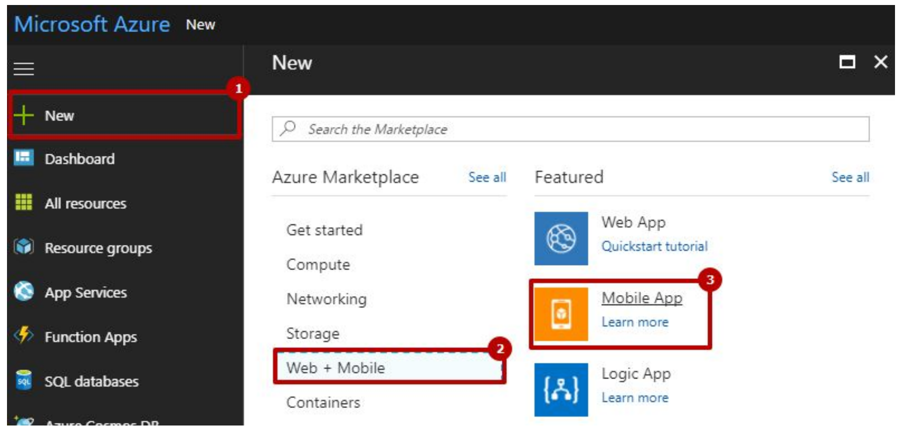
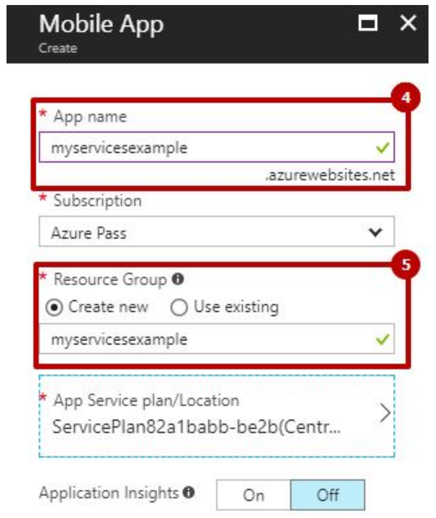
Saving game progress from Unity to the Cloud.





* App name: myservicesexample .azurewebsites.net

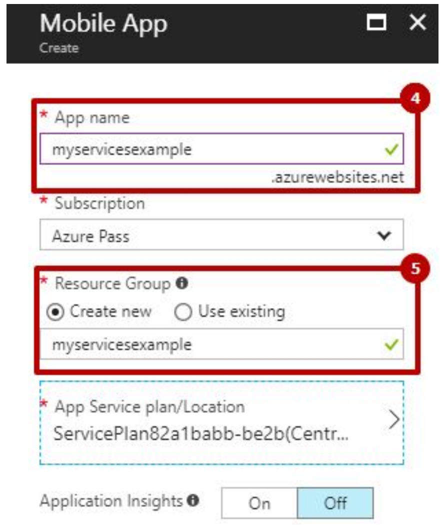
* Subscription: Azure Pass

* Resource Group: [✓ Create new] Use existing

myservicesexample

* App Service plan/Location: ServicePlan82a1babb-be2b(Centr...

Application Insights: On [✓ Off]



* App name: myservicesexample .azurewebsites.net

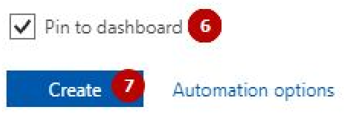
* Subscription: Azure Pass

* Resource Group: [✓ Create new] Use existing

myservicesexample

* App Service plan/Location: ServicePlan82a1babb-be2b(Centr...

Application Insights: On [✓ Off]



✓ Pin to dashboard

Creating services on the cloud and configuring them

Click **New** in the Azure control panel and select **Mobile App** from the list of services.

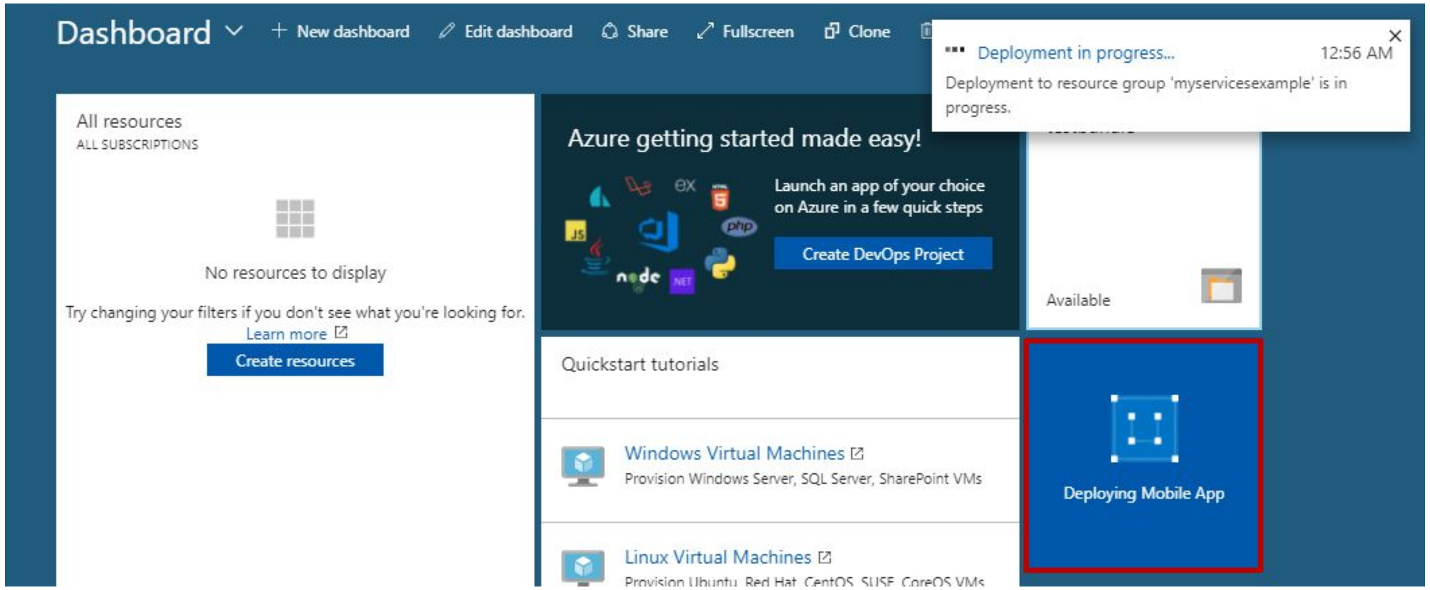
App name - specify a unique name for your service (in this case **myserviceexample**)

Resource Group you must select an existing one or specify a new one.
Within the resource groups, you can create the right groups of applications, services, and databases.

We recommend you to tick the **Pin to dashboard** option so that a link to our service appears on the main page.

Click **Create**.

Saving game progress from Unity to the Cloud.



Deployment in progress... 12:56 AM

Deployment to resource group 'myservicesexample' is in progress.

No resources to display

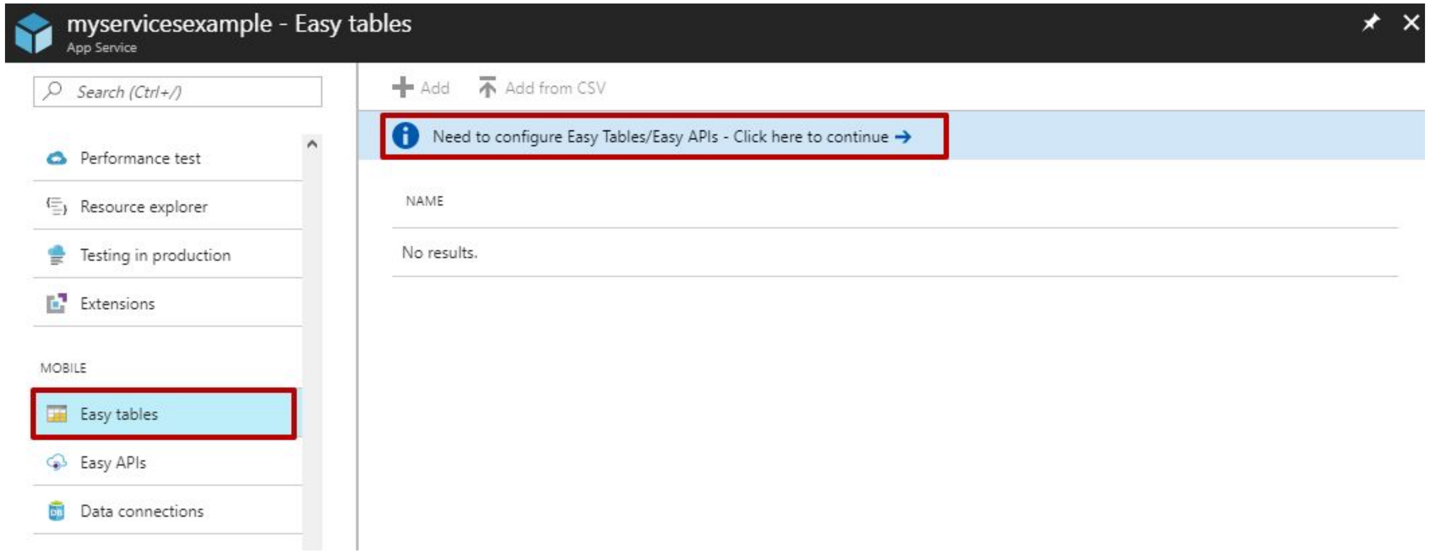
Try changing your filters if you don't see what you're looking for.

Provision Windows Server, SQL Server, SharePoint VMs

Provision Ubuntu, Red Hat, CentOS, SUSE, CoreOS VMs

By clicking the Create button, the process of deploying the service on the cloud will begin.

After the end of this process, the service will be ready for use.

