.NET Web Development and Tools Blog

Your official information source from the .NET Web Development and Tools group at Microsoft.

Visual Studio

Visual Studio Tools for Azure Functions

★★★★★

December 1, 2016 by Andrew B Hall - MSFT // 134 Comments

Share 504 624 187

Update 3-10-2017: This preview copy of Azure Functions Tools does not work with the newly released Azure SDK 3.0. If you want to continue using these tools on Visual Studio 2015, you will need to remain on the 2.9.6 SDK. Additionally, there are no Azure Function Tools currently available for Visual Studio 2017. We are actively working on the 2017 tools, and will provide an update in the next few weeks regarding our plans and strategy.

Today we are pleased to announce a preview of tools for building Azure Functions for Visual Studio 2015. Azure Functions provide event-based serverless computing that make it easy to develop and scale your application, paying only for the resources your code consumes during execution. This preview offers the ability to create a function project in Visual Studio, add functions using any supported language, run them locally, and publish them to Azure. Additionally, C# functions support both local and remote debugging.

In this post, I'll walk you through using the tools by creating a C# function, covering some important concepts along the way. Then, once we've seen the tools in action I'll cover some known limitations we currently have.

Also, please take a minute and let us know who you are so we can follow up and see how the tools are working.

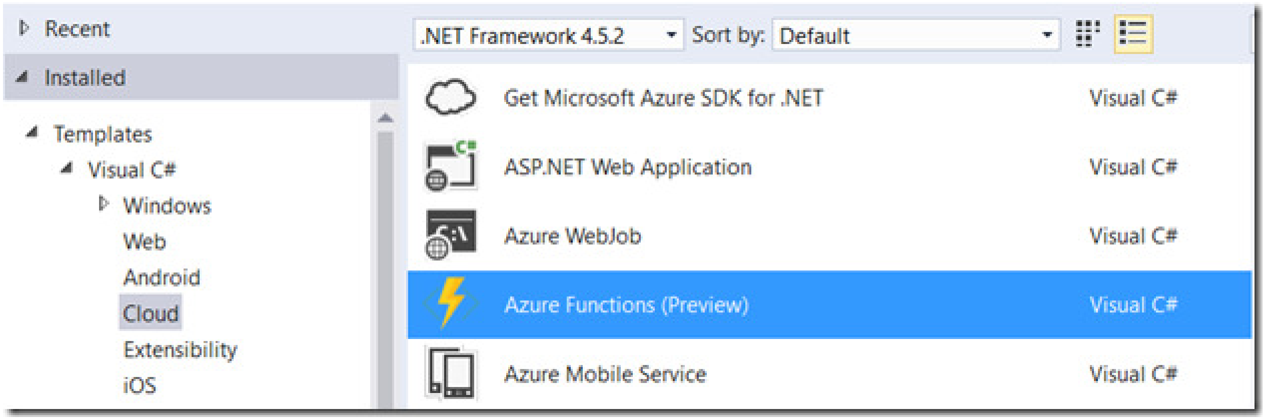
Getting Started

Before we dive in, there are a few things to note:

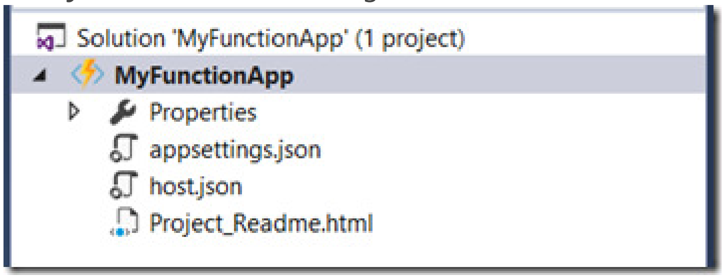
- These tools are offered as a preview release and will have some rough spots and limitations
- They currently only work with Visual Studio 2015 Update 3 with "Microsoft Web Developer Tools" installed
- You must have Azure 2.9.6 .NET SDK installed
- Download and install Visual Studio Tools for Azure Functions

For our sample function, we'll create a C# function that is triggered when a message is published into a storage Queue, reverses it, and stores both the original and reversed strings in Table storage.

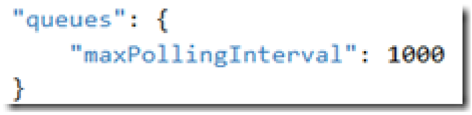
- To create a function, go to:
- File -> New Project
- Then select the "Cloud" node under the "Visual C#" section and choose the "Azure Functions (Preview) project type



- This will give us an empty function project. There are a few things to note about the structure of the project:
  - **appsettings.json** is where we'll store configuration information such as connection strings
    It is recommended that you exclude this file from source control so you don't check in your developer secrets.
  - **host.json** enables us to configure the behavior of the Azure Functions host

