

Phil Bennett  
February 9, 2018

## Serverless Scaling with Azure Functions and Cosmos DB

### Abstract

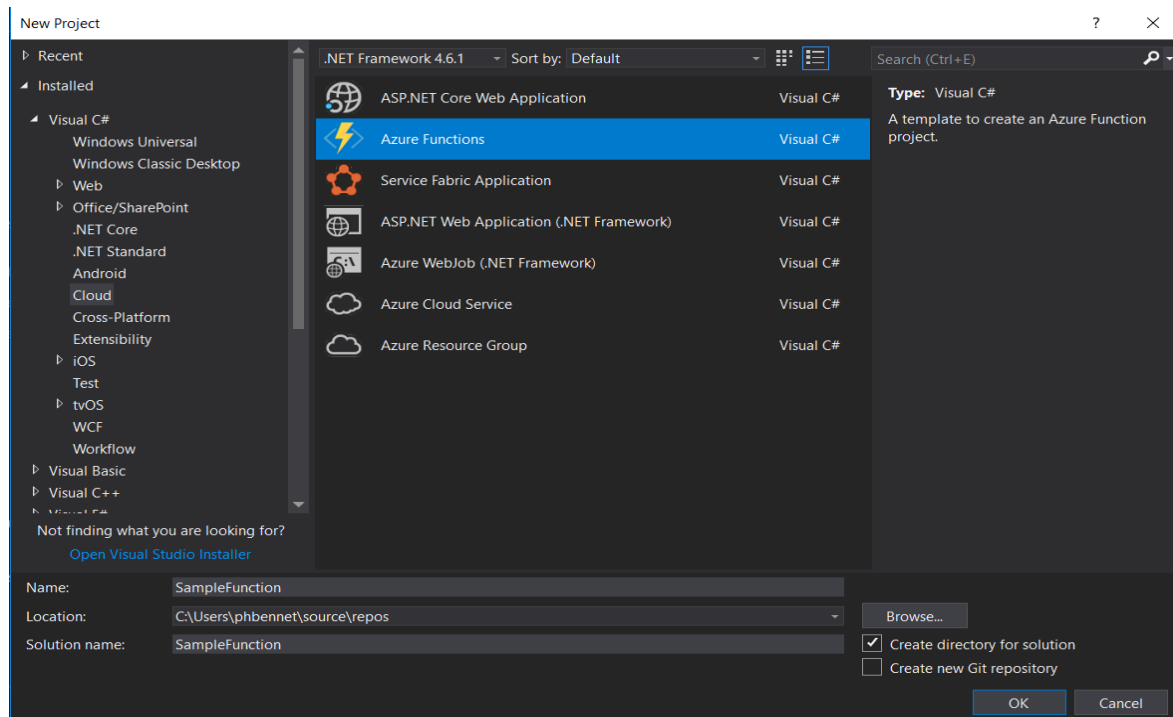
**\*NOTICE: The guide assumes the user has a basic understanding of Microsoft Azure and Visual Studio\***

This guide will walk you through creating an Azure function that scales up throughput on a Cosmos DB collection when a throttling (429 error) alert is fired. To achieve this functionality, we will (1) write an azure function to scale throughput on a collection, (2) publish the function to Azure, and (3) connect the function to a CosmosDB account alert using an http webhook. Some things to consider before implementing this solution: The solution only scales up, but can be used to scale down if you supply a negative value for the CosmosDB\_RUIncrement settings attribute. It is suggested to implement a limit on scaling to keep your bill manageable. Alerts are set at the account level meaning that this function will be triggered any time a resource under the account, for which the alert was set, is throttled.