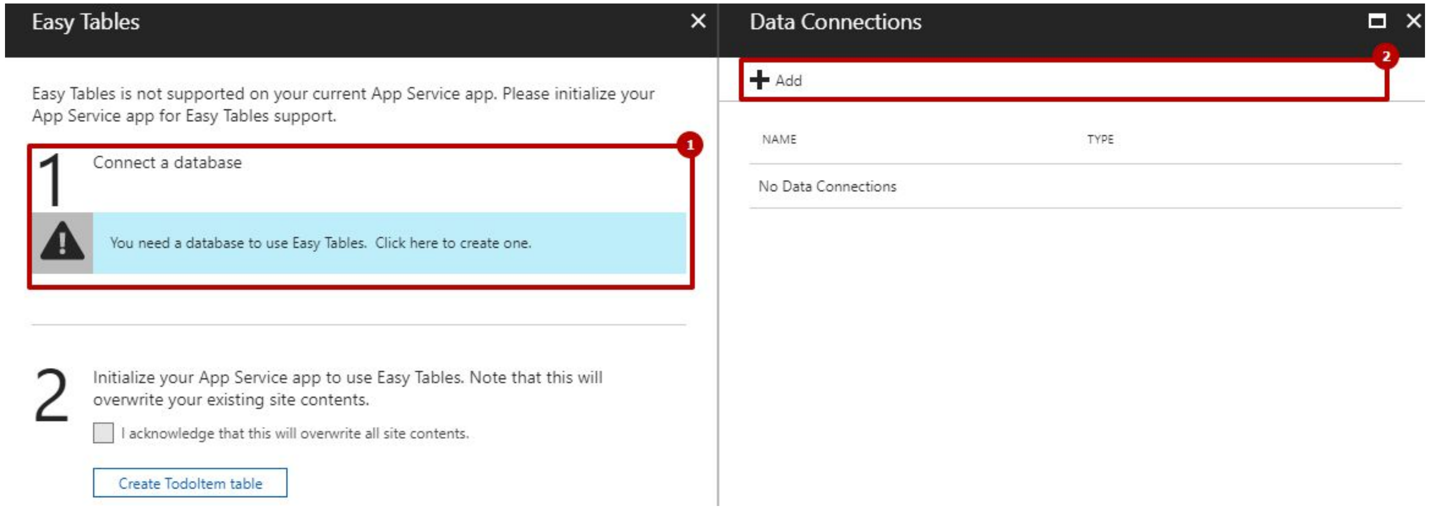


Need to configure Easy Tables/Easy APIs - Click here to continue →

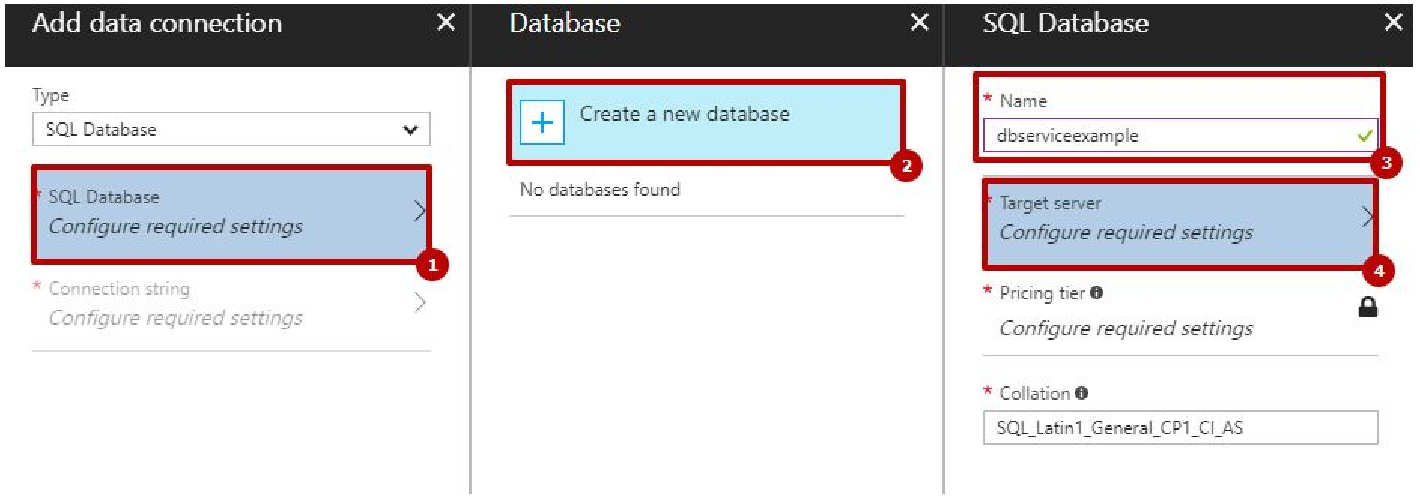
No results.

To store the data of our game we will use **Easy Tables**. First, we need to quickly create an **MS SQL** database in which we will store our tables.

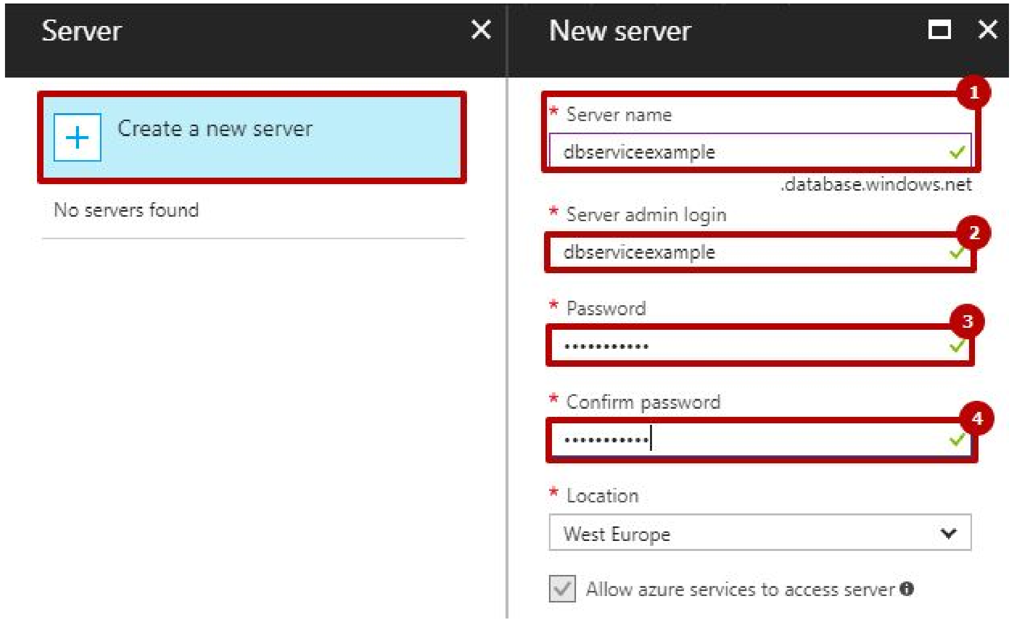
Select in the menu on the left **Easy Tables** and then **Click to continue**, click the arrow to the right, as in the screenshot.


Easy Tables
Easy Tables is not supported on your current App Service app. Please initialize your App Service app for Easy Tables support.
1 Connect a database
You need a database to use Easy Tables. Click here to create one.
2 Initialize your App Service app to use Easy Tables. Note that this will overwrite your existing site contents.
I acknowledge that this will overwrite all site contents.
Create TodoItem table
Data Connections
Add
NAME
TYPE
No Data Connections

Now we will continue setting up our tables, for step **1 Connect** a database, click on the exclamation mark and in the window **Data Connections** click button Add.



In order to create a new connection, we need a database. Click **Create a new database** then enter the name of our database. Important field **Target server**. By default, you will not have a server, so we'll create it later.



Server

Create a new server

No servers found

New server

* Server name

dbserviceexample

.database.windows.net

* Server admin login

dbserviceexample

* Password

••••••••••••

* Confirm password

••••••••••••

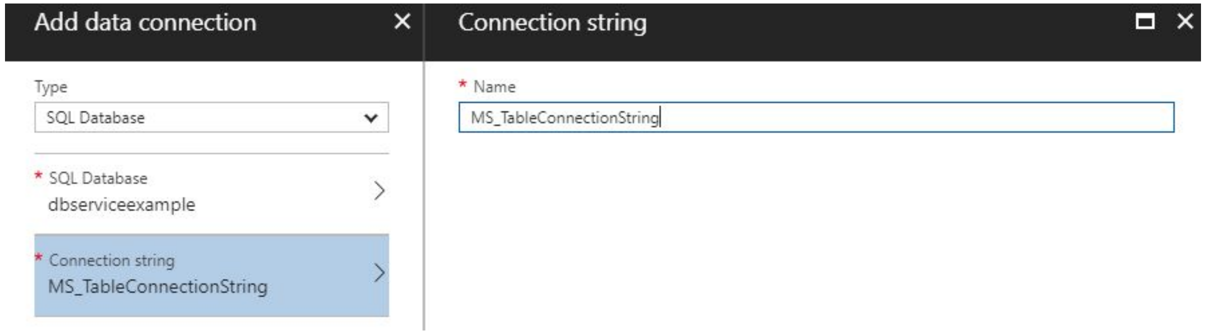
* Location

West Europe

Allow azure services to access server

Click **Create a new server** to create the server.

Server name can be any unique name, then create **Server admin login** (you need to remember it, copy it into a text file) and fill **Password** field - the password for the administrator (you also need to remember it, copy it to a text file) then select the server location and click Select.



Add data connection

Type

SQL Database

* SQL Database

dbserviceexample

* Connection string

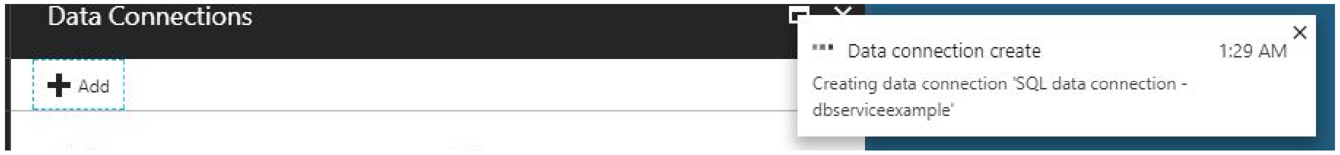
MS_TableConnectionString

Connection string

* Name

MS_TableConnectionString

After that we need to connect to the newly created MS SQL server. In the **Add data connection** window, select **Connection String**. In addition to the name, sometimes the system asks for the administrator's name (we just created it) and the password. (We also just created it and I hope it was not forgotten). We press the button Ok.