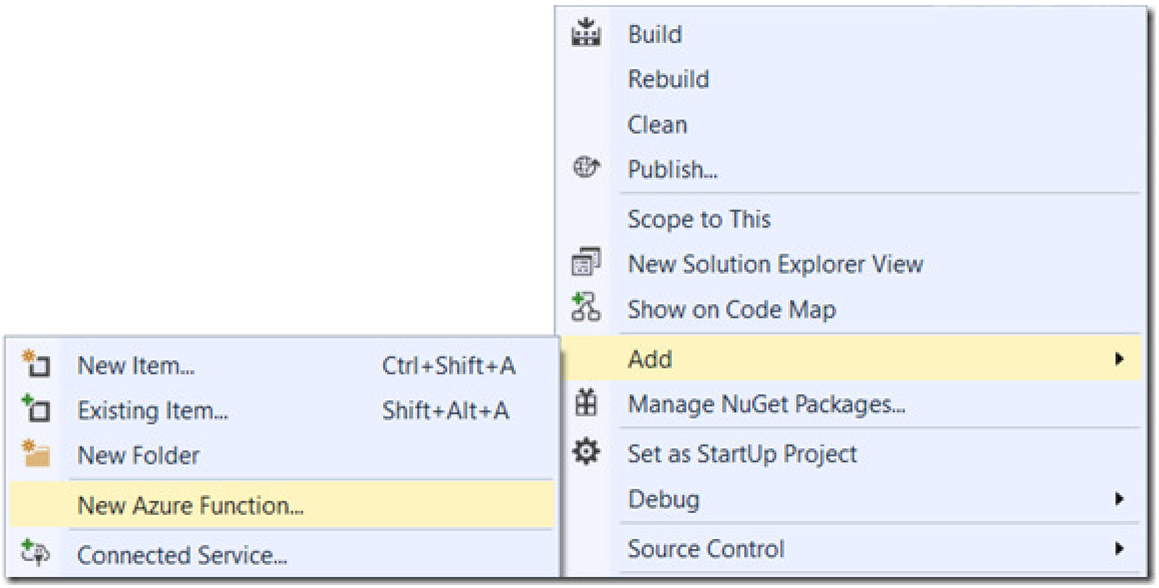


- For the purposes of this blog post, we'll add an entry that speeds up the queue polling interval from the default of once a minute to once a second by setting the "maxPollingInterval" in the host.json (value is in ms)

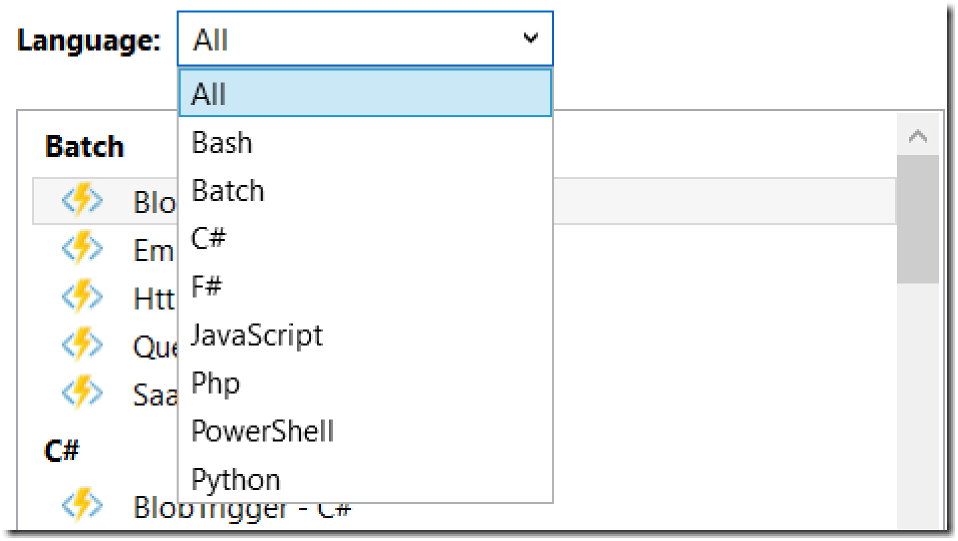


```
"queues": {
    "maxPollingInterval": 1000
}
```