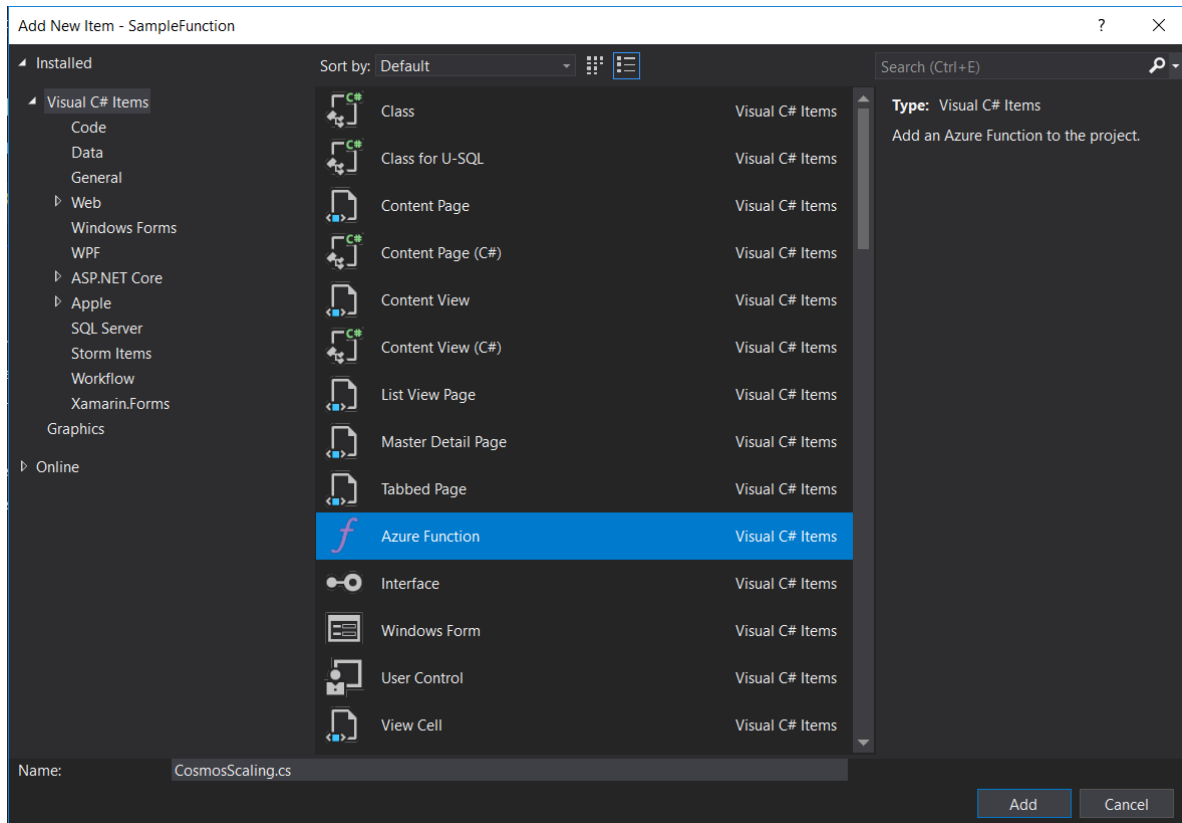
### Create the Function

First, write an azure function that increments the current throughput provisioned for a collection to avoid task cancellation if an operation cannot be resolved due to “Request Rate Too Large” (throttling) errors.

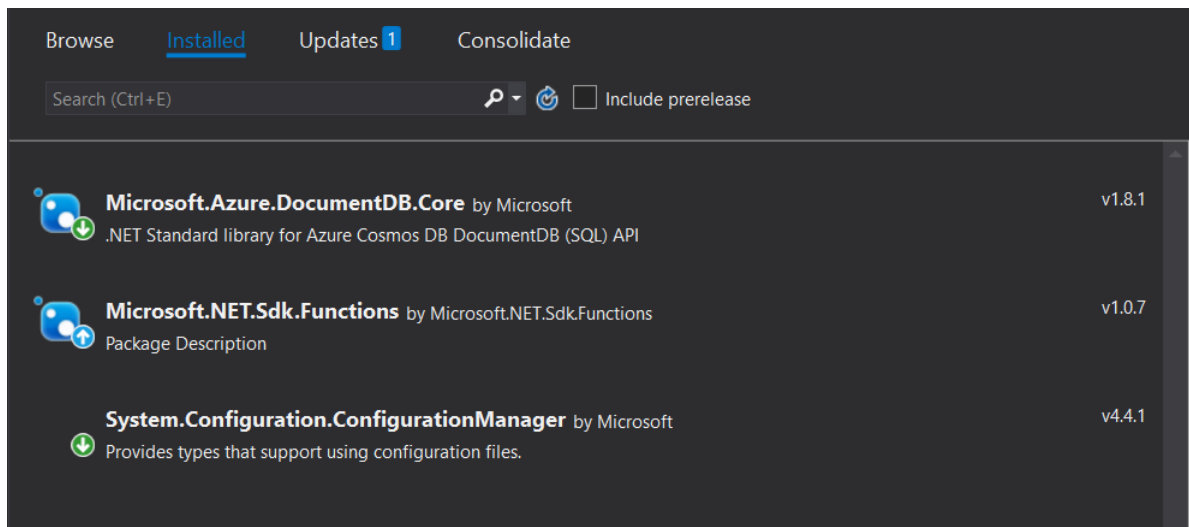
- 1) Open Visual Studio.
- 2) File > New > Project > Visual C# > Cloud > Azure Functions: Give your function a name and click ok.