Data Connections

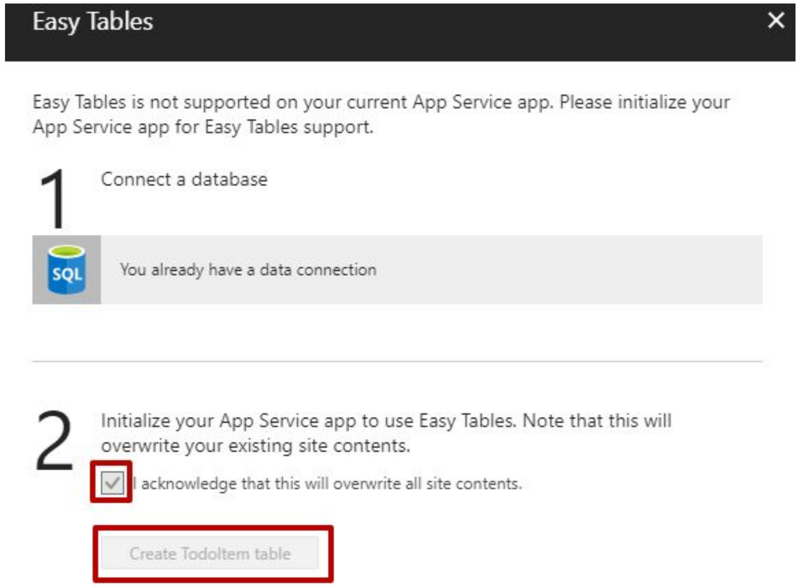
Add

Data connection create 1:29 AM

Creating data connection 'SQL data connection - dbserviceexample'

Now we are waiting for the connection to the database to appear.

Saving game progress from Unity to the Cloud.



Easy Tables

Easy Tables is not supported on your current App Service app. Please initialize your App Service app for Easy Tables support.

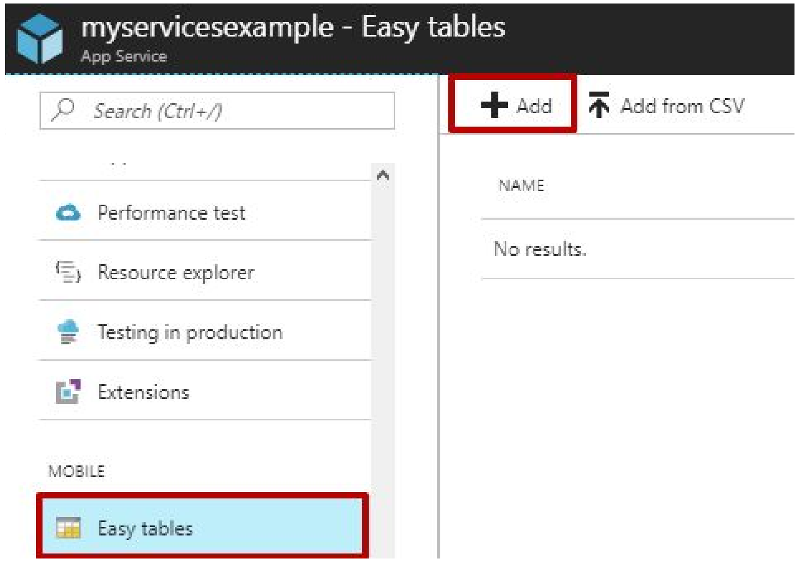
1 Connect a database

You already have a data connection

2 Initialize your App Service app to use Easy Tables. Note that this will overwrite your existing site contents.

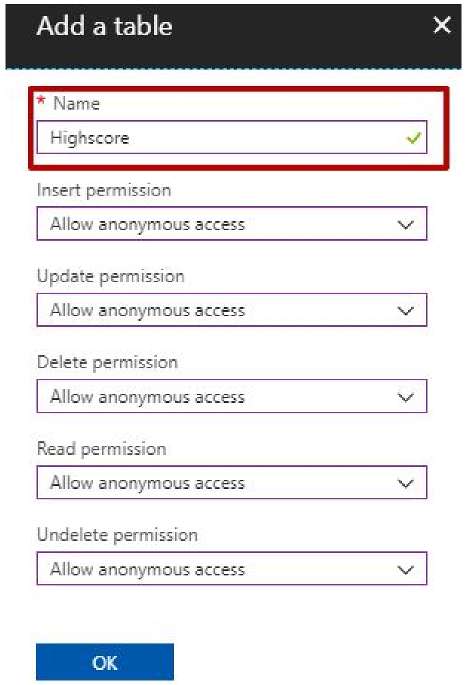
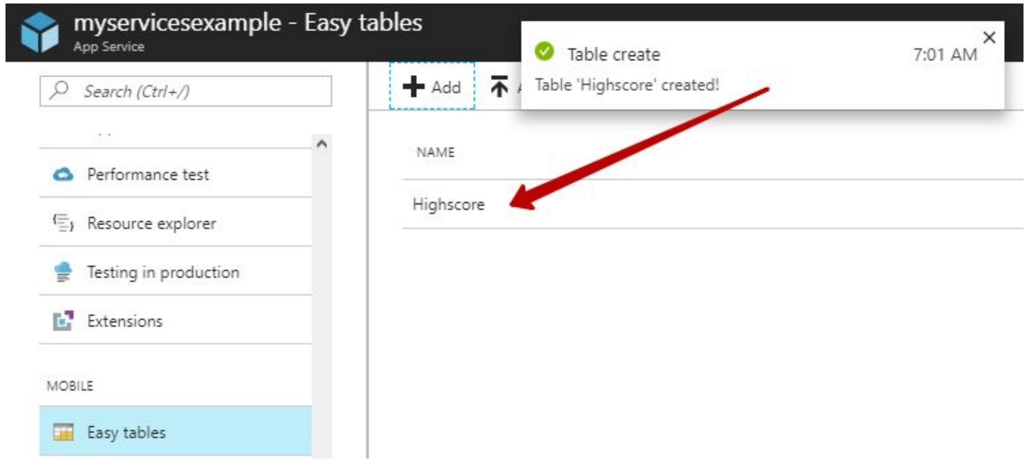
☑ I acknowledge that this will overwrite all site contents.

Now we have to finish the **Easy Tables** setup. Select the checkbox **I acknowledge that this will overwrite all site contents** and click the **Create TodoItem table** button.



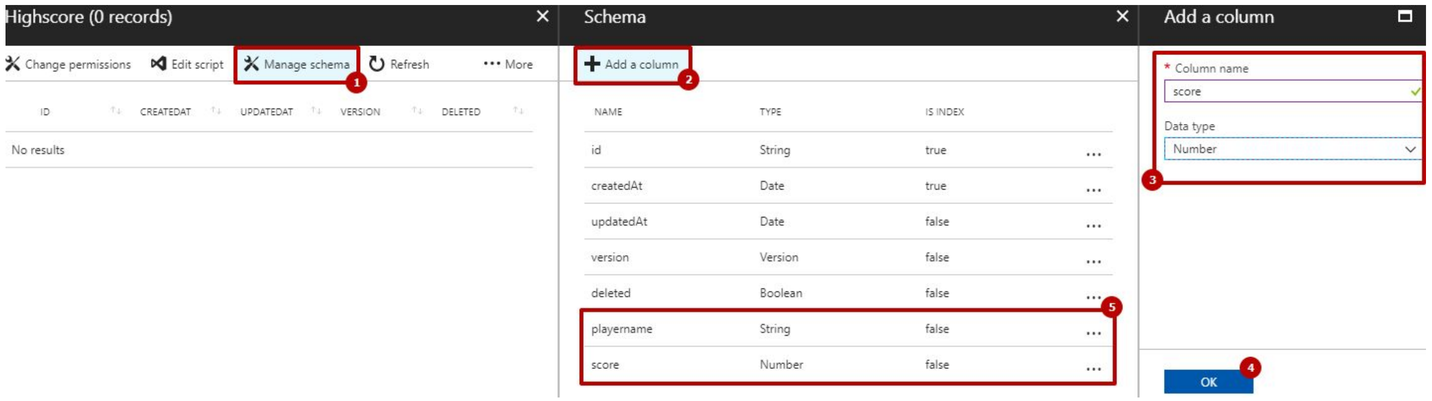
Now we need to create a table to store our data. Select again in the **Easy tables** menu and click the **Add** button.

Saving game progress from Unity to the Cloud.

 Add a table ✕ * Name: Highscore ✓ Insert permission: Allow anonymous access Update permission: Allow anonymous access Delete permission: Allow anonymous access Read permission: Allow anonymous access Undelete permission: Allow anonymous access OK	You will see the **Add a table** menu, in order to avoid confusion, let's create a table named **Highscore**. Then press the Ok button.
 myservicesexample - Easy tables App Service Table create 7:01 AM Table 'Highscore' created! NAME Highscore	After the successful creation of the table, you will see it in the list.

Saving game progress from Unity to the Cloud.

Azure



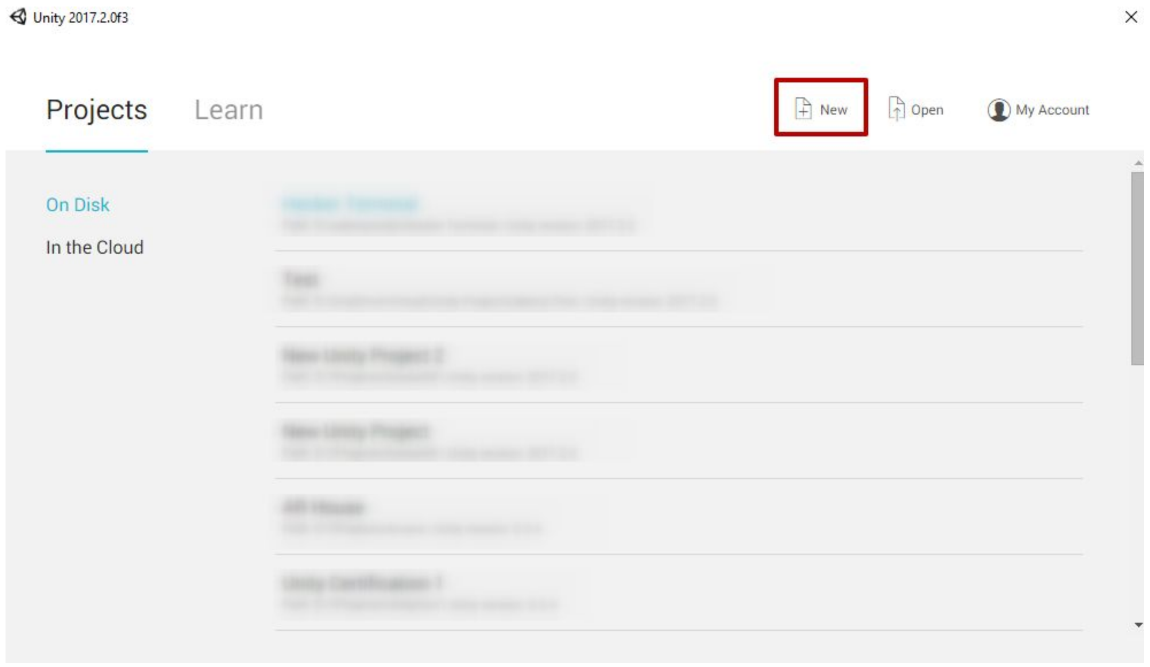
ID	CREATEDAT	UPDATEDAT	VERSION	DELETED
No results				

NAME	TYPE	IS INDEX
id	String	true
createdAt	Date	true
updatedAt	Date	false
version	Version	false
deleted	Boolean	false
playername	String	false
score	Number	false

* Column name: score

Data type: Number

Now we need to configure our table, adding there a couple of fields that we want to use in the future. To do this, click **Manage Schema** on the right will appear the **Schema** window. In it you can see the standard fields that were created by default in the table.
And now we need to add two new fields. Click **Add a column**. The first field we add is called **playername** of type string and the second field is a **score** of type number.