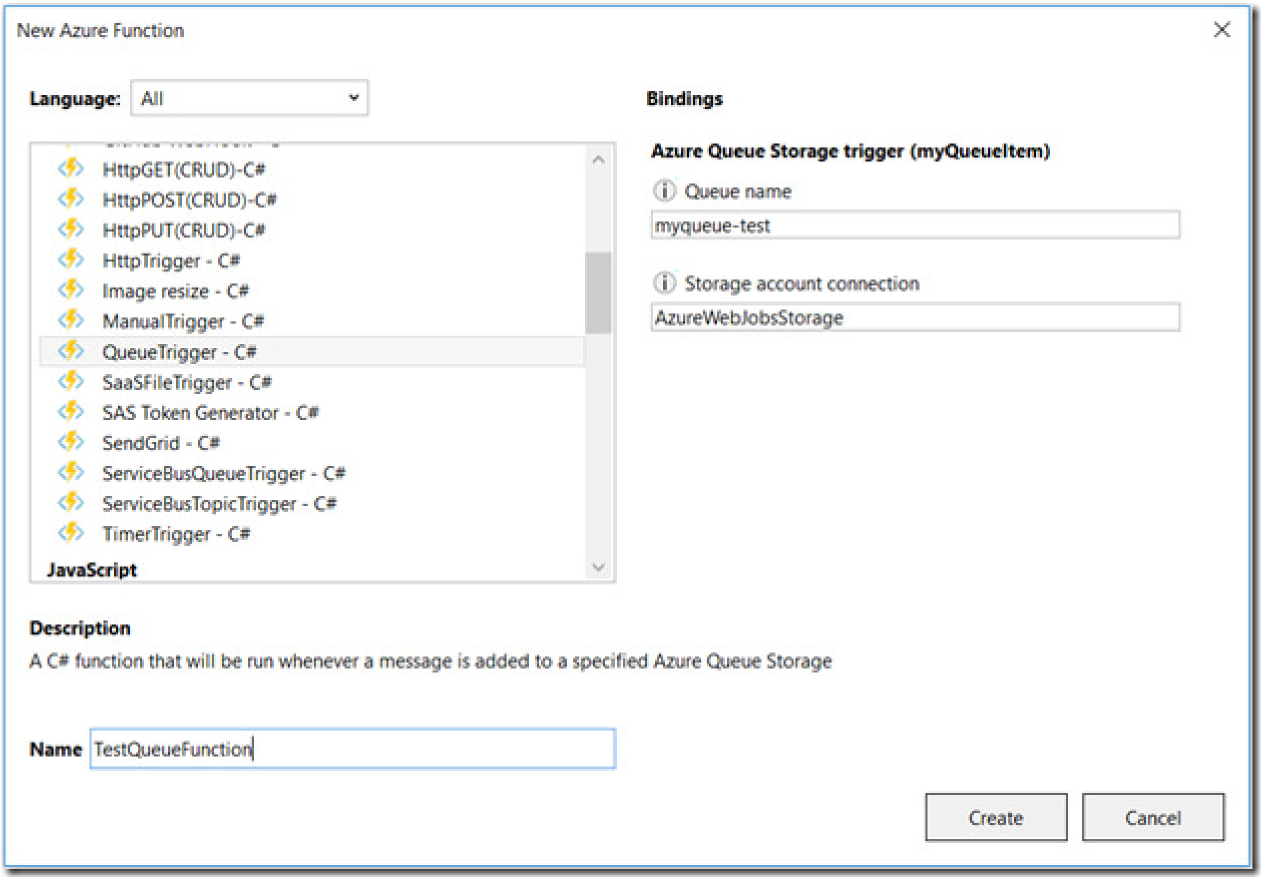
- Next, we'll add a function to the project, by right clicking on the project in Solution Explorer, choose "Add" and then "New Azure Function"



- This will bring up the New Azure Function dialog which enables us to create a function using any language supported by Azure Functions



- For the purposes of this post we'll create a "QueueTrigger – C#" function, fill in the "Queue name" field, "Storage account connection" (this is the name of the key for the setting we'll store in "appsettings.json"), and the "Name" of our function. *Note: All function types except HTTP triggers* *require a storage connection or you will receive an error at run time*



- HttpGET(CRUD)-C#
- HttpPOST(CRUD)-C#
- HttpPUT(CRUD)-C#
- HttpTrigger - C#
- Image resize - C#
- ManualTrigger - C#
- QueueTrigger - C#
- SaaSFileTrigger - C#
- SAS Token Generator - C#
- SendGrid - C#
- ServiceBusQueueTrigger - C#
- ServiceBusTopicTrigger - C#
- TimerTrigger - C#

JavaScript

Bindings

Azure Queue Storage trigger (myQueueItem)

Queue name: myqueue-test

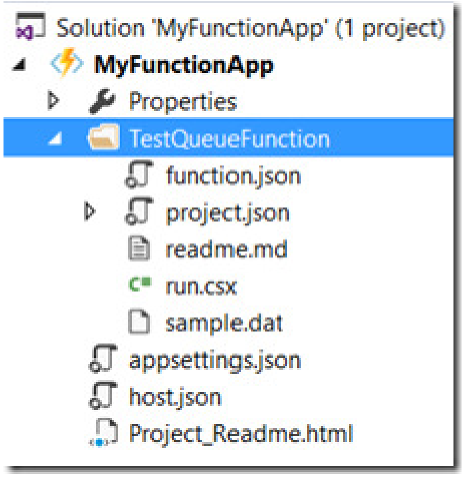
Storage account connection: AzureWebJobsStorage

Description

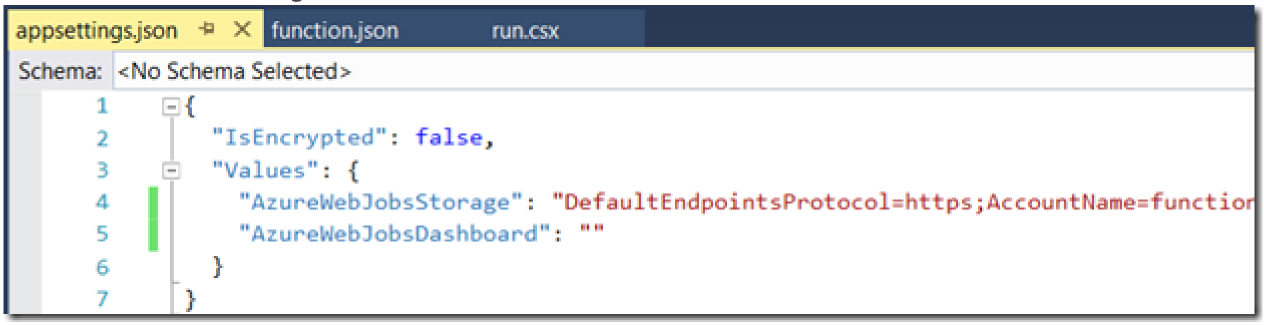
A C# function that will be run whenever a message is added to a specified Azure Queue Storage

