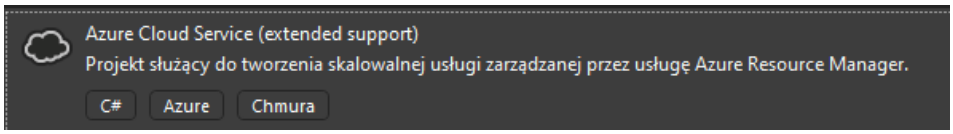
Windows Azure ver. 1.1
laboratorium
2024/25

K.M. Ocetkiewicz, T. Goluch

1. Wstęp

Windows Azure jest przykładem usługi, popularnie nazywanej chmurą. W modelu tym dostajemy do dyspozycji pewne środowisko obejmujące elementy takie jak maszyny wirtualne, bazy danych, przestrzeń na dane, szynę komunikacyjną itp. Z jednej strony rozwiązanie to ma zalety – nie musimy administrować fizycznymi komputerami, martwić się o ich wydajność i skalowalność naszej aplikacji (zawsze możemy zażyczyć sobie większej liczby maszyn), z drugiej jednak strony środowisko to narzuca pewne ograniczenia co do modelu programistycznego, a każdy element środowiska jest odpowiednio wyceniony i użycie go kosztuje (i to niemało) co powoduje, że Azure znajdzie raczej zastosowanie w dużych, komercyjnych projektach.

Projekty korzystające z Azure tworzymy wybierając nowy projekt → Templates → Visual C# → Cloud → Azure Cloud Service.



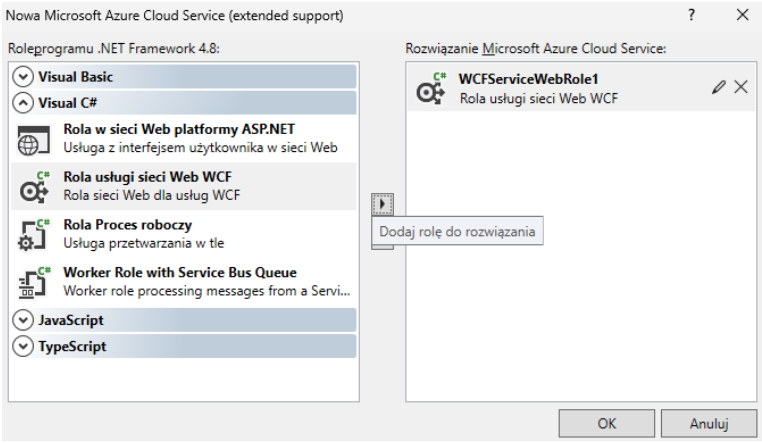
Większość API Azure korzysta z HTTP i w związku z tym błędy opisane są w sposób zaczerpnięty z tego protokołu. Na przykład, próba dostępu do nieistniejącego zasobu zakończy się błędem „404 Not found” a próba utworzenia kontenera o nieprawidłowej nazwie skutkuje błędem „400 Bad Request”.

Dodatkowa literatura:

- Azure for developers overview: https://learn.microsoft.com/azure/developer/intro/azure-developer-overview
- Describe cloud computing: https://learn.microsoft.com/training/modules/describe-cloud-compute/
- Azure documentation: https://learn.microsoft.com/azure/
- Pricing calculator: https://azure.microsoft.com/pricing/calculator

2. Usługa WCF

Zbudowanie usługi WCF w środowisku Azure wymaga jedynie dodania do projektu roli WCF Service Role (prawym przyciskiem myszy na projekcie, New Web Role Project... i WCF Web Service Role (lub dodania jej od razu podczas tworzenia projektu).



Serwis tworzymy tak samo, jak serwis hostowany na poprzednich laboratoriach. Możemy go także przetestować narzędziem WCFTestClient. Szczegóły dotyczące endpointu możemy ustawić wybierając właściwości naszej roli WCF (w poddrzewie Roles naszego projektu w Solution Explorer) i przechodząc do zakładki Endpoints.