- 3) Right Click on your project > Add > New Item > Azure Function: Give your function class a name and click ok.



- 4) Right click on your solution file > Manage NuGet Packages for Solution... > Install the packages Microsoft.Azure.Documents.Core, and System.Configuration.ConfigurationManager. Your solution will already have Microsoft.NET.Sdk.Functions installed.



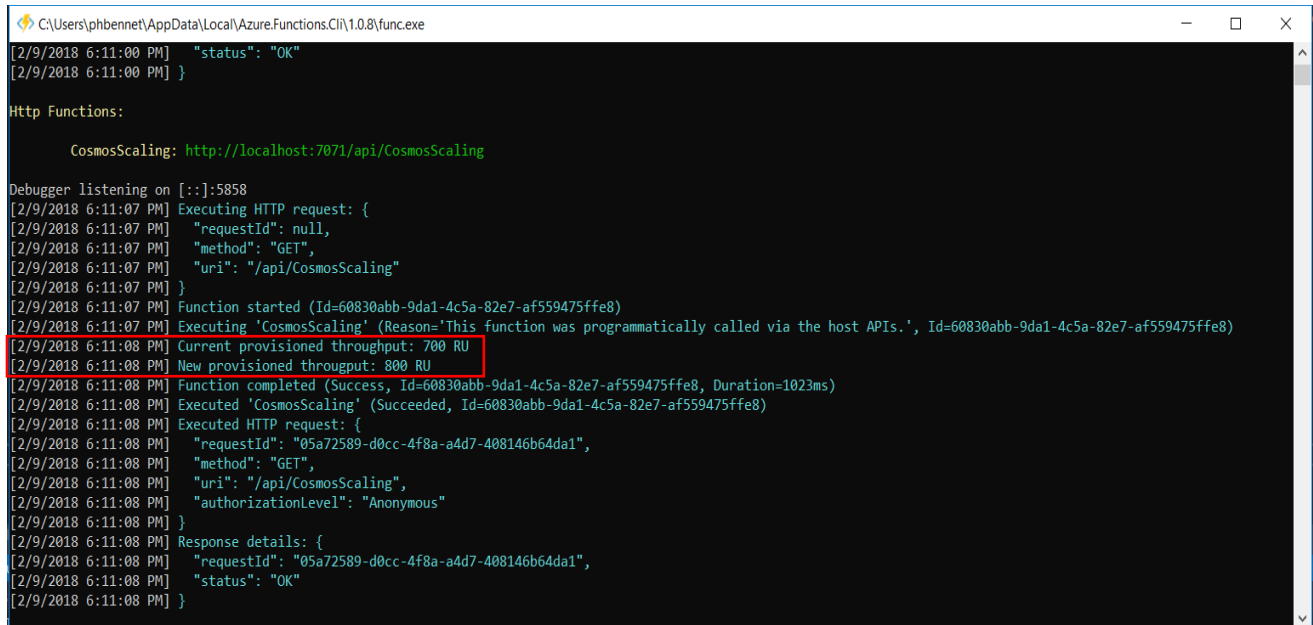
- 5) Add the values highlighted in yellow to “local.settings.json”. These values will be used to test the function locally before deploying it to Azure:

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "",
    "AzureWebJobsDashboard": "",
    "CosmosDB_Uri": "https://<uri>.documents.azure.com:443/",
    "CosmosDB_Key": "<primary key>=",
    "CosmosDB_DatabaseId": "<database name>",
    "CosmosDB_CollectionId": "<collection name>",
    "CosmosDB_RUIncrement": 100
  }
}
```

- 6) Add the following method to the body of the C# sharp class you created for your function (code attached in CosmosScaling.cs):

