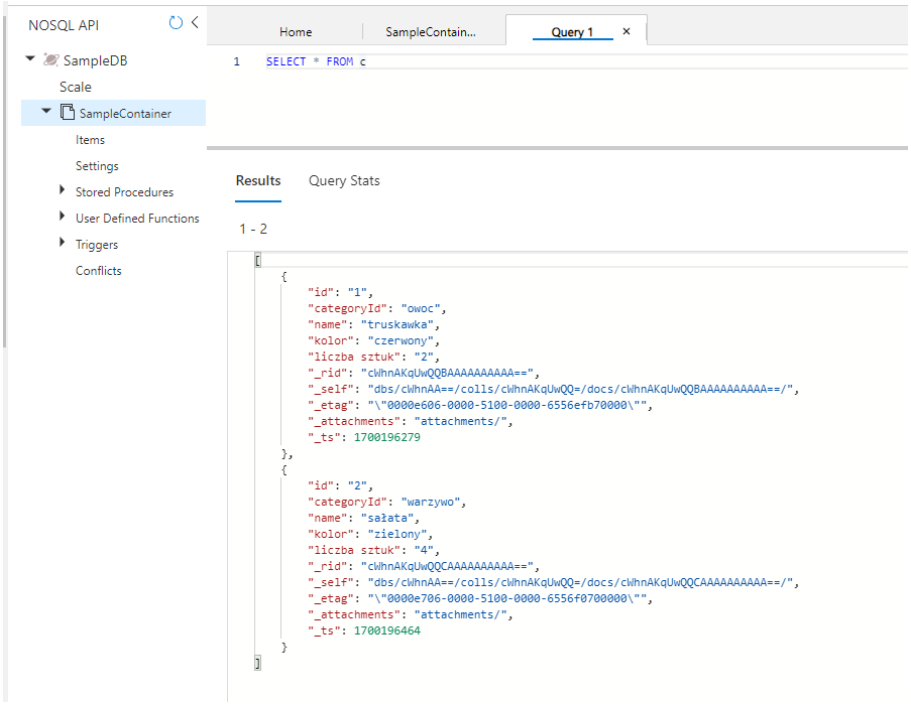
Krok 9. Azure Cosmos DB

Utworzono nowe konto Azure Cosmos DB z API SQL, baza danych SampleDB oraz kontener SampleContainer. Następnie skorzystano z zakładki Data Explorer oraz dodano do bazy danych przykładowe dokumenty o kategoriach owoc i warzywo za pomoca okna graficznego. Następnie to samo zrobiono w sposób tekstowy w query.



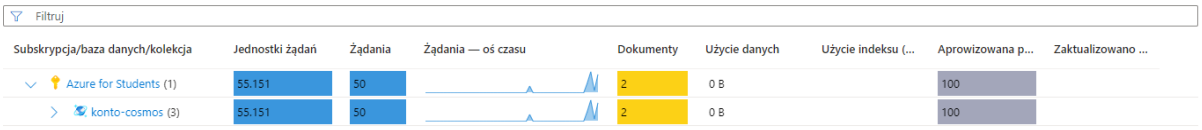


Następnie wykorzystano query do wyświetlenia wszystkich przedmiotów w tabeli, dodatkowo można również obejrzeć różne metryki. Ustawiony parametr maksymalnej liczby jednostek przetwarzania żądań na 1000 RU/s.