3. Tabele

Tabele są pojemnikami na nierelacyjne dane, zawierające jednak pewną strukturę. Tabela jest kolekcją encji, zaś encja (odpowiednik wiersza tabeli w SQL) jest kolekcją właściwości w postaci par klucz-wartość. Tabela nie wymusza struktury – każda encja w tabeli może mieć inny zbiór kluczy. Encja może posiadać do 252 właściwości. Jej całkowity rozmiar nie może przekraczać 1MB. Każda encja identyfikowana jest przez klucz główny (RowKey, typu String), który musi być unikatowy w ramach węzła. Dodatkowo posiada także klucz partycji (PartitionKey, typu String)
– encje o takim samym kluczu partycji przechowywane będą w tym samym węźle (maszynie) oraz znacznik czasu (Timestamp). Wartości przypisane do kluczy mogą mieć następujące typy: Binary (tablica bajtów, maksymalnie 64KB), Bool, DateTime, Double, GUID, Int, Int64, String (maksymalnie 64KB).

Dostęp do tabel możliwy jest poprzez Storage Account. Wszystkie tabele zawarte w jednym Storage Account nie mogą przekraczać łącznie 100TB.

Aby skorzystać z tabeli, musimy dodać referencje do `Azure.Data.Tables` poprzez „Manage NuGetPackages...” lub wydać polecenie `install-package Azure.Data.Tables` w konsoli menedżera pakietów. Następnie musimy zdefiniować klasę, która będzie reprezentowała nasze dane. Powinna ona dziedziczyć po klasie `ITableEntity` oraz posiadać bezparametrowy konstruktor. Pola, które mają być zapamiętane w tabeli muszą być właściwościami dostępnymi publicznie (także do zapisu).

```
public class TestEntity : ITableEntity
{
    public TestEntity(string rk, string pk)
    {
        this.PartitionKey = pk; // ustawiamy klucz partycji
        this.RowKey = rk;       // ustawiamy klucz główny
    }
    public TestEntity() { }
    public string s { get; set; }
    public int i { get; set; }
    public string PartitionKey { get; set; }
    public string RowKey { get; set; }
    public DateTimeOffset? Timestamp { get; set; }
    public ETag ETag { get; set; }
}
```

Dostęp do konkretnej tabeli uzyskujemy poprzez referencję:

```
// w przypadku „rzeczywistego” Azure
```

```
var storageAccountName = "konto";
string tableUri = $"https://{storageAccountName}.table.core.windows.net";
var sc1 = new TableServiceClient(new Uri(tableUri), new DefaultAzureCredential());
```

Klasa `DefaultAzureCredential` wymaga pakietu `Azure.Identity`.

```
// w przypadku emulatora
var sc1 = new TableServiceClient("UseDevelopmentStorage=true");
TableClient c1 = sc1.GetTableClient(tableName);
c1.CreateIfNotExists();
```

Nazwa tabeli musi zaczynać się od litery. Może ona zawierać od 3 do 63 znaków, wyłącznie alfanumerycznych. Wielkość liter nie jest brana pod uwagę. Więcej informacji tutaj. Podejrzeć zawartość utworzonych tabel możemy w „Server Explorer" z menu View i rozwijając drzewo „Windows Azure Storage".

Dodanie nowej encji do tabeli wymaga utworzenia operacji wstawienia, przekazania jej encji do wstawienia i wykonania operacji na tabeli:

```
var te = new TestEntity(testData.pk, testData.rk);
c1.AddEntity(t);
```

Operacje wstawiania możemy łączyć akcje dodania (i nie tylko) w większe grupy. Wykonanie takiej grupy operacji ma postać:

```
var actions = new List<TableTransactionAction>();
actions.Add(
    new TableTransactionAction(TableTransactionActionType.Add, new TestEntity
    {
        PartitionKey = pKey,
        RowKey = rKey
    })
            ...
    );
c1.SubmitTransactionAsync(actions);
```