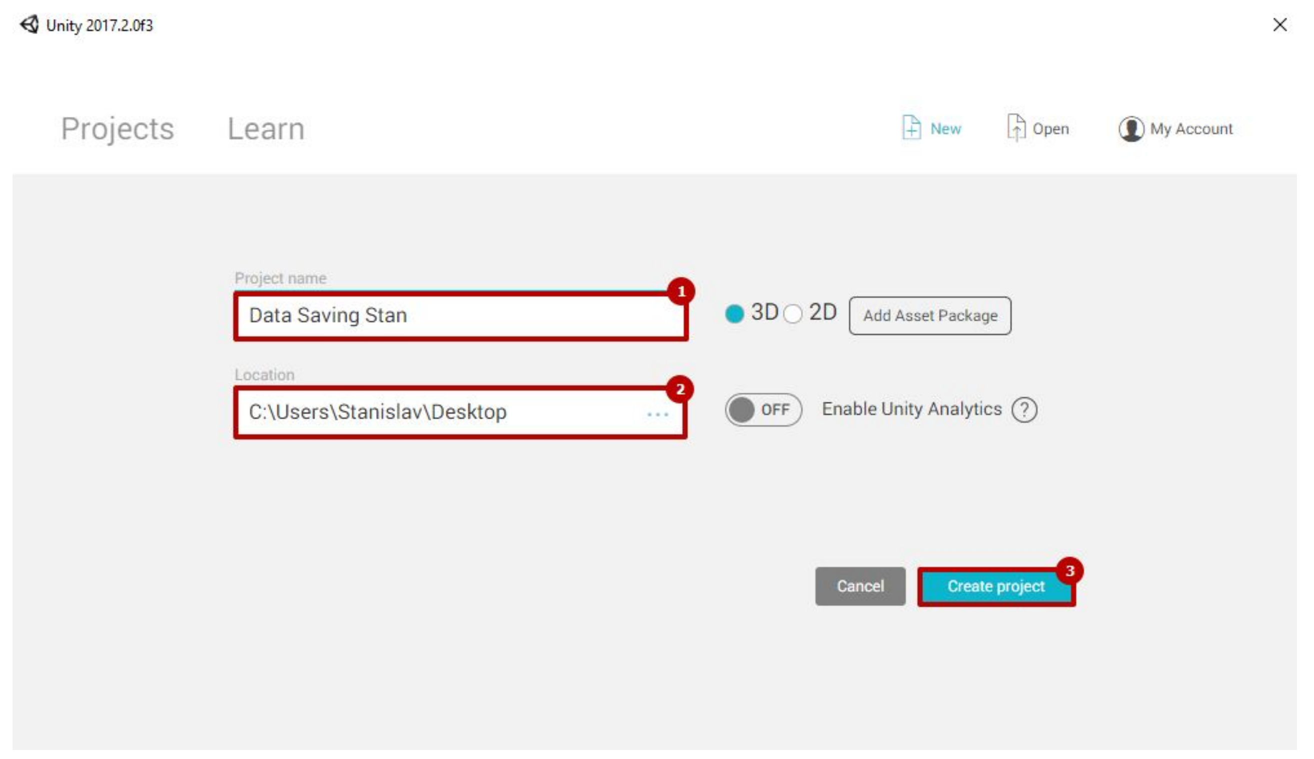
Connecting Services to a New Unity Project

Now let's connect our service to the unity project.
To do this, we need to open Unity and create a new project.

In this project we used version Unity 2017.2.0f3

Saving game progress from Unity to the Cloud.

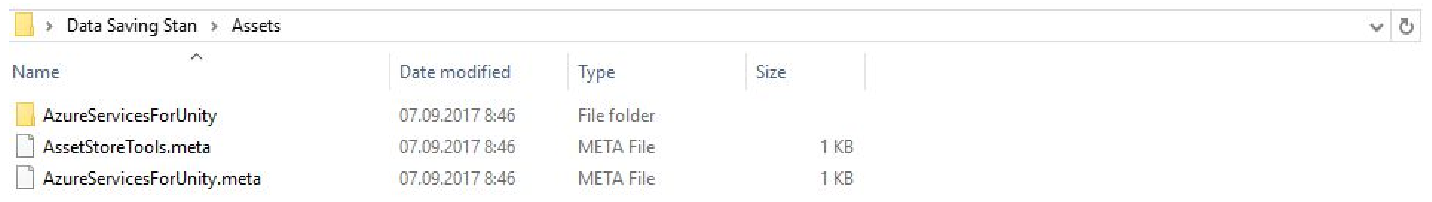
Azure



1 - Project name - enter the unique name of your new project.

2 - Location - specify the directory where the new project will be stored.

3 - Create project Click to start creating a new project.

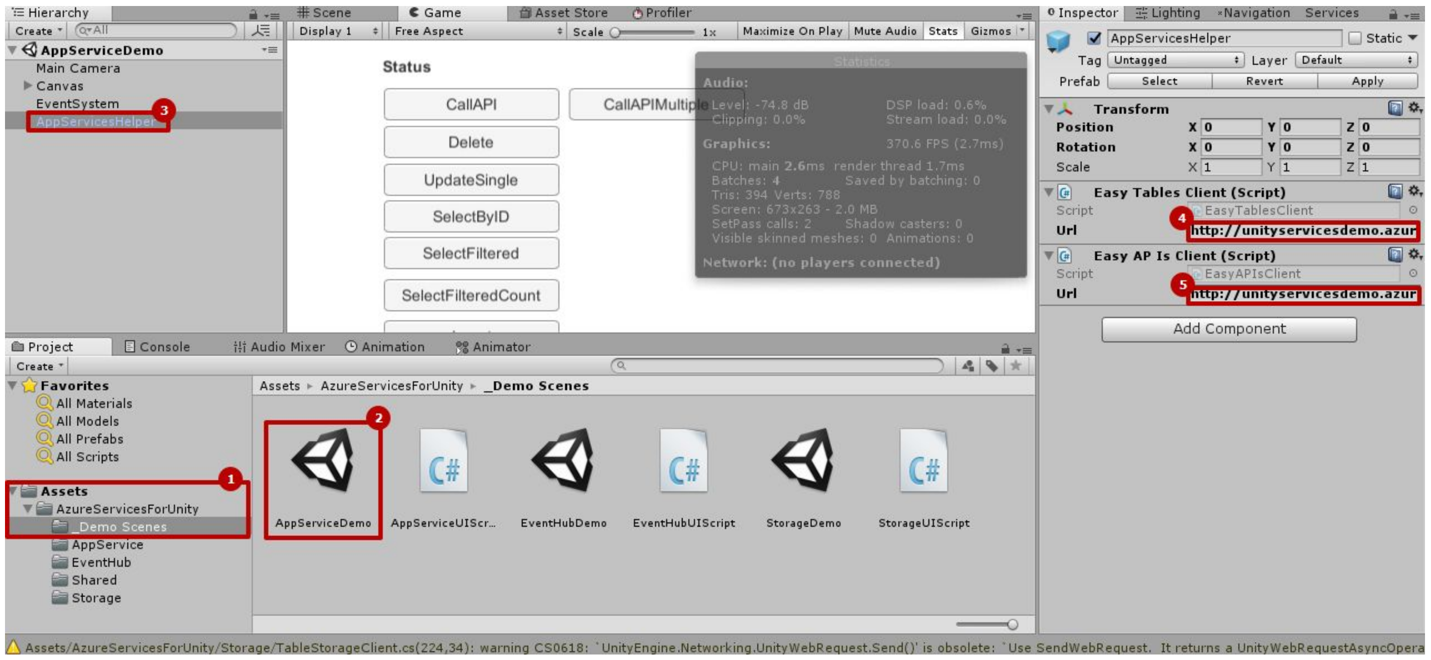


Name	Date modified	Type	Size
AzureServicesForUnity	07.09.2017 8:46	File folder	
AssetStoreTools.meta	07.09.2017 8:46	META File	1 KB
AzureServicesForUnity.meta	07.09.2017 8:46	META File	1 KB

Now we need to download the Assets for working with Azure services in Unity, link to Assets: https://github.com/dgkanatsios/AzureServicesForUnity

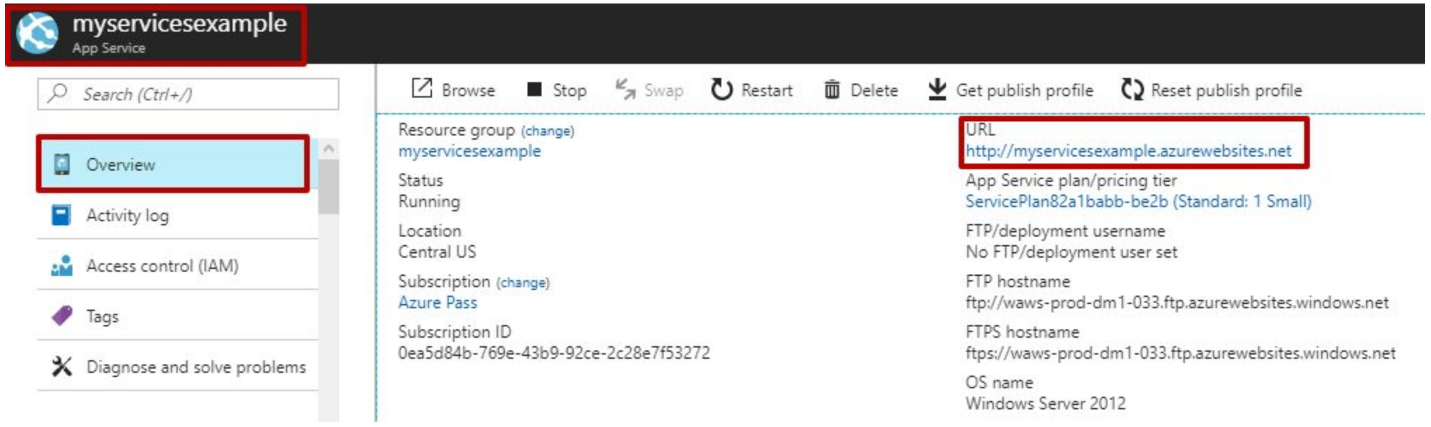
Files from the Assets directory from the archive must be transferred to the **Assets** directory of your new Unity project.