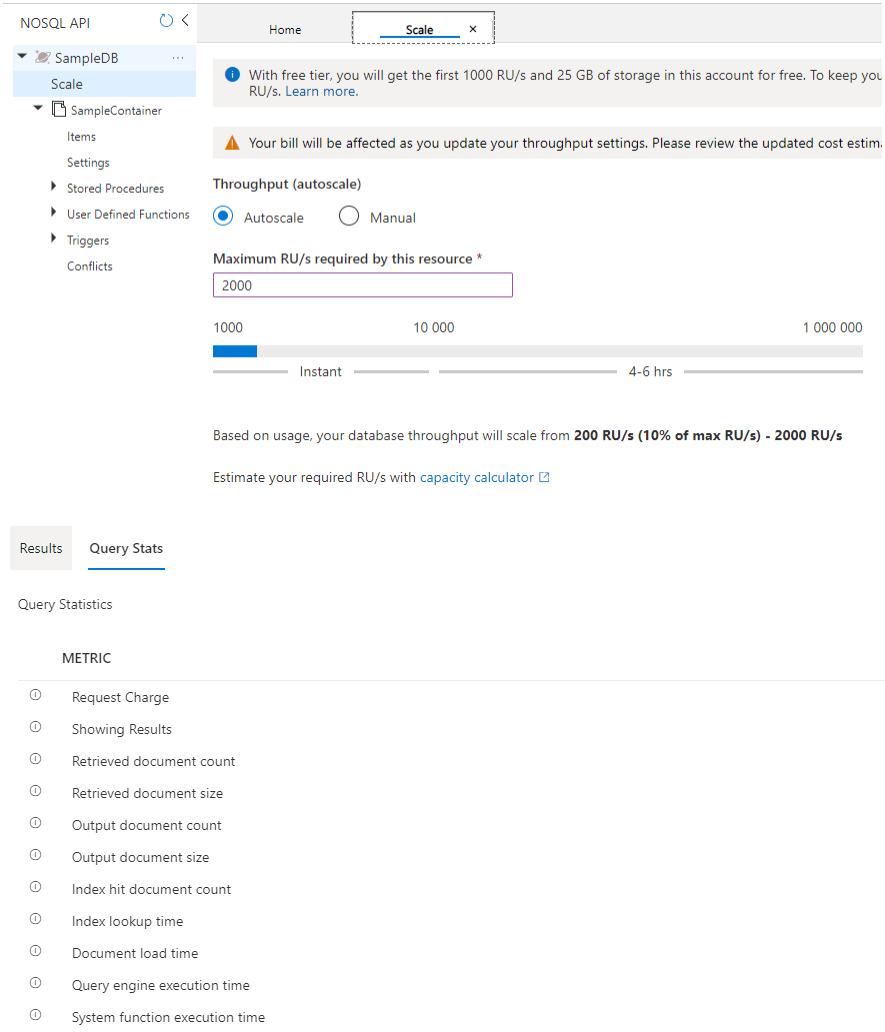


Query Statistics

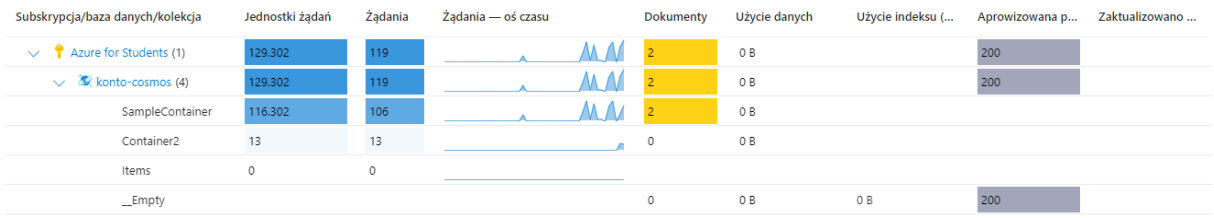
METRIC	VALUE
Request Charge	2.28 RUs
Showing Results	1 - 2
Retrieved document count	2
Retrieved document size	598 bytes
Output document count	2
Output document size	648 bytes
Index hit document count	2
Index lookup time	0 ms
Document load time	0.02 ms
Query engine execution time	0.01 ms
System function execution time	0 ms



Następnie sprawdzono skalowanie, przypisując wartość tego parametru na 2000 RU/s. Oznacza to, że baza danych będzie miała możliwość skalowania z 200 do 2000 jednostek RU/s. Jedyną obserwowalną różnicą w metrykach jest zmniejszenie czasu document load time, ale parametr ten raczej nie jest ściśle związany z tą jednostką. Nic więcej nie uległo zmianie w egzekwowaniu zapytania.