Podobnie wygląda pobranie danych z tabeli w przypadku pojedynczej encji (w przypadku braku szukanej encji Result będzie miało wartość null):

```
string pKey = "kluczpartycji";
string rKey = "kluczwiersza";
Response<TestEntity> result = c1.GetEntity<TestEntity>(pKey, rKey);
TestEntity e = result.Value;
```

lub wybranych encji:

```
string q1 = TableQuery.CombineFilters(
        TableQuery.GenerateFilterCondition("PartitionKey", QueryComparisons.Equal,
pkey),
        TableOperators.And,
        TableQuery.GenerateFilterCondition("RowKey",
        QueryComparisons.Equal, rkey));
AsyncPageable<TestEntity> queryResultsMaxPerPage = c1.QueryAsync<TestEntity>(filter:
q1, maxPerPage: 10);
```

Wyświetlenie z wykorzystaniem paginacji:

```
await foreach (Page<TestEntity> page in queryResultsMaxPerPage.AsPages())
{
    Console.WriteLine("This is a new page!");
```

```
    foreach (TestEntity qEntity in page.Values)
    {
        Console.WriteLine($"{qEntity.PartitionKey}, {qEntity.RowKey}");
    }
}
```

Dostępne są także operacje usuwania (`DeleteEntity`), modyfikacji (`UpdateEntity<T>`), dodania lub modyfikacji (`UpsertEntity<T>`). Wszystkie one pobierają encję jako argument.

Pozycje źródłowe:

- Azure Table storage documentation – https://learn.microsoft.com/azure/storage/tables/
- Naming rules and restrictions for Azure resources – https://learn.microsoft.com/azure/azure-resource-manager/management/resource-name-rules
- Transactions and Batches in Azure Table Storage – https://medium.com/@zilberberg.vadim/transactions-and-batches-in-azure-table-storage-849ba1dc72d6

4. BLOBy

BLOBy (Binary Large Object) umożliwiają przechowywanie w ramach Storage dużych danych binarnych. Rozmiar BLOBa blokowego jest ograniczony przez 200GB, maksymalny rozmiar BLOBa stronicowanego (który jest efektywniejszy w przypadku częstych modyfikacji zawartości) to 1TB. Dostęp do BLOBów jest możliwy jest poprzez Storage Account. Tak jak encje przechowywane są w tabelach, tak BLOBy podzielone są na kontenery. Nazwa kontenera powinna:

- zwierać tylko małe litery, cyfry i minusy,
- nie może rozpoczynać się ani kończyć minusem ani zawierać dwóch kolejnych minusów,
- mieć długość od 3 do 63 znaków.

Nazwy BLOBów z kolei, mogą mieć od 1 do 1024 znaków i zawierać wszystkie znaki dostępne w adresach URL. Nie powinny jednak kończyć się kropką, ukośnikiem (/) ani żadną ich kombinacją. Nie powinny też zawierać odwróconego ukośnika (\ ).

Dostęp do danych mamy poprzez strumienie – zapisując dane do BLOBa podajemy strumień z danymi, odczytując podajemy strumień, do którego dane mają zostać zapisane.

```
// account utworzone jak w przypadku tabel
CloudBlobClient client = account.CreateCloudBlobClient();
CloudBlobContainer container = client.GetContainerReference(nazwaKontenera);
container.CreateIfNotExists();

// pobranie referencji do blokowego BLOBa
var blob = container.GetBlockBlobReference(nazwaBloba);
// null gdy nie istnieje

// zapisanie strumienia do BLOBa
var bytes = new System.Text.ASCIIEncoding().GetBytes(value));
var s = new System.IO.MemoryStream(bytes);
blob.UploadFromStream(s);
// odczytanie bloba do strumienia var s2 = new
System.IO.MemoryStream(); blob.DownloadToStream(s2); string content
= System.Text.Encoding.UTF8.GetString(s2.ToArray());

// usunięcie
blob.Delete();
blob.DeleteIfExists();

// lista BLOBów
string lst = "";
foreach (IListBlobItem item in container.ListBlobs(null, false))
```

```
{     if (item.GetType() ==
typeof(CloudBlockBlob))
    {
        CloudBlockBlob blob = (CloudBlockBlob)item;         lst +=
String.Format("Block blob of length {0}: {1}",
blob.Properties.Length, blob.Uri) + "\n";
    }     else if (item.GetType() ==
typeof(CloudPageBlob))
    {
        CloudPageBlob pageBlob = (CloudPageBlob)item;         lst +=
String.Format("Page blob of length {0}: {1}",
pageBlob.Properties.Length, pageBlob.Uri) + "\n";
    }     else if (item.GetType() ==
typeof(CloudBlobDirectory))
    {
        CloudBlobDirectory   directory   =   (CloudBlobDirectory)item;
lst += String.Format("Directory: {0}", directory.Uri) + "\n";     }
}
```