Saving game progress from Unity to the Cloud.



```
Assets/AzureServicesForUnity/Storage/TableStorageClient.cs(224,34): warning CS0618: `UnityEngine.Networking.UnityWebRequest.Send()' is obsolete: `Use SendWebRequest. It returns a UnityWebRequestAsyncOpera
```

After we add the Asset to the project, it must be configured. Open the first demo scene with the name "**AppServiceDemo**", as shown in the screenshot, then select GameObject with the name "**AppServiceManager**" and in the inspector we see two scripts for the settings. We are interested in the **EasyTablesClient** script, we insert the path to our service, which you can take from Azure, as shown below.



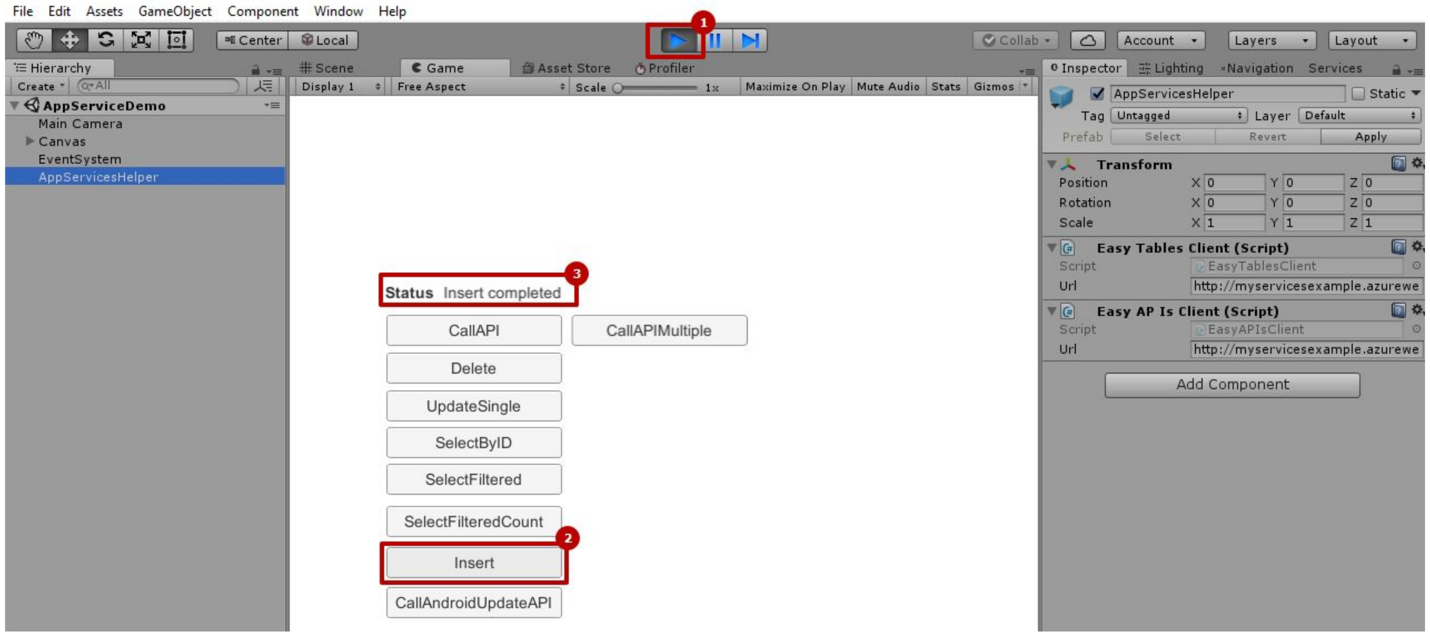
Resource group (change)
myservicesexample
Status
Running
Location
Central US
Subscription (change)
Azure Pass
Subscription ID
0ea5d84b-769e-43b9-92ce-2c28e7f53272

URL
http://myservicesexample.azurewebsites.net
App Service plan/pricing tier
ServicePlan82a1babb-be2b (Standard: 1 Small)
FTP/deployment username
No FTP/deployment user set
FTP hostname
ftp://waws-prod-dm1-033.ftp.azurewebsites.windows.net
FTPS hostname
ftps://waws-prod-dm1-033.ftp.azurewebsites.windows.net
OS name
Windows Server 2012

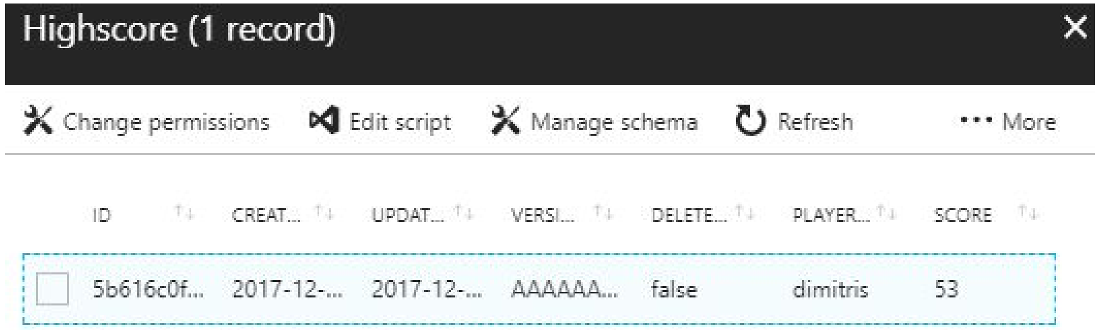
The path to the service you can take by contacting the previously created service **App Service** on the **Overview** tab.

Important: At this moment we are working with a demonstration project. When you work in your game project with Easy Tables, do not forget to put the **EasyTablesClient** script on the scene.

Saving game progress from Unity to the Cloud.

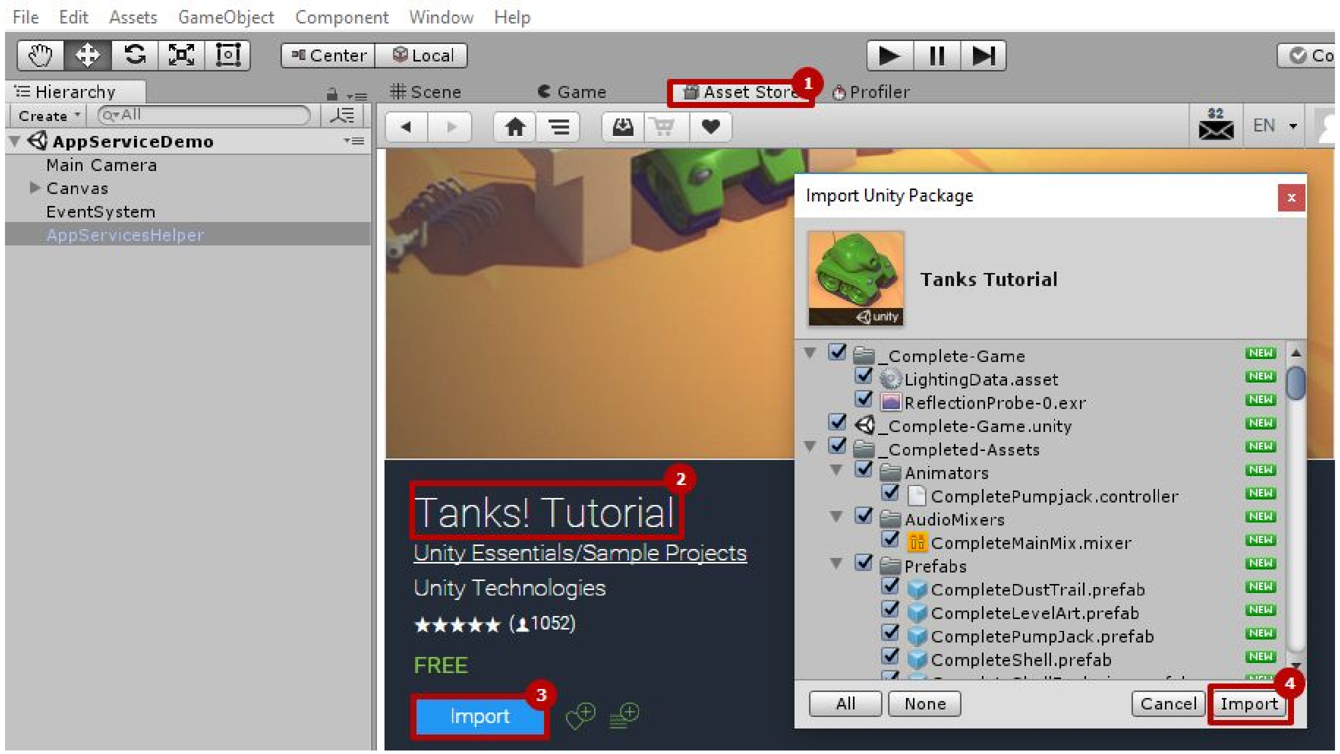


Let's now start our scene and see how it works. After startup, we need to press the **Insert** button and wait until the download status is set to **Insert completed**.

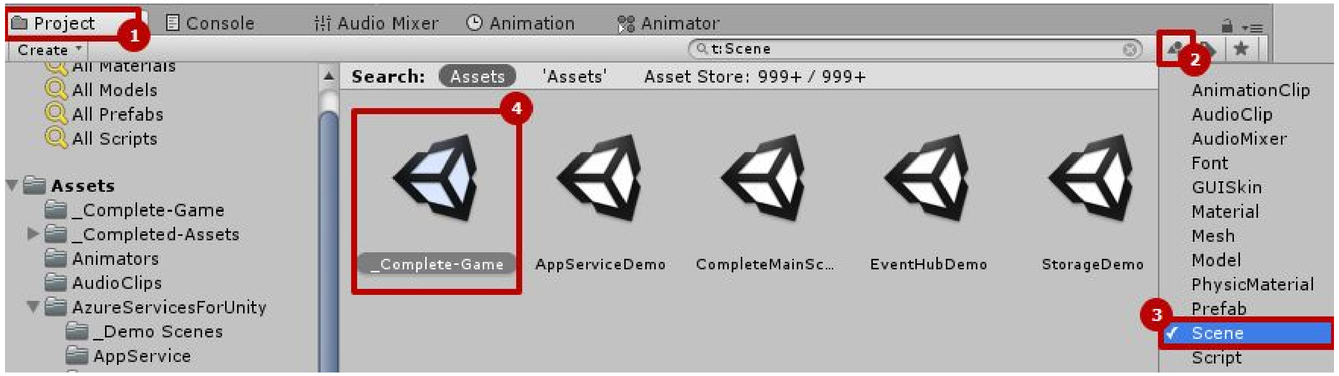


ID	CREAT...	UPDAT...	VERSI...	DELETE...	PLAYER...	SCORE
5b616c0f...	2017-12-...	2017-12-...	AAAAAA...	false	dimitris	53

On the cloud in our **Highscore** table will have to appear a new record.