Name: TestQueueFunction

- This will create a new folder in the project with the name of our function with the following key files:
  - **function.json:** contains the configuration data for the function (including the information we specified as part of creating the new function)
  - **project.json (C#):** is where we'll specify any NuGet dependencies our function may have. *Note: Azure functions automatically import some namespaces and assemblies (e.g. Json.NET).*
  - **run.csx:** this contains the body of the function that will be executed when triggered

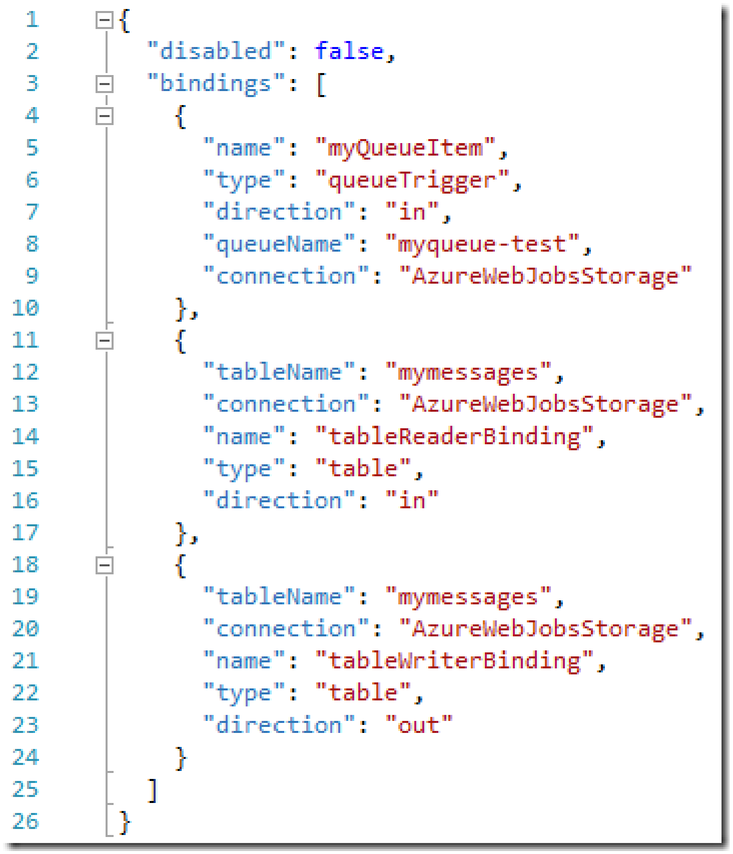


- The last thing we need to do in order to hook up function to our storage Queue is provide the connecting string in the appsettings.json file (in this case by setting the value of "AzureWebJobsStorage")



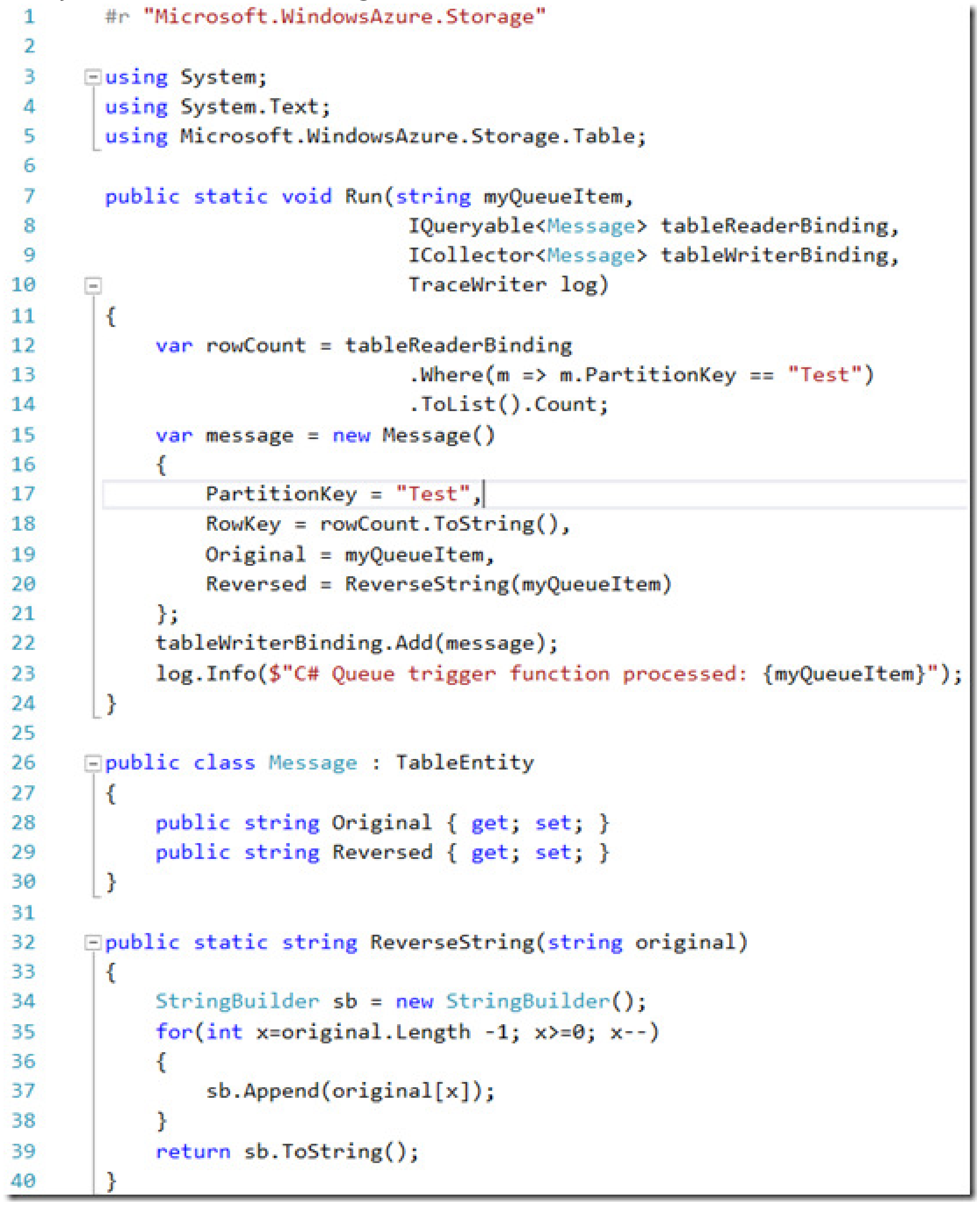
```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "DefaultEndpointsProtocol=https;AccountName=functior
    "AzureWebJobsDashboard": ""
  }
}
```