Pozycje źródłowe:

How to use the Windows Azure Blob Storage Service in .NET https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/

Understanding Block Blobs and Page Blobs
http://msdn.microsoft.com/en-us/library/windowsazure/ee691964.aspx

5. Kolejki

Kolejki są kolejnym mechanizmem przechowywania lub przekazywania danych w ramach Storage Account. Kolejka jest typową kolejką FIFO (First In First Out), w której przechowujemy wiadomości. Zawartością wiadomości może być napis lub tablica bajtów. Rozmiar pojedynczej wiadomości jest ograniczony przez 64KB zaś liczbę wiadomości ogranicza limit rozmiaru Storage (100TB). Korzystanie z kolejki wpisuje się w schemat wyznaczony przez tabele i BLOBy – pobieramy referencję a następnie z niej korzystamy. Dodatkowo, potrzebujemy jeszcze przestrzeni nazw Microsoft.WindowsAzure.Storage.Queue. Nazwa kolejki ma takie same ograniczenia jak nazwa kontenera.

```
// account utworzone jak pooprzednio
CloudQueueClient client = account.CreateCloudQueueClient();
CloudQueue queue = client.GetQueueReference(nazwaKolejki);

// utworzenie kolejki
queue.CreateIfNotExists();

// usunięcie kolejki
queue.Delete();
queue.DeleteIfExists(); // wysłanie
wiadomości
var msg = new CloudQueueMessage("wiadomość");
queue.AddMessage(msg);
```

Zerknięcie (ang. *peek*) jest operacją zwracającą nam pierwszą wiadomość w kolejce. Wiadomość ta nie jest jednak usuwana. Jeżeli kolejka jest pusta, rezultatem będzie null.

```
// zerknięcie na wiadomość var
msg2 = queue.PeekMessage();
```

Odebranie wiadomość jest operacją dwufazową. Metoda GetMessge zwraca nam wiadomość (lub null gdy kolejka jest pusta), Operacja ta powoduje, że wiadomość staje się chwilowo (domyślnie przez 30 sekund, czas ten można zmodyfikować dodając parametr do GetMessage) niewidoczna. W tym czasie należy wiadomość usunąć, inaczej środowisko stwierdzi, że obsługa wiadomości nie powiodła się i wiadomość wróci do kolejki (aby można było ją jeszcze raz obsłużyć).

```
// odebranie wiadomości var
msg3 = queue.GetMessage();
queue.DeleteMessage(msg3);

// odebranie grupy wiadomości wiadomości
IEnumerable<CloudQueueMessage> lst = queue.GetMessages(10); // liczba wiadomości
foreach(var m in lst) queue.DeleteMessage(m);
```

Wiadomości posiadają następujące pola:

AsString - zawartość wiadomości jako napis
AsBytes - zawartość wiadomości jako tablica bajtów
ExpirationTime - czas życia wiadomości (kiedy przestanie być aktualna)
InsertionTime - czas wstawienia do kolejki
DequeueCount - liczba pobrań wiadomości z kolejki

Wiadomości można także modyfikować. Aby to zrobić, pobieramy wiadomość z kolejki i zamiast ją usunąć, wołamy metodę UpdateMessage:

```
CloudQueueMessage msg = queue.GetMessage();
msg.SetMessageContent("nowa treść");
queue.UpdateMessage(message,
                    TimeSpan.FromSeconds(0.0),
                    MessageUpdateFields.Content
                    | MessageUpdateFields.Visibility);
```

Pozycje źródłowe:

How to use the Queue Storage Service
https://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-queues/