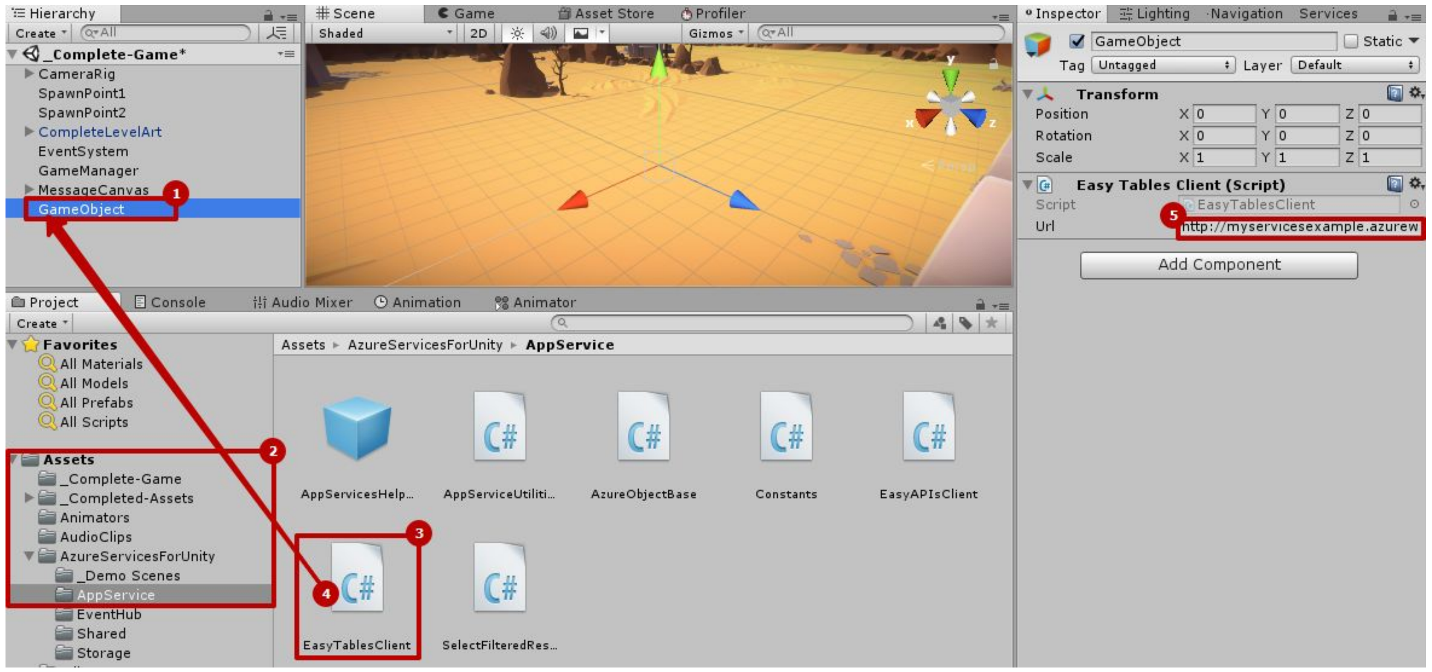
An excellent example of the official free tutorial unity called **Tanks!** Tutorial which we download from the **Unity Asset Store**. The screenshot below shows the sequence of actions to load this game into our project. First we open the **Asset Store** window in it in the search for the new **Tanks! Tutorial**, you will have the **Download** button first, when the Asset is downloaded, the **Import** button appears. Then click another **Import** button and wait for it to load into your project.



Let's open the main scene of our project, called "**_Complete-Game**". To do this, you need to open a filter in the **Project** window and select **Scenes**.

Saving game progress from Unity to the Cloud.

Azure



GameObject

Tag Untagged | Layer Default

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

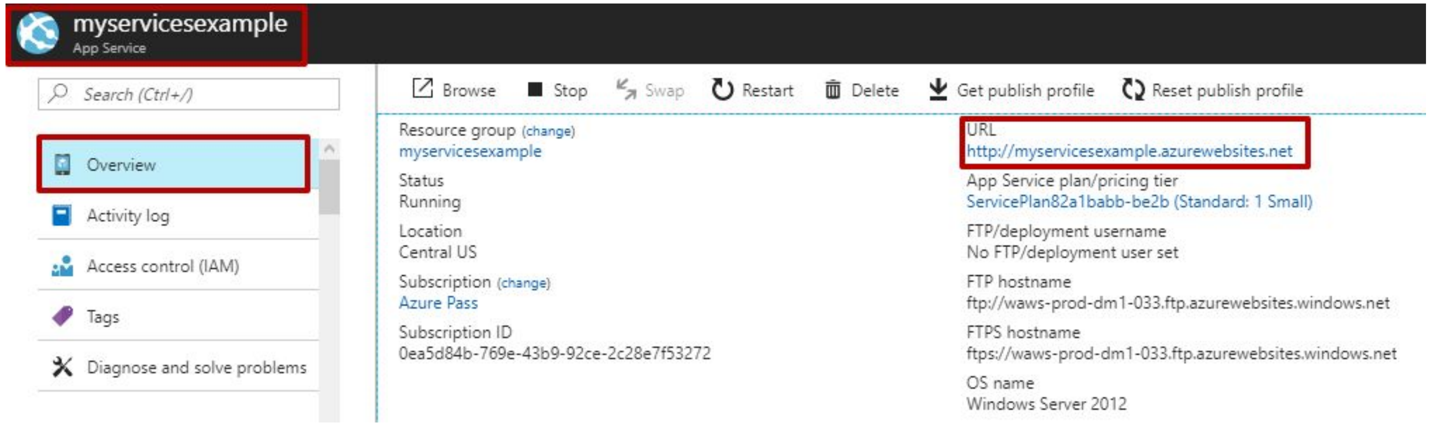
Script EasyTablesClient

Url http://myservicesexample.azurew

In order for our save script to work in Azure, create an empty **GameObject** by right-clicking on the empty area of the **Hierarchy** window on the left side of the Unity

It's time to move the **EasyTablesClient** script to it (you can quickly find it by analogy of the scene search, specifying Scripts as a filter, or find it in the **Assets \ AzureServicesForUnity \ AppService directory**)

And now insert the URL of our service into the **URI** string of our script.



Resource group (change)
myservicesexample

Status
Running

Location
Central US

Subscription (change)
Azure Pass

Subscription ID
0ea5d84b-769e-43b9-92ce-2c28e7f53272

URL
http://myservicesexample.azurewebsites.net

App Service plan/pricing tier
ServicePlan82a1babb-be2b (Standard: 1 Small)

FTP/deployment username
No FTP/deployment user set

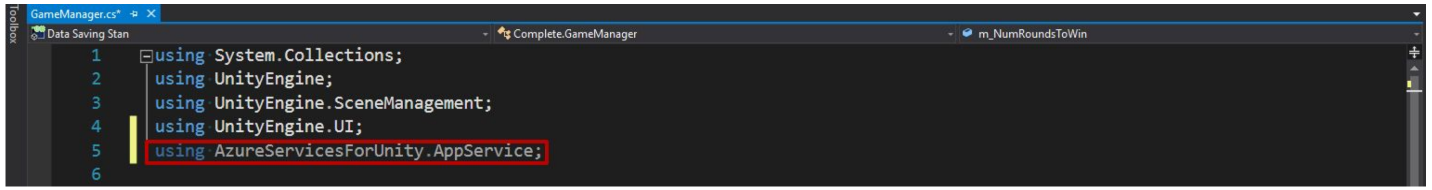
FTP hostname
ftp://waws-prod-dm1-033.ftp.azurewebsites.windows.net

FTPS hostname
ftps://waws-prod-dm1-033.ftp.azurewebsites.windows.net

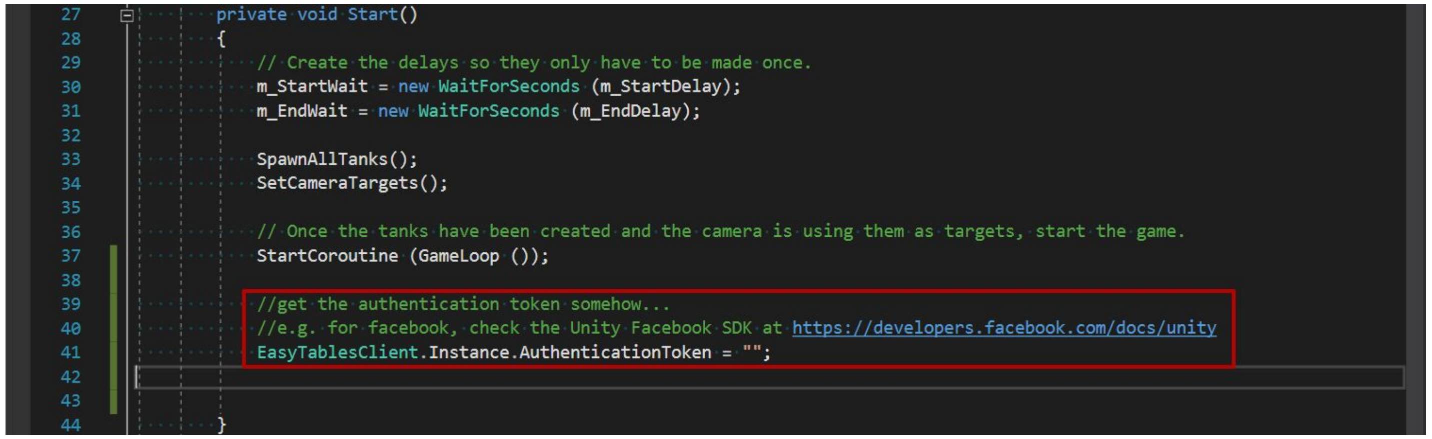
OS name
Windows Server 2012

The address of our service has not changed and you can copy it by analogy with the previous example.