Po tej zmianie Azure Monitor wygląda następująco



Następnie wykorzystano funkcję ‘Szybki start’, aby utworzyć kontener ‘Items’ oraz skorzystać z utworzonej bazy danych ‘ToDoList’ Azure. Następnie pobrano i uruchomiono aplikację .NET. Poniżej pokazano wynik z konsoli oraz dołączono kod.

```
Beginning operations...

Created Database: ToDoList

Created Container: Items

Current provisioned throughput : 400

New provisioned throughput : 500

Created item in database with id: Andersen.1 Operation consumed 12,95 RUs.

Created item in database with id: Wakefield.7 Operation consumed 14,86 RUs.

Running query: SELECT * FROM c WHERE c.PartitionKey = 'Andersen'

Updated Family [Wakefield,Wakefield.7].
        Body is now: {"id":"Wakefield.7","partitionKey":"Wakefield","LastName":"Wakefield","Parents":[{"FamilyName":"Wak
efield","FirstName":"Robin"},{"FamilyName":"Miller","FirstName":"Ben"}],"Children":[{"FamilyName":"Merriam","FirstName":
"Jesse","Gender":"female","Grade":6,"Pets":[{"GivenName":"Goofy"},{"GivenName":"Shadow"}]},{"FamilyName":"Miller","First
Name":"Lisa","Gender":"female","Grade":1,"Pets":null}],"Address":{"State":"NY","County":"Manhattan","City":"NY"},"IsRegi
stered":true}

Deleted Family [Wakefield,Wakefield.7]

Deleted Database: ToDoList

End of demo, press any key to exit.
```

```
using System;
using System.Threading.Tasks;
using System.Configuration;
using System.Collections.Generic;
using System.Net;
using Microsoft.Azure.Cosmos;

namespace CosmosGettingStartedTutorial
{
    class Program
    {
        // The Azure Cosmos DB endpoint for running this sample.
        private static readonly string EndpointUri = ConfigurationManager.AppSettings["EndPointUri"];

        // The primary key for the Azure Cosmos account.
        private static readonly string PrimaryKey = ConfigurationManager.AppSettings["PrimaryKey"];

        // The Cosmos client instance
        private CosmosClient cosmosClient;

        // The database we will create
        private Database database;

        // The container we will create.
        private Container container;

        // The name of the database and container we will create
        private string databaseId = "ToDoList";
        private string containerId = "Items";

        // <Main>
        public static async Task Main(string[] args)
        {
            try
            {
                Console.WriteLine("Beginning operations...\n");
                Program p = new Program();
                await p.GetStartedDemoAsync();

            }
            catch (CosmosException de)
            {
                Exception baseException = de.GetBaseException();
                Console.WriteLine("{0} error occurred: {1}", de.StatusCode, de);
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: {0}", e);
            }
```

```
            finally
            {
                Console.WriteLine("End of demo, press any key to exit.");
                Console.ReadKey();
            }
        }
        // </Main>

        // <GetStartedDemoAsync>
        /// <summary>
        /// Entry point to call methods that operate on Azure Cosmos DB resources in this sample
        /// </summary>
        public async Task GetStartedDemoAsync()
        {
            // Create a new instance of the Cosmos Client
            this.cosmosClient = new CosmosClient(EndpointUri, PrimaryKey, new CosmosClientOptions() {
ApplicationName = "CosmosDBDotnetQuickstart" });
            await this.CreateDatabaseAsync();
            await this.CreateContainerAsync();
            await this.ScaleContainerAsync();
            await this.AddItemsToContainerAsync();
            await this.QueryItemsAsync();
            await this.ReplaceFamilyItemAsync();
            await this.DeleteFamilyItemAsync();
            await this.DeleteDatabaseAndCleanupAsync();
        }
        // </GetStartedDemoAsync>

        // <CreateDatabaseAsync>
        /// <summary>
        /// Create the database if it does not exist
        /// </summary>
        private async Task CreateDatabaseAsync()
        {
            // Create a new database
            this.database = await this.cosmosClient.CreateDatabaseIfNotExistsAsync(databaseId);
            Console.WriteLine("Created Database: {0}\n", this.database.Id);
        }
        // </CreateDatabaseAsync>

        // <CreateContainerAsync>
        /// <summary>
        /// Create the container if it does not exist.
        /// Specifiy "/partitionKey" as the partition key path since we're storing family information,
to ensure good distribution of requests and storage.
        /// </summary>
        /// <returns></returns>
        private async Task CreateContainerAsync()
        {
            // Create a new container
            this.container = await this.database.CreateContainerIfNotExistsAsync(containerId,
"/partitionKey");
            Console.WriteLine("Created Container: {0}\n", this.container.Id);
        }
        // </CreateContainerAsync>

        // <ScaleContainerAsync>