6. Worker Role

Worker Role (pracownik) jest obiektem wykonującym pracę lub obliczenia. Dodanie Worker Role do projektu wymaga kliknięcia na poddrzewie Roles naszego projektu i wybraniu Add -> New Worker Role Project... Otrzymamy w ten sposób klasę dziedziczącą po RoleEntryPoint i implementującą trzy metody: OnStart, OnStop oraz Run. Metoda OnStart jest wywoływana w momencie uruchomienia pracownika i powinna zwrócić true jeżeli inicjalizacja się powiodła (w przypadku zwrócenia false obiekt jest niszczony). Metoda OnStop wywoływana jest w momencie zatrzymywania pracownika (w przypadku zatrzymania lub aktualizacji usługi, restartu maszyny Azure lub gdy metoda Run się zakończy). Metoda ta powinna zakończyć się szybko lub być gotowa na sytuacje, gdy jej wykonanie zostanie przerwane. Wreszcie metod Run wykonuje właściwą pracę i w zasadzie nie powinna się kończyć (główna pętla powinna mieć postać zbliżoną do while(true) {...}).

Komunikaty diagnostyczne, wypisywane poprzez Trace.WriteLine można obejrzeć klikając prawym przyciskiem myszy na ikonie emulatora Azure w zasobniku systemowym i wybierając „Show Compute Emulator UI" oraz przechodząc do odpowiedniego pracownika.

Pozycje źródłowe:

Build an Application that Runs in a Cloud Service in Windows Azure http://msdn.microsoft.com/en-us/library/windowsazure/hh180152.aspx

Configuring a Windows Azure Project
http://msdn.microsoft.com/en-us/library/windowsazure/ee405486.aspx

7. Azure SQL

Kolejnym dostępnym kontenerem na dane jest relacyjna baza danych. Jest to wariant Silnik bazy danych ma swoje ograniczenia i nie jest to (jak w przypadku emulatora) SQL Server. Tworzenie bazy danych w realnym systemie Azure jest opisane w pozycji „How to use Windows Azure SQL Database in .NET applications". W przypadku emulatora należy uruchomić SQL Server Object Explorer (z menu View). Jeżeli nie jest on dostępny (wykonanie komendy powoduje błąd) trzeba zainstalować aktualizację: z menu "Tools" wybieramy "Extensions and Updates", z zakładki Updates instalujemy "Microsoft SQL Server Update for database tooling" i restartujemy Visual Studio. Następnie w podwęźle (localdb)\v11.0 węzła SQL Server klikamy prawym przyciskiem myszy na Databases i wybieramy Add new database podając w oknie dialogowym nazwę i lokalizację bazy.

Do połączenia się z bazą danych możemy skorzystać z przestrzeni nazw System.Data.SqlClient. Najpierw tworzymy połączenie:

```
SqlConnection c = new SqlConnection();
```

i ustawiamy w nim ConnectionString:

```
c.ConnectionString = "Driver={SQL Server Native Client 11.0};"+
            "Server=(LocalDB)\\v11.0;Database=nazwa;Trusted_Connection=yes;";
```

lub (w zależności od wersji sterowników bazy danych):

```
c.ConnectionString =
      "Data Source=(LocalDB)\\v11.0;Database=nazwa;Trusted_Connection=yes;";
```

gdzie *nazwa* jest nazwą utworzonej bazy danych. Uwaga: w realnym środowisku Azure ConnectionString będzie miał postać:
Driver={SQL Server Native Client 10.0};
Server=tcp:xxxxxxxxxx.database.windows.net;
Database=testDB;
Uid=MyAdmin@xxxxxxxxxx;
Pwd=pass@word1;Encrypt=yes;

Po otrzymaniu połączenia otwieramy je metodą Open() (na koniec należy je zamknąć metodą Close()). Wykonanie zapytania wymaga utworzenia obiektu SqlCommand wołając CreateCommand na połączeniu. W tak utworzonym obiekcie ustawiamy pole CommandText na treść zapytania SQL i wykonujemy je jedną z trzech metod:

ExecuteNonQuery	- jeżeli zapytanie nie zwraca wyniku (np. CREATE TABLE, INSERT itp.) wartością zwróconą jest liczba zmodyfikowanych wierszy lub -1 w przypadku np. tworzenia tabeli
ExecuteScalar	- jeżeli interesuje nas tylko jedna pierwsza wartość pierwszego wiersza wyniku wartością zwróconą jest object reprezentujący zwróconą wartość lub null w przypadku jego braku
ExecuteReader	- gdy chcemy otrzymać pełny wynik wartością zwróconą jest obiekt typu SqlDataReader; posiada on metodę Read przechodzącą do następnego wiersza oraz metody Get...(*num*) zwracające wartość kolumny o numerze *num* (licząc od zera) w postaci podanego typu (GetString(0), GetGuid(0), GetInt32(0) itp.)

np.:

```
c.Open()
var cmd = c.CreateCommand(); cmd.CommandText = "insert into tabela
values(1, 'test')"; cmd.ExecuteNonQuery();            cmd.CommandText
= "select count(id) from tabela"; int num =
(Int32)cmd.ExecuteScalar(); cmd.CommandText = "select value, text from
tabela";            var rc = cmd.ExecuteReader(); string rv = "";
while(rc.Read()) rv += rc.GetString(0) + " " + rc.GetString(1) + "\n";
c.Close();
```

Transakcje możemy przeprowadzać w następujący sposób:

```
var t = conn.BeginTransaction();
cmd.Transaction = transaction;
cmd.CommandText = ...;
cmd.Execute...(); if(/* OK */)
t.Commit(); else t.Rollback();
```

Jeżeli potrzebujemy automatycznie generowanych identyfikatorów, należy wybrać uniqueidentifier jako typ kolumny klucza oraz ustawić domyślną wartość na newid(). Kolumna taka będzie miała typ Guid.

Dodatkowa literatura:
SQL Database documentation http://azure.microsoft.com/en-us/documentation/services/sql-database/

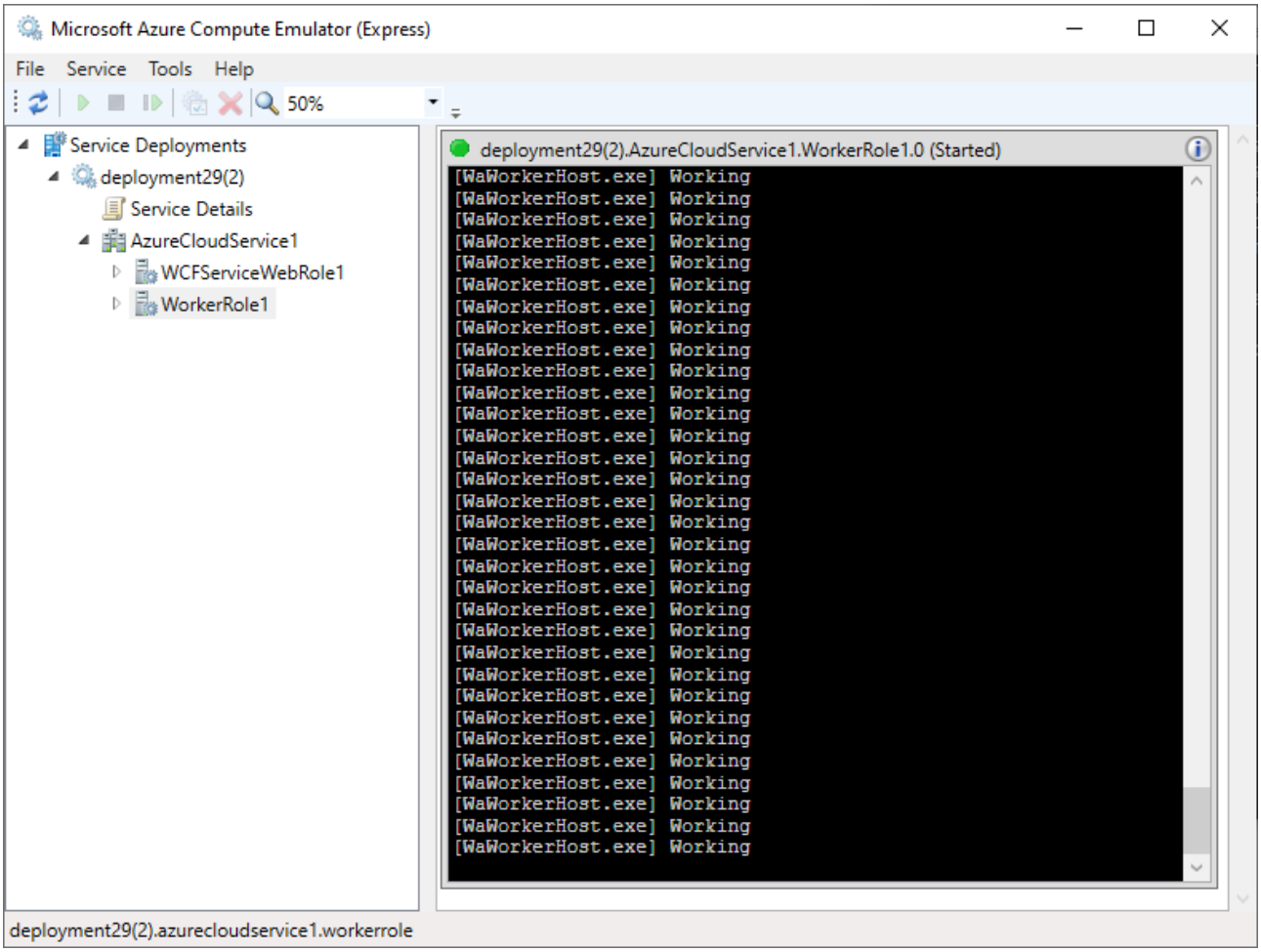
Windows Azure SQL Database http://msdn.microsoft.com/en-us/library/windowsazure/ee336279.aspx