Saving game progress from Unity to the Cloud.



```csharp
using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;
using AzureServicesForUnity.AppService;
```



```csharp
        private void Start()
        {
            // Create the delays so they only have to be made once.
            m_StartWait = new WaitForSeconds (m_StartDelay);
            m_EndWait = new WaitForSeconds (m_EndDelay);

            SpawnAllTanks();
            SetCameraTargets();

            // Once the tanks have been created and the camera is using them as targets, start the game.
            StartCoroutine (GameLoop ());

            //get the authentication token somehow...
            //e.g. for facebook, check the Unity Facebook SDK at https://developers.facebook.com/docs/unity
            EasyTablesClient.Instance.AuthenticationToken = "";


        }
```

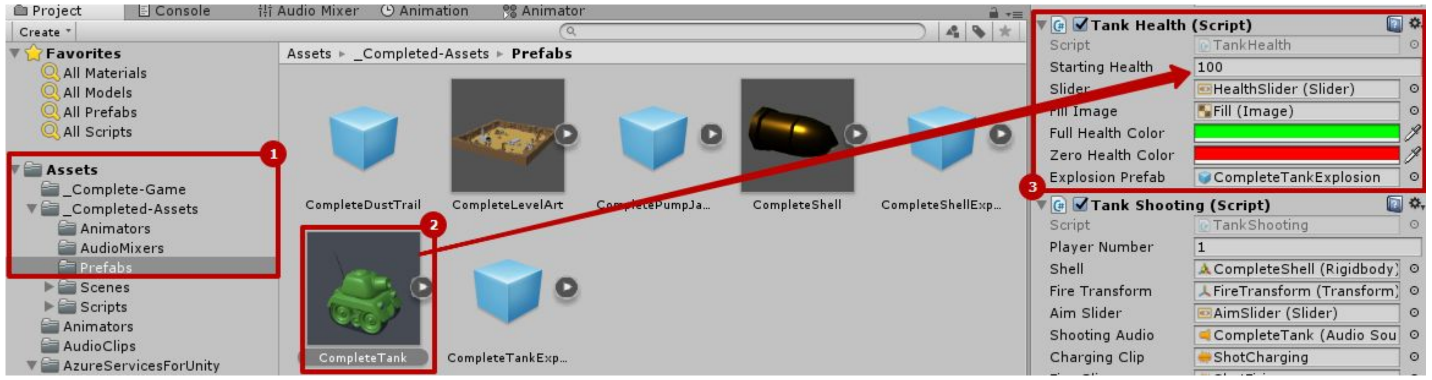
Let's now go into the **GameManager** script (it's in the **Assets \ _Completed-Assets \ Scripts \ Managers directory**) and add a couple of lines associated with the authorization.

We have to specify a namespace **using AzureServicesForUnity.AppService;**

In the current example, we leave this field empty.

Authorization code:

//get the authentication token somehow...
//e.g. for facebook, check the Unity Facebook SDK at https://developers.facebook.com/docs/unity
EasyTablesClient.Instance.AuthenticationToken = "";



Tank Health (Script)

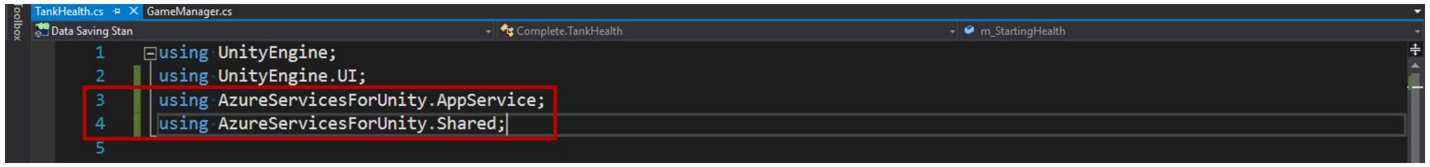
Script	TankHealth
Starting Health	100
Slider	HealthSlider (Slider)
Fill Image	Fill (Image)
Full Health Color	🟩
Zero Health Color	🟥
Explosion Prefab	CompleteTankExplosion

Tank Shooting (Script)

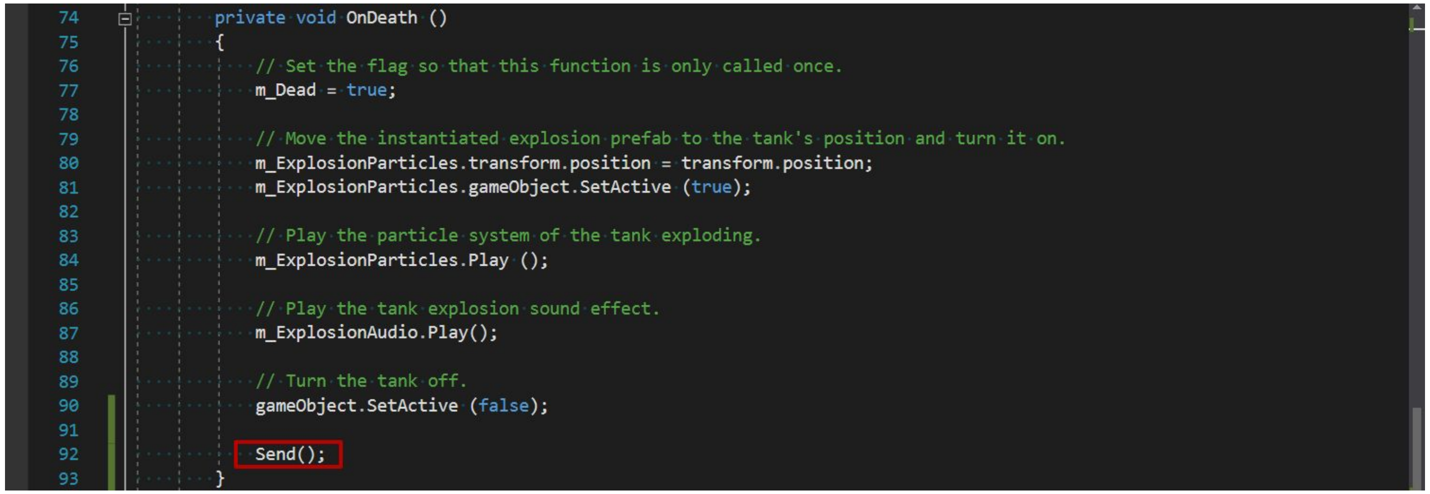
Script	TankShooting
Player Number	1
Shell	CompleteShell (Rigidbody)
Fire Transform	FireTransform (Transform)
Aim Slider	AimSlider (Slider)
Shooting Audio	CompleteTank (Audio Sou
Charging Clip	ShotCharging

In our example of using **EasyTables** we will write in them the winner of the current round. In fact, it's possible to record any important game data such as progress, settings, player profile data. In order to understand which of the tanks won, his health should be zero, based on this we will modify the **TankHealth** script.

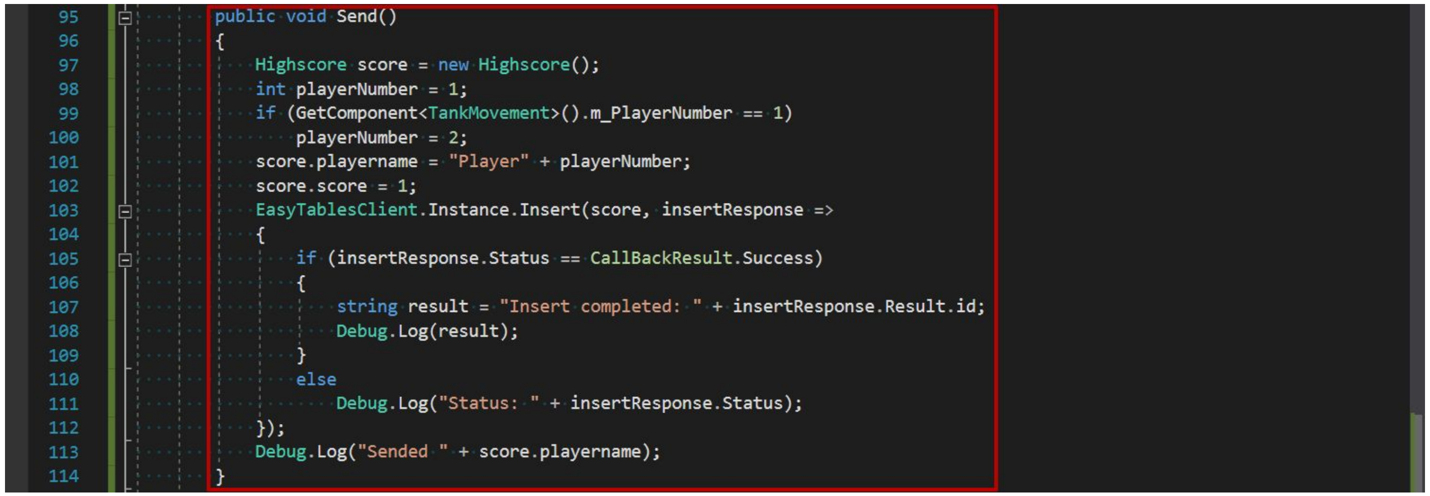
Saving game progress from Unity to the Cloud.



```csharp
using UnityEngine;
using UnityEngine.UI;
using AzureServicesForUnity.AppService;
using AzureServicesForUnity.Shared;
```



```csharp
        private void OnDeath ()
        {
            // Set the flag so that this function is only called once.
            m_Dead = true;

            // Move the instantiated explosion prefab to the tank's position and turn it on.
            m_ExplosionParticles.transform.position = transform.position;
            m_ExplosionParticles.gameObject.SetActive (true);

            // Play the particle system of the tank exploding.
            m_ExplosionParticles.Play ();

            // Play the tank explosion sound effect.
            m_ExplosionAudio.Play();

            // Turn the tank off.
            gameObject.SetActive (false);

            Send();
        }
```