        /// <summary>
        /// Scale the throughput provisioned on an existing Container.
        /// You can scale the throughput (RU/s) of your container up and down to meet the needs of the
workload. Learn more: https://aka.ms/cosmos-request-units
        /// </summary>
        /// <returns></returns>
        private async Task ScaleContainerAsync()
        {
            // Read the current throughput
            try
            {
                int? throughput = await this.container.ReadThroughputAsync();
                if (throughput.HasValue)
                {
                    Console.WriteLine("Current provisioned throughput : {0}\n", throughput.Value);
                    int newThroughput = throughput.Value + 100;
                    // Update throughput
```

```
                    await this.container.ReplaceThroughputAsync(newThroughput);
                    Console.WriteLine("New provisioned throughput : {0}\n", newThroughput);
                }
            }
            catch (CosmosException cosmosException) when (cosmosException.StatusCode ==
HttpStatusCode.BadRequest)
            {
                Console.WriteLine("Cannot read container throuthput.");
                Console.WriteLine(cosmosException.ResponseBody);
            }

        }
        // </ScaleContainerAsync>

        // <AddItemsToContainerAsync>
        /// <summary>
        /// Add Family items to the container
        /// </summary>
        private async Task AddItemsToContainerAsync()
        {
            // Create a family object for the Andersen family
            Family andersenFamily = new Family
            {
                Id = "Andersen.1",
                PartitionKey = "Andersen",
                LastName = "Andersen",
                Parents = new Parent[]
                {
                    new Parent { FirstName = "Thomas" },
                    new Parent { FirstName = "Mary Kay" }
                },
                Children = new Child[]
                {
                    new Child
                    {
                        FirstName = "Henriette Thaulow",
                        Gender = "female",
                        Grade = 5,
                        Pets = new Pet[]
                        {
                            new Pet { GivenName = "Fluffy" }
                        }
                    }
                },
                Address = new Address { State = "WA", County = "King", City = "Seattle" },
                IsRegistered = false
            };

            try
            {
                // Read the item to see if it exists.
                ItemResponse<Family> andersenFamilyResponse = await
this.container.ReadItemAsync<Family>(andersenFamily.Id, new PartitionKey(andersenFamily.PartitionKey));
                Console.WriteLine("Item in database with id: {0} already exists\n",
andersenFamilyResponse.Resource.Id);
            }
            catch(CosmosException ex) when (ex.StatusCode == HttpStatusCode.NotFound)
            {
                // Create an item in the container representing the Andersen family. Note we provide
the value of the partition key for this item, which is "Andersen"
                ItemResponse<Family> andersenFamilyResponse = await
this.container.CreateItemAsync<Family>(andersenFamily, new PartitionKey(andersenFamily.PartitionKey));

                // Note that after creating the item, we can access the body of the item with the
Resource property off the ItemResponse. We can also access the RequestCharge property to see the amount
of RUs consumed on this request.
                Console.WriteLine("Created item in database with id: {0} Operation consumed {1}
RUs.\n", andersenFamilyResponse.Resource.Id, andersenFamilyResponse.RequestCharge);
            }

            // Create a family object for the Wakefield family
            Family wakefieldFamily = new Family
            {
                Id = "Wakefield.7",
```

```
                PartitionKey = "Wakefield",
                LastName = "Wakefield",
                Parents = new Parent[]
                {
                    new Parent { FamilyName = "Wakefield", FirstName = "Robin" },
                    new Parent { FamilyName = "Miller", FirstName = "Ben" }
                },
                Children = new Child[]
                {
                    new Child
                    {
                        FamilyName = "Merriam",
                        FirstName = "Jesse",
                        Gender = "female",
                        Grade = 8,
                        Pets = new Pet[]
                        {
                            new Pet { GivenName = "Goofy" },
                            new Pet { GivenName = "Shadow" }
                        }
                    },
                    new Child
                    {
                        FamilyName = "Miller",
                        FirstName = "Lisa",
                        Gender = "female",