How to use Windows Azure SQL Database in .NET applications https://www.windowsazure.com/en-us/develop/net/how-to-guides/sql-database/?fb=pl-pl#using-ODBC

Guidelines and Limitations (Windows Azure SQL Database) http://msdn.microsoft.com/en-us/library/windowsazure/ff394102.aspx

8. Błędy

Nie można połączyć się z serwerem zdalnym sprawdzić w zasobniku systemowym czy Compute i Storage Emulator jest uruchomiony. W przypadku poprawnie działającego Compute emulatora w oknie „Show Compute Emulator UI” powinniśmy otrzymać podobny do poniższego wynik:



Podobnie w przypadku poprawnie działającego Storage emulatora. W oknie „Show Storage Emulator UI" powinniśmy otrzymać podobny do poniższego wynik:

```
Windows Azure Storage Emulator 5.10.0.0 command line tool
Usage:
    AzureStorageEmulator.exe init           : Initialize the emulator database and configuration.
    AzureStorageEmulator.exe start          : Start the emulator.
    AzureStorageEmulator.exe stop           : Stop the emulator.
    AzureStorageEmulator.exe status         : Get current emulator status.
    AzureStorageEmulator.exe clear          : Delete all data in the emulator.
    AzureStorageEmulator.exe help [command] : Show general or command-specific help.

See the following URL for more command line help: http://go.microsoft.com/fwlink/?LinkId=392235

C:\Program Files (x86)\Microsoft SDKs\Azure\Storage Emulator>
```

Nie można znacjonalizować emulatora magazynu Microsoft Azure

sprawdzić w zasobniku systemowym czy Storage Emulator jest uruchomiony. Uruchomienie Storage Emulatora (uprawnienia administratora):

```
AzureStorageEmulator.exe init
AzureStorageEmulator.exe start
```

Specified argument was out of the range of valid values. Parameter name: site

Dodać do rejestru następujące wpisy (uprawnienia administratora):

```
reg add HKLM\Software\WOW6432Node\Microsoft\InetStp /v MajorVersion /t REG_DWORD /d 10 /f
reg add HKLM\Software\Microsoft\InetStp /v MajorVersion /t REG_DWORD /d 10 /f
```