- Next we'll edit the "function.json" file to add two bindings, one that gives us the ability to read from the table we'll be pushing to, and another that gives us the ability to write entries to the table



```
{
  "disabled": false,
  "bindings": [
    {
      "name": "myQueueItem",
      "type": "queueTrigger",
      "direction": "in",
      "queueName": "myqueue-test",
      "connection": "AzureWebJobsStorage"
    },
    {
      "tableName": "mymessages",
      "connection": "AzureWebJobsStorage",
      "name": "tableReaderBinding",
      "type": "table",
      "direction": "in"
    },
    {
      "tableName": "mymessages",
      "connection": "AzureWebJobsStorage",
      "name": "tableWriterBinding",
      "type": "table",
      "direction": "out"
    }
  ]
}
```

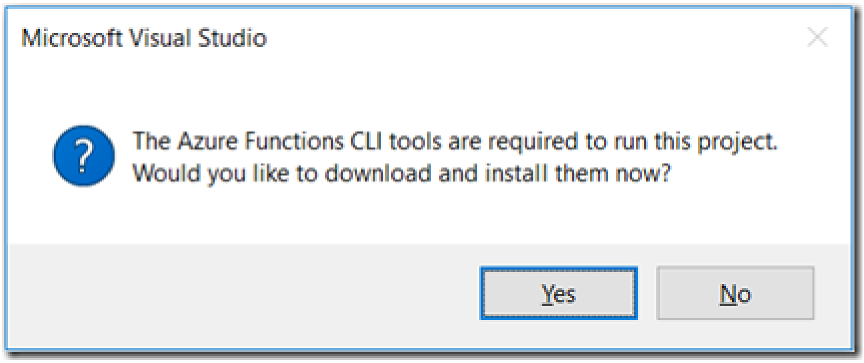
- Finally, we'll write our function logic in the run.csx file



```
#r "Microsoft.WindowsAzure.Storage"

using System;
using System.Text;
using Microsoft.WindowsAzure.Storage.Table;

public static void Run(string myQueueItem,
                        IQueryable<Message> tableReaderBinding,
                        ICollector<Message> tableWriterBinding,
                        TraceWriter log)
{
    var rowCount = tableReaderBinding
                        .Where(m => m.PartitionKey == "Test")
                        .ToList().Count;
    var message = new Message()
    {
        PartitionKey = "Test",
        RowKey = rowCount.ToString(),
        Original = myQueueItem,
        Reversed = ReverseString(myQueueItem)
    };
    tableWriterBinding.Add(message);
    log.Info($"C# Queue trigger function processed: {myQueueItem}");
}

public class Message : TableEntity
{
    public string Original { get; set; }
    public string Reversed { get; set; }
}

public static string ReverseString(string original)
{
    StringBuilder sb = new StringBuilder();
    for(int x=original.Length -1; x>=0; x--)
    {
        sb.Append(original[x]);
    }
    return sb.ToString();
}
```

- Running the function locally works like any other project in Visual Studio, Ctrl + F5 starts it without debugging, and F5 (or the Start/Play button on the toolbar) launches it with debugging. *Note: Debugging currently only works for C# functions. Let's hit F5 to debug the function.*
- The first time we run the function, we'll be prompted to install the Azure Functions CLI (command line) tools. Click "Yes" and wait for them to install, our function app is now running locally. We'll see a command prompt with some messages from the Azure Functions CLI pop up, if there were any compilation problems, this is where the messages would appear since functions are dynamically

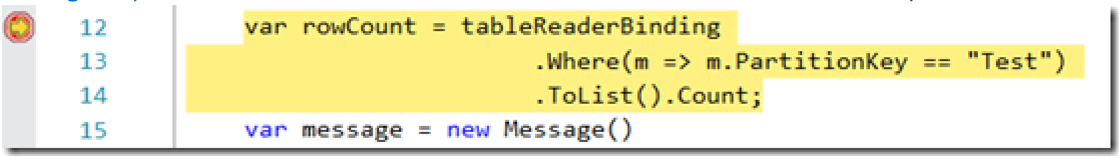
compiled by the CLI tools at runtime.