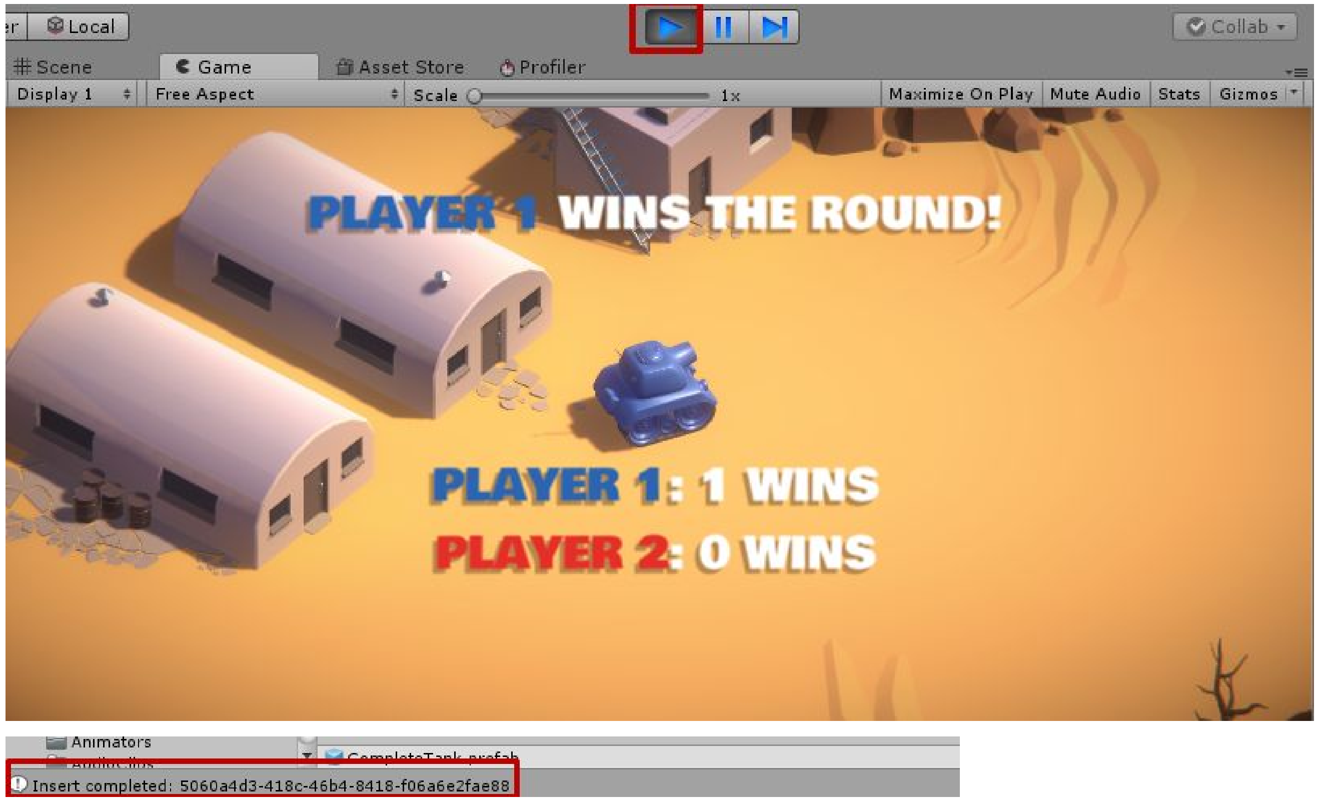


```csharp
        public void Send()
        {
            Highscore score = new Highscore();
            int playerNumber = 1;
            if (GetComponent<TankMovement>().m_PlayerNumber == 1)
                playerNumber = 2;
            score.playername = "Player" + playerNumber;
            score.score = 1;
            EasyTablesClient.Instance.Insert(score, insertResponse =>
            {
                if (insertResponse.Status == CallBackResult.Success)
                {
                    string result = "Insert completed: " + insertResponse.Result.id;
                    Debug.Log(result);
                }
                else
                    Debug.Log("Status: " + insertResponse.Status);
            });
            Debug.Log("Sended " + score.playername);
        }
```

Double-click on the name of the script **TankHealth** in the Script field will open it in the editor. In the **OnDeath ()** method, add the call to the **Send ()** method. This method works on the principle of Insert, after the tank dies, it creates a new record in the database, where it writes the name of the winner and just one in the score

```
public void Send()
        {
            Highscore score = new Highscore();
            int playerNumber = 1;
            if (GetComponent<TankMovement>().m_PlayerNumber
== 1)
                playerNumber = 2;
            score.playername = "Player" + playerNumber;
            score.score = 1;
            EasyTablesClient.Instance.Insert(score, insertResponse
=>
            {
                if (insertResponse.Status == CallBackResult.Success)
                {
                    string result = "Insert completed: " +
insertResponse.Result.id;
                    Debug.Log(result);
                }
                else
                    Debug.Log("Status: " + insertResponse.Status);
            });
            Debug.Log("Sended " + score.playername);
        }
```

Saving game progress from Unity to the Cloud.

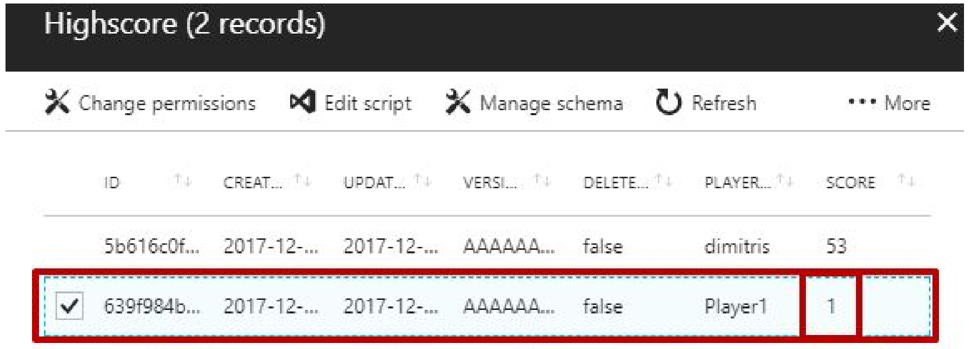


```
Insert completed: 5060a4d3-418c-46b4-8418-f06a6e2fae88
```

After all the changes, do not forget to save the script. Then go to Unity and start the game.

After one of the tanks wins another in our database in the cloud will have to be a new record with the winner.

Unity will display the **Insert completed** status in the bottom corner of the console.



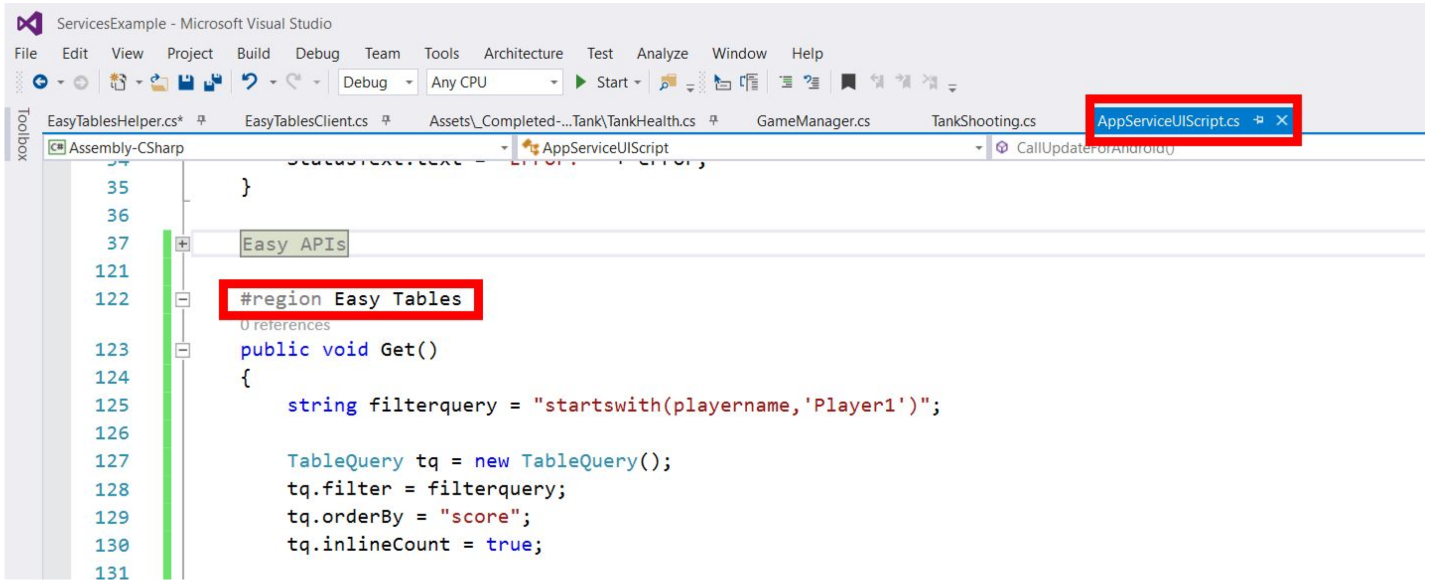
Highscore (2 records)

ID	CREAT...	UPDAT...	VERSI...	DELETE...	PLAYER...	SCORE
5b616c0f...	2017-12-...	2017-12-...	AAAAAA...	false	dimitris	53
639f984b...	2017-12-...	2017-12-...	AAAAAA...	false	Player1	1

Now go to the cloud on **EasyTables** in the **Highscore** table. You should find a new winner's player record.

Below we will consider an example of how you can get data about a particular player, or more precisely how you can easily make a method Select.

Saving game progress from Unity to the Cloud.



```
    }

    Easy APIs

    #region Easy Tables
    0 references
    public void Get()
    {
        string filterquery = "startswith(playername,'Player1')";

        TableQuery tq = new TableQuery();
        tq.filter = filterquery;
        tq.orderBy = "score";
        tq.inlineCount = true;
```

Open the **AppServiceScript** script, then find the Easy Tables region, this is a list of methods for working with Easy Tables Api.

Insert - the method adds a new record to the table
Select (Query) - A method for retrieving records from a table using a query.
SelectByID - a method by which you can get a record of its Id
Update - You can modify the data in the table.
DeleteByID - Method of deleting a field in a database by its Id

https://github.com/rio900/unityazureservices

Below is a description of the available methods of working with **EasyTables**, which can be useful in your game.
Below is an example of a method for getting all the records for the first player. The query value can be edited by retrieving specific samples from the database.

```
public void Get()
  {
      string filterquery = "startswith(playername,'Player1')";

      TableQuery tq = new TableQuery();
      tq.filter = filterquery;
      tq.orderBy = "score";
      tq.inlineCount = true;

      EasyTablesClient.Instance.SelectFiltered<Highscore>(tq, x
=>
      {
          if (x.Status == CallBackResult.Success)
          {
              foreach (var item in x.Result.results)
                  if (Globals.DebugFlag)
Debug.Log(string.Format("ID is {0},score is {1},name is {2}",
                      item.id, item.score, item.playername));
              StatusText.text = string.Format("Brought {0} rows
out of {1}",
                  x.Result.results.Count(), x.Result.count);
          }
          else
              ShowError(x.Exception.Message);
      });
      StatusText.text = "Loading...";
  }
```