Azura Cosmos DB

Chad Armstrong

# Deliverable Summary

Included in this deliverable is the Unity Asset Package, and a copy of an Azure Function Visual Studio Solution. The Unity Package containing the scripts, test scene(with a prefab included), and the plugins needed. For the plugins there is a .xml file with the libraries source location and more references for documentation on how to use these libraries going forward. The Azure Function is written for the sample database under my trial account but can be easily switched to another database with some minor changes.

## Setup

To run this project you must sign up for an Azure account and sign in. Once there, on the left panel click the “+”(“Create a resource”) sign at the top of the panel. Then select or search for Azure Cosmos DB. Fill in the following information(if you don’t have a resource group make a new one) and go ahead and create the database.   
 Once you create it go to your home screen(Left panel->Home) and click on your recently created database. Go to the Keys page(Settings->Keys), you’ll need the URI and PRIMARY KEY to hook up Unity to this database.

## Unity

The deliverable includes a packaged up scene for Unity, to use it you have to import it from an existing scene(Assets ->Import Package -> Custom Package). The package mainly consist of a simple scene made in unity with two buttons and several “Empty” Gameobjects that have scripts attached to them. Before you run the scene be sure to find the “DatabaseConnection” Gameobject in the scene and fill in the public variables in the inspector(Primary key and EndpointURI from the Keys page during setup).

When you hit play the scene initializes any Database connection with the connection keys that you passed in. Upon making a new user (with the “New User” button) the project will make a database with a collection of 50,000 float entities. This may take some time to build, since it builds the database before initializing the users that interact with it(14 seconds average).  
 Once the user is created, it will start to write out random floats at random indices to the database once every 2-60 seconds. This can be toggles on and off through the bool checkbox on the User gameobject(User GameObject inspector-> User Script Object -> bool IsOn).

## Azure Change Feed

The Azure Change Feed allows the database to be triggered on input to a database. The included Azure Function Visual Studio Solution utilizes this by writing an Azure Function that uses this Change Feed trigger to write out to a queue to can be read from the Azure Portal(where the database is).

To start this process, you will have to add a Function App to your Azure Portal. Do this by clicking the “Create a resource” button on the top of the left panel. From search for “Function App” and click it to create a new Function App. Fill out the form to finish creating the Function app.  
 Once you have a Function App made open the Visual Studio Solution included in the

Deliverable, publish the project to your connected Azure Account through Visual Studio. Do this by navigating to: Build->Publish {ProjectName} and Select Azure Function App, Select Existing and Run from package file. If you are unable to connect to an Existing account, try it again with Create new and it should get your data to fill out the settings. Hit OK/Create and then hit “Publish” once it finishes loading.

Once it is published, make sure the “ConnectionStringSetting” variable is filled out in the function preview.

# Notes

* I recommend using the Azura Cosmos Emulator, prevents develops from doing harmful operations on the main server and also they can control everything about it.  
  <https://docs.microsoft.com/en-us/azure/cosmos-db/local-emulator>
* Unity has threads happening in a thread pool behind the scenes, this makes it tricky using multithreaded operations/api calls while using Unity.
* If you want 50,000 element database involved when the first user is created, having an asynchronous call will not cut it. You’d have to create the database on the main thread, which for 50,000 elements takes 10+ seconds. After this the user can safely interact with the database and read/write as much as they need.
* Drops to 1 FPS if several users write/update at one time (5+ users). So scalabilty will be tough, unless this was more of a stress test and there won’t be an upwards of 125,000 updates to the database a second.
* One recommendation I have would be to look into Unity’s job system, it’s how the “handle” threading.
* Azura Cosmos Change Feed is not available on the emulator, and not available with a free Azura subscription.
* Change Feed Processor API SDK provides examples using obsolete data types interfaces and classes, documentation of their suggested classes use the same obsolete software products. Ultimately there documentation for using Change Feed seems to be a work-in-progress and pushes the developer towards their Azure Functions.