Microsoft Visual Studio

The Azure Functions CLI tools are required to run this project.
Would you like to download and install them now?

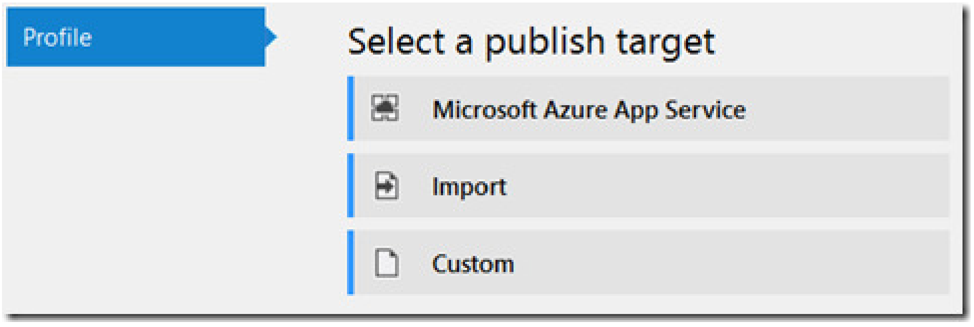
- We now need to manually trigger our function by pushing a message into the queue with Azure Storage Explorer. This will cause the function to execute and hit our breakpoint in Visual Studio.



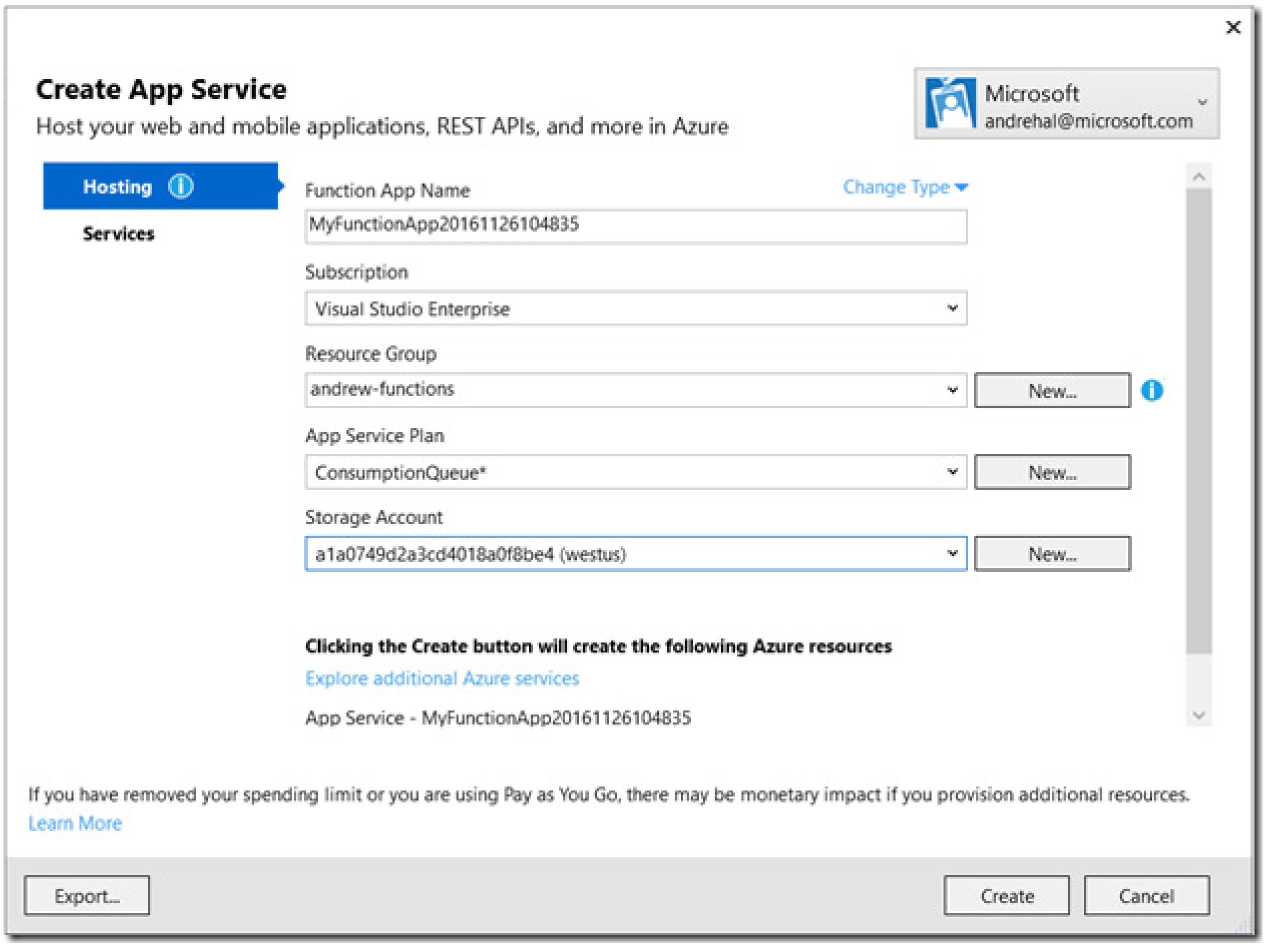
```
var rowCount = tableReaderBinding
                .Where(m => m.PartitionKey == "Test")
                .ToList().Count;
var message = new Message()
```