```
19 public static async Task<HttpStatusCode> Run([HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)]HttpRequestMessage req, TraceWriter log)
20 {
21     try
22     {
23         //1) initialize the document client
24         using (DocumentClient client = new DocumentClient(new Uri(ConfigurationManager.AppSettings["CosmosDB_Uri"]), ConfigurationManager.AppSettings["CosmosDB_Key"]))
25         {
26             //2) get the database self link
27             string selfLink = client.CreateDocumentCollectionQuery(
28                 UriFactory.CreateDatabaseUri(ConfigurationManager.AppSettings["CosmosDB_DatabaseId"]))
29                 .Where(c => c.Id == ConfigurationManager.AppSettings["CosmosDB_CollectionId"])
30                 .AsEnumerable()
31                 .FirstOrDefault()
32                 .SelfLink;
33
34             //3) get the current offer for the collection
35             Offer offer = client.CreateOfferQuery()
36                 .Where(r => r.ResourceLink == selfLink)
37                 .AsEnumerable()
38                 .SingleOrDefault();
39
40             //4) get the current throughput from the offer
41             int throughputCurrent = (int)offer.GetProperty<JsonObject>("content").GetValue("offerThroughput");
42             log.Info(string.Format("Current provisioned throughput: {0} RU", throughputCurrent.ToString()));
43
44             //5) get the RU increment from AppSettings and parse to an int
45             if (int.TryParse(ConfigurationManager.AppSettings["CosmosDB_RUIncrement"], out int RUIncrement))
46             {
47                 //5.a) create the new offer with the throughput increment added to the current throughput
48                 int newThroughput = throughputCurrent + RUIncrement;
49                 offer = new OfferV2(offer, newThroughput);
50
51                 //5.b) persist the changes
52                 await client.ReplaceOfferAsync(offer);
53                 log.Info(string.Format("New provisioned throughput: {0} RU", newThroughput.ToString()));
54                 return req.CreateResponse(HttpStatusCode.OK, "The collection's throughput was changed...");
55             }
56             else
57             {
58                 //5.c) if the throughputIncrement cannot be parsed return throughput not changed
59                 return req.CreateResponse(HttpStatusCode.OK, "PARSE ERROR: The collection's throughput was not changed...");
60             }
61         }
62     }
63     catch (Exception e)
64     {
65         log.Info(e.Message);
66         return req.CreateResponse(HttpStatusCode.OK, "ERROR: The collection's throughput was not changed...");
67     }
68 }
```

- 7) Test the app locally by clicking the run button. Send a GET or POST request to the function at: `http://localhost:7071/api/<function name>`. If you have set up everything correctly you will see the following response in your in console. Notice the logger information messages that specify current and new provisioned throughput:



```
C:\Users\phbennet\AppData\Local\Azure.Functions.Cli\1.0.8\func.exe
[2/9/2018 6:11:00 PM] "status": "OK"
[2/9/2018 6:11:00 PM] }

Http Functions:

CosmosScaling: http://localhost:7071/api/CosmosScaling

Debugger listening on [::]:5858
[2/9/2018 6:11:07 PM] Executing HTTP request: {
[2/9/2018 6:11:07 PM]   "requestId": null,
[2/9/2018 6:11:07 PM]   "method": "GET",
[2/9/2018 6:11:07 PM]   "uri": "/api/CosmosScaling"
[2/9/2018 6:11:07 PM] }
[2/9/2018 6:11:07 PM] Function started (Id=60830abb-9da1-4c5a-82e7-af559475ffe8)
[2/9/2018 6:11:07 PM] Executing 'CosmosScaling' (Reason='This function was programmatically called via the host APIs.', Id=60830abb-9da1-4c5a-82e7-af559475ffe8)
[2/9/2018 6:11:08 PM] Current provisioned throughput: 700 RU
[2/9/2018 6:11:08 PM] New provisioned throughput: 800 RU
[2/9/2018 6:11:08 PM] Function completed (Success, Id=60830abb-9da1-4c5a-82e7-af559475ffe8, Duration=1023ms)
[2/9/2018 6:11:08 PM] Executed 'CosmosScaling' (Succeeded, Id=60830abb-9da1-4c5a-82e7-af559475ffe8)
[2/9/2018 6:11:08 PM] Executed HTTP request: {
[2/9/2018 6:11:08 PM]   "requestId": "05a72589-d0cc-4f8a-a4d7-408146b64da1",
[2/9/2018 6:11:08 PM]   "method": "GET",
[2/9/2018 6:11:08 PM]   "uri": "/api/CosmosScaling",
[2/9/2018 6:11:08 PM]   "authorizationLevel": "Anonymous"
[2/9/2018 6:11:08 PM] }
[2/9/2018 6:11:08 PM] Response details: {
[2/9/2018 6:11:08 PM]   "requestId": "05a72589-d0cc-4f8a-a4d7-408146b64da1",
[2/9/2018 6:11:08 PM]   "status": "OK"
[2/9/2018 6:11:08 PM] }
```

### Create a Functions App on Azure and Publish the Function

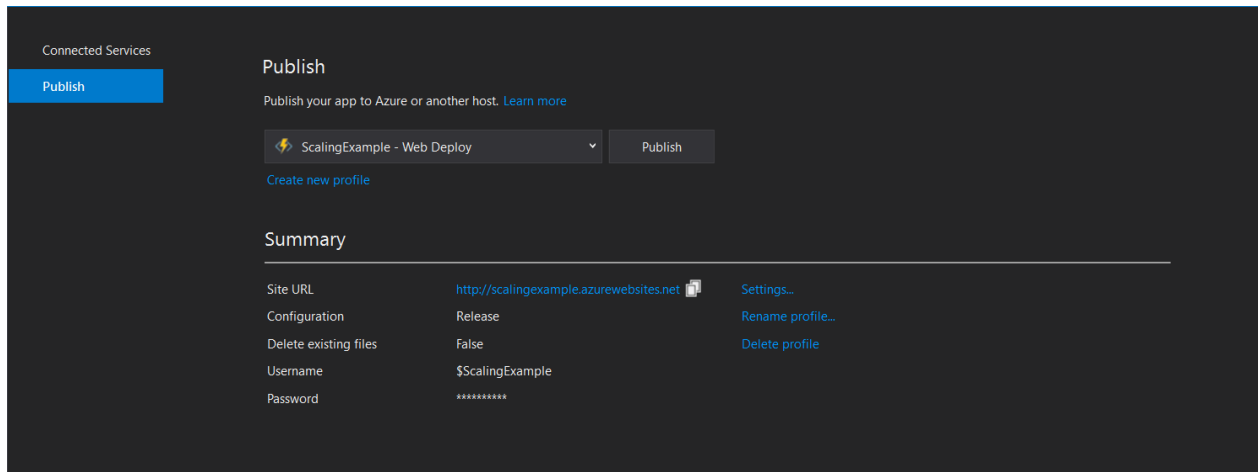
In this step we will deploy an Azure Functions App through the portal, and publish the function we wrote in the previous step to the new Function App.

- 1) Navigate to the portal and provision a Serverless Function App with default settings.
- 2) Click on “Function app settings” on your Function App’s homepage, then click on “Manage application settings”. Add the values in the table below to Application settings:

Adding values to Application settings allows the function’s manager to edit the values if needed.

Key	Value
CosmosDB_Uri	https://<uri>.documents.azure.com:443/
CosmosDB_Key	<primary key>==
CosmosDB_DatabaseId	<database name>
CosmosDB_CollectionId	<collection name>
CosmosDB_RUIncrement	<RU Increment as an integer>

- 3) Publish your app in Visual Studio: right click on the project file > Publish... > Select Existing > Publish > Select the Function App you provisioned in the previous step and click ok.



- 4) Test your function on Azure by navigating to the function, in the portal blade, and clicking run. You should see the following output if the function succeeds. If the function fails, check the key-value pairs you defined in the previous step and make sure they are correct.



## Create a Throttling Alert for your Cosmos DB Account

In this step we will create an alert and bind it to our function.

- 1) Get your function's URL by clicking on `</> Get Function URL` in the upper right-hand side of the online code editor. Your URL should be of the form:

`https://<function app name>.azurewebsites.net/api/<function name>?code=<your code>`

- 2) Define an alert on the desired Cosmos DB account with the following settings. Remember to add your function's URL to the final form field entitled "Webhook":

Home > All resources > phb-dev-test - Alert rules > Edit Rule

### Edit Rule

scaleUpOn429

Save Discard Enable Disable Delete

Description

Scales up the collection "sample collection" by 100 RU when a 429 error is encountered

Metric ⓘ

Throttled Requests

Condition

greater than or equal to

\* Threshold ⓘ

1

count

Period ⓘ

Over the last 24 hours

Email owners, contributors, and readers

☐

Additional administrator email(s)

Add email addresses separated by semicolons

Webhook ⓘ

https://<function app name>.azurewebsites.net/api/<function name>?code=<your code>

[Learn more about configuring webhooks](#)

That's it! Now each time a resource in your account is throttled the resource specified by `CosmosDB_DatabaseId/CosmosDB_CollectionId` will scale up.