                        Grade = 1
                    }
                },
                Address = new Address { State = "NY", County = "Manhattan", City = "NY" },
                IsRegistered = true
            };

            try
            {
                // Read the item to see if it exists
                ItemResponse<Family> wakefieldFamilyResponse = await
this.container.ReadItemAsync<Family>(wakefieldFamily.Id, new
PartitionKey(wakefieldFamily.PartitionKey));
                Console.WriteLine("Item in database with id: {0} already exists\n",
wakefieldFamilyResponse.Resource.Id);
            }
            catch(CosmosException ex) when (ex.StatusCode == HttpStatusCode.NotFound)
            {
                // Create an item in the container representing the Wakefield family. Note we provide
the value of the partition key for this item, which is "Wakefield"
                ItemResponse<Family> wakefieldFamilyResponse = await
this.container.CreateItemAsync<Family>(wakefieldFamily, new
PartitionKey(wakefieldFamily.PartitionKey));

                // Note that after creating the item, we can access the body of the item with the
Resource property off the ItemResponse. We can also access the RequestCharge property to see the amount
of RUs consumed on this request.
                Console.WriteLine("Created item in database with id: {0} Operation consumed {1}
RUs.\n", wakefieldFamilyResponse.Resource.Id, wakefieldFamilyResponse.RequestCharge);
            }
        }
        // </AddItemsToContainerAsync>

        // <QueryItemsAsync>
        /// <summary>
        /// Run a query (using Azure Cosmos DB SQL syntax) against the container
        /// Including the partition key value of lastName in the WHERE filter results in a more
efficient query
        /// </summary>
        private async Task QueryItemsAsync()
        {
            var sqlQueryText = "SELECT * FROM c WHERE c.PartitionKey = 'Andersen'";

            Console.WriteLine("Running query: {0}\n", sqlQueryText);

            QueryDefinition queryDefinition = new QueryDefinition(sqlQueryText);
            FeedIterator<Family> queryResultSetIterator =
this.container.GetItemQueryIterator<Family>(queryDefinition);
```

```
            List<Family> families = new List<Family>();

            while (queryResultSetIterator.HasMoreResults)
            {
                FeedResponse<Family> currentResultSet = await queryResultSetIterator.ReadNextAsync();
                foreach (Family family in currentResultSet)
                {
                    families.Add(family);
                    Console.WriteLine("\tRead {0}\n", family);
                }
            }
        }
        // </QueryItemsAsync>

        // <ReplaceFamilyItemAsync>
        /// <summary>
        /// Replace an item in the container
        /// </summary>
        private async Task ReplaceFamilyItemAsync()
        {
            ItemResponse<Family> wakefieldFamilyResponse = await
this.container.ReadItemAsync<Family>("Wakefield.7", new PartitionKey("Wakefield"));
            var itemBody = wakefieldFamilyResponse.Resource;

            // update registration status from false to true
            itemBody.IsRegistered = true;
            // update grade of child
            itemBody.Children[0].Grade = 6;

            // replace the item with the updated content
            wakefieldFamilyResponse = await this.container.ReplaceItemAsync<Family>(itemBody,
itemBody.Id, new PartitionKey(itemBody.PartitionKey));
            Console.WriteLine("Updated Family [{0},{1}].\n \tBody is now: {2}\n", itemBody.LastName,
itemBody.Id, wakefieldFamilyResponse.Resource);
        }
        // </ReplaceFamilyItemAsync>

        // <DeleteFamilyItemAsync>
        /// <summary>
        /// Delete an item in the container
        /// </summary>
        private async Task DeleteFamilyItemAsync()
        {
            var partitionKeyValue = "Wakefield";
            var familyId = "Wakefield.7";

            // Delete an item. Note we must provide the partition key value and id of the item to
delete
            ItemResponse<Family> wakefieldFamilyResponse = await
this.container.DeleteItemAsync<Family>(familyId,new PartitionKey(partitionKeyValue));
            Console.WriteLine("Deleted Family [{0},{1}]\n", partitionKeyValue, familyId);
        }
        // </DeleteFamilyItemAsync>

        // <DeleteDatabaseAndCleanupAsync>
        /// <summary>
        /// Delete the database and dispose of the Cosmos Client instance
        /// </summary>
        private async Task DeleteDatabaseAndCleanupAsync()
        {
            DatabaseResponse databaseResourceResponse = await this.database.DeleteAsync();
            // Also valid: await this.cosmosClient.Databases["FamilyDatabase"].DeleteAsync();

            Console.WriteLine("Deleted Database: {0}\n", this.databaseId);

            //Dispose of CosmosClient
            this.cosmosClient.Dispose();
        }
        // </DeleteDatabaseAndCleanupAsync>
    }
}
```