Publishing to Azure

- Now that we've tested the function locally, we're ready to publish our function to Azure. To do this right click on the project and choose "Publish...", then choose "Microsoft Azure App Service" as the publish target



- Next, you can either pick an existing app, or create a new one. We'll create a new one by clicking the "New..." button on the right side of the dialog
- This will pop up the provisioning dialog that lets us choose or setup the Azure environment (we can customize the names or choose existing assets). These are:
  - **Function App Name:** the name of the function app, this must be unique
  - **Subscription:** the Azure subscription to use
  - **Resource Group:** what resource group the to add the Function App to
  - **App Service Plan:** What app service plan you want to run the function on. For complete information read about hosting plans, but it's important to note that if you choose an existing App Service plan you will need to set the plan to "always on" or your functions won't always trigger (Visual Studio automatically sets this if you create the plan from Visual Studio)
- Now we're ready to provision (create) all of the assets in Azure. *Note: that the "Validate Connection" button does not work in this preview for Azure Functions*



Function App Name: MyFunctionApp20161126104835

Subscription: Visual Studio Enterprise

Resource Group: andrew-functions

App Service Plan: ConsumptionQueue*

Storage Account: a1a0749d2a3cd4018a0f8be4 (westus)

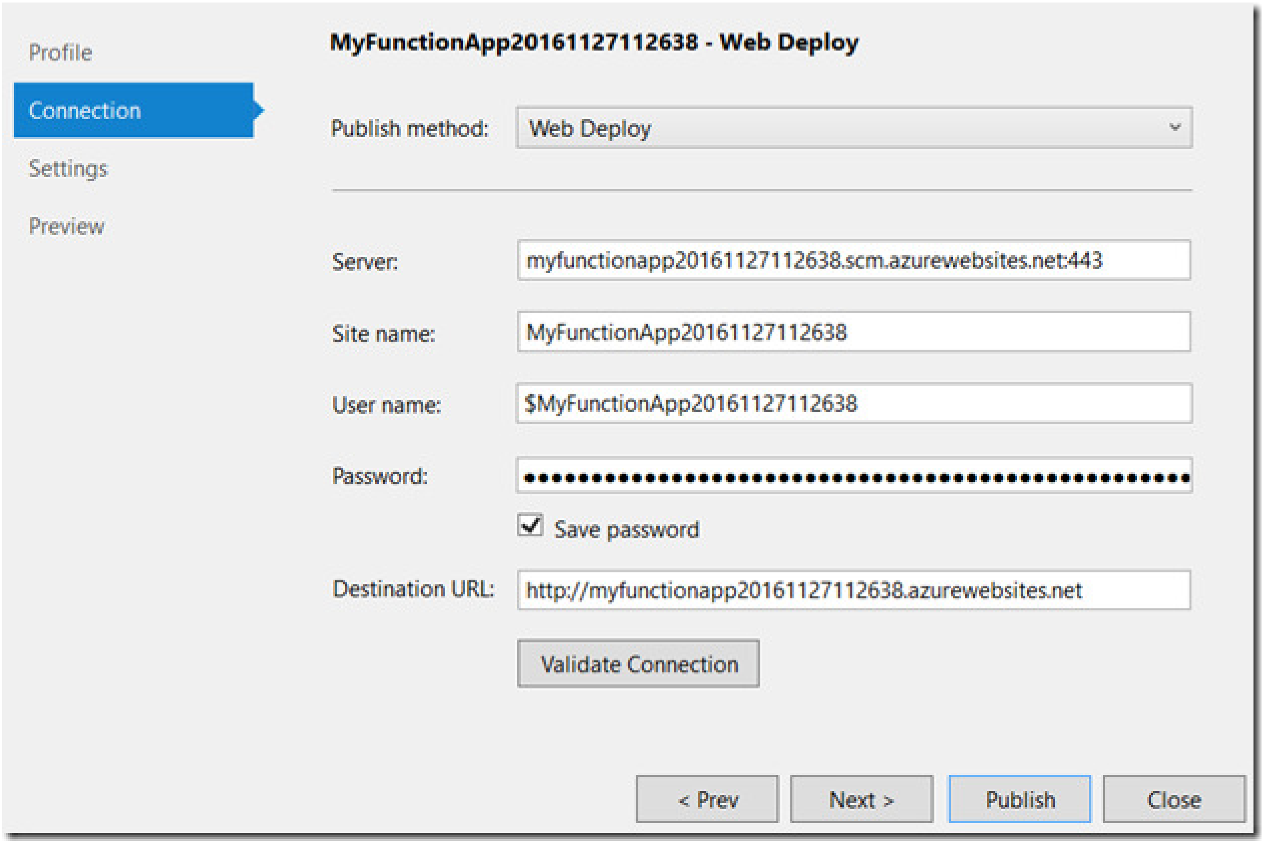
Clicking the Create button will create the following Azure resources

Explore additional Azure services

App Service - MyFunctionApp20161126104835

If you have removed your spending limit or you are using Pay as You Go, there may be monetary impact if you provision additional resources.
Learn More

- Once provisioning is complete, click "Publish" to publish the Function to Azure. We now have a publish profile which means all future publishes will skip the provisioning steps



MyFunctionApp20161127112638 - Web Deploy

Publish method: Web Deploy

Server: myfunctionapp20161127112638.scm.azurewebsites.net:443

Site name: MyFunctionApp20161127112638

User name: $MyFunctionApp20161127112638

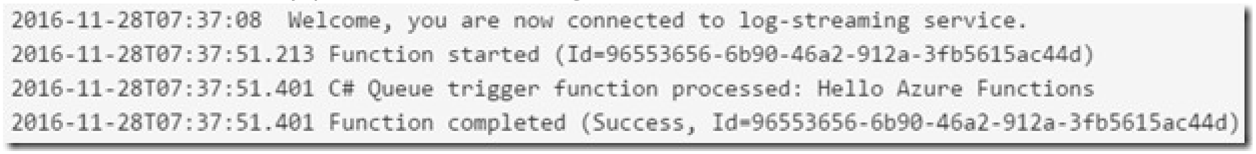
Password: ••

☑ Save password

Destination URL: http://myfunctionapp20161127112638.azurewebsites.net

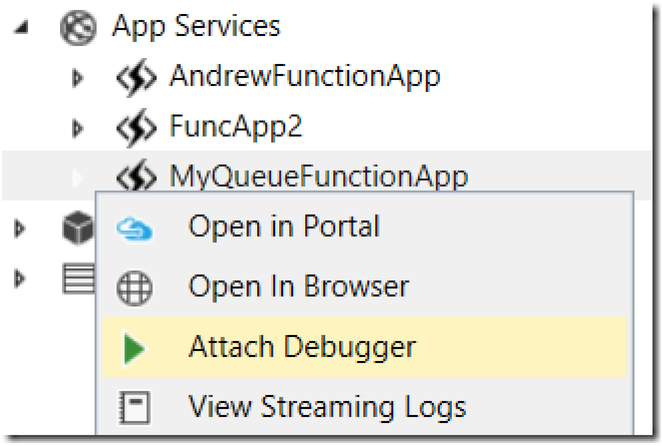
Note: If you publish to a Consumption plan, there is currently a bug where new triggers that you define (other than HTTP) will not be registered in Azure, which can cause your functions not to trigger correctly. To work around this, open your Function App in the Azure portal and click the "Refresh" button on the lower left to fix the trigger registration. This bug with publish will be fixed on the Azure side soon.

- To verify our function is working correctly in Azure, we'll click the "Logs" button on the function's page, and then push a message into the Queue using Storage Explorer again. We should see a message that the function successfully processed the message



```
2016-11-28T07:37:08  Welcome, you are now connected to log-streaming service.
2016-11-28T07:37:51.213 Function started (Id=96553656-6b90-46a2-912a-3fb5615ac44d)
2016-11-28T07:37:51.401 C# Queue trigger function processed: Hello Azure Functions
2016-11-28T07:37:51.401 Function completed (Success, Id=96553656-6b90-46a2-912a-3fb5615ac44d)
```

- The last thing to note, is that it is possible to remote debug a C# function running in Azure from Visual Studio. To do this:
  - Open Cloud Explorer
  - Browse to the Function App
  - Right click and choose "Attach Debugger"



Known Limitations

As previously mentioned, this is the first preview of these tools, and we have several known limitations with them. They are as follow:

- **IntelliSense:** IntelliSense support is limited, and available only for C#, and JavaScript by default. F#, Python, and PowerShell support is available if you have installed those optional components. It is also important to note that C# and F# IntelliSense is limited at this point to classes and methods defined in the same .csx/.fsx file and a few system namespaces.
- **Cannot add new files using "Add New Item":** Adding new files to your function (e.g. .csx or .json files) is not available through "Add New Item". The workaround is to add them using file explorer, the Add New File extension, or another tool such as Visual Studio Code.
- **Function bindings generate incorrectly when creating a C# Image Resize function:** The settings for the binding "Azure Storage Blob out (imageSmall)" are overridden by the settings for the binding "Azure Storage Blob out (imageMedium)" in the generated function.json. The workaround is to go to the generated function.json and manually edit the "imageSmall" binding.
- **Local deployment and web deploy packages are not supported:** Currently, only Web Deploy to App Service is supported. If you try to use Local Deploy or a Web Deploy Package, you'll see the error "GatherAllFilesToPublish does not exist in the project".

- **The Publish Preview shows all files in the project's folder even if they are not part of the project:** Publish preview does not function correctly, and will cause all files in the project folder to be picked up and and published. Avoid using the Preview view.
- **The publish option "Remove additional files at destination" does not work correctly:** The workaround is to remove these files manually by going to the Azure Functions Portal, Function App Settings -> App Service Editor

Conclusion

Please download and try out this preview of Visual Studio Tools for Azure Functions and let us know who you are so we can follow up and see how they are working. Additionally, please report any issues you encounter on our GitHub repo (include "Visual Studio" in the issue title) and provide any comments or questions you have below, or via Twitter.

Search MSDN with Bing

Search this blog [✓ Search all blogs]

ASP.NET Resources

www.ASP.Net

ASP.NET Forums

Web Developer Checklist

ASP.NET User Voice

.NET Meetups

Related Blogs

Scott Guthrie's Blog

Scott Hanselman's Blog

Sayed Hashimi's Blog

Jon Galloway's blog

Mads Kristensen's blog

Entity Framework Team blog

Brady Gaster's Blog

NuGet Team Blog

Jeff Fritz's Blog

Vittorio Bertocci's Blog