Azure Logic App Cloud Adapters, Functions, and Storage

Receiving Real-time Alerts on Negative Email Feedback using Azure Logic Apps

Stephen W. Thomas  
[www.StephenWThomas.com](http://www.StephenWThomas.com)  
@StephenWThomas

Prepared for the Global Integration Bootcamp, New York City. 2017.

Distribute freely and openly.

Lab created on Match 25th, 2017.

Contents

[Scenario Overview 3](#_Toc478150215)

[Prerequisites 3](#_Toc478150216)

[Technologies 3](#_Toc478150217)

[General Considerations 4](#_Toc478150218)

[Setting Up Cognitive Services 5](#_Toc478150219)

[Setting Up Azure Storage 6](#_Toc478150220)

[Creating the Function 10](#_Toc478150221)

[Creating the Logic App 12](#_Toc478150222)

# Scenario Overview

You are an Integration Architect at a large fashion design company called Lighting Clothing (It’s Striking). They have a team of resources that monitor the social media feeds and email looking for positive and negative mentions and responding to feedback. Most of the credible feedback arrives via email. This can sometimes point out critical manufacturing flaws that need to be addressed right away. The email sometimes arrives as HTML and needs to be cleaned.

Since people are using email to make a formal complaint, they sometimes attach a write up as a PDF or Word documents. People will also attach pictures of the clothes.

The Vice President has tasked you with the assignment of delivering a notification system that will notify him via text when overly negative feedback is received. You must also save any attachments for later review.

# Prerequisites

The following is required to complete this lab:

* Active Windows Azure subscription
* Microsoft Azure Storage Explorer or other storage explorer to test your solution
  + Download from <http://storageexplorer.com/>
* Outlook 365 Hosted Exchange account
  + One will be provided for boot camp attendees
* Twilio account and API key with a validated cell phone number
  + One will be provided for boot camp attendees

# Technologies

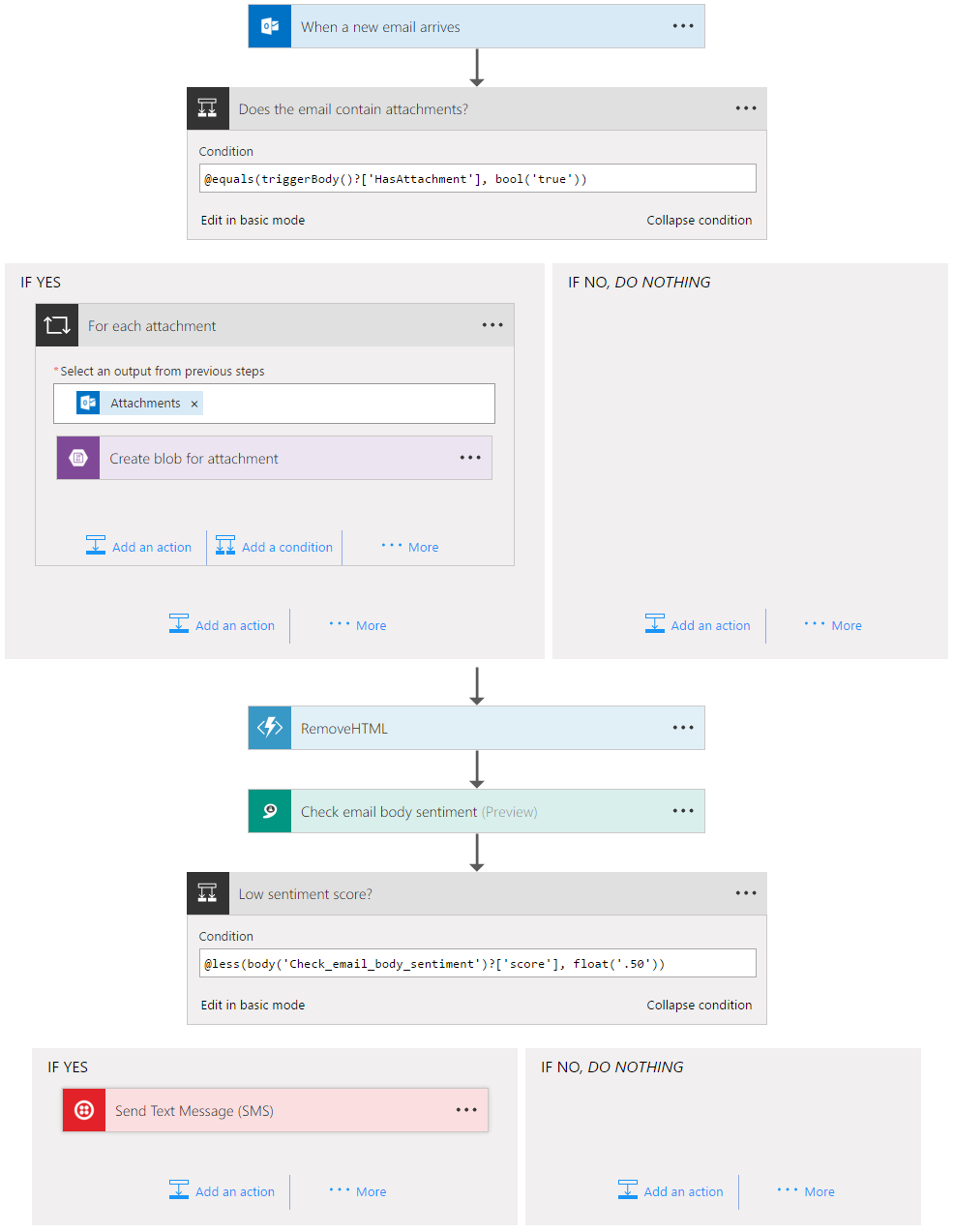
This lab covers the following technologies:

* Windows Azure Logic Apps
  + Office 365 Connector
  + Twilio Connector
  + Cognitive Services Connector
  + Azure Storage Connector
* Windows Azure Functions
  + This is used to clean the HTML from the body of the email
* Windows Azure Storage Accounts
  + This is used to store the attachments for later processing
* Windows Acount Cognitive Services
  + Test Text Analytics here: <https://text-analytics-demo.azurewebsites.net/>

# General Considerations

When setting up the first part of this lab, the Cognitive Service, you will create a Resource Group inside a Region. For simplicity, ensure you use the same Resource Group throughout the lab. Try to keep as many items as possible in the same Region. Not all components are available in all regions. When done, ensure to delete the Resource Group and all included resources as no all items are free of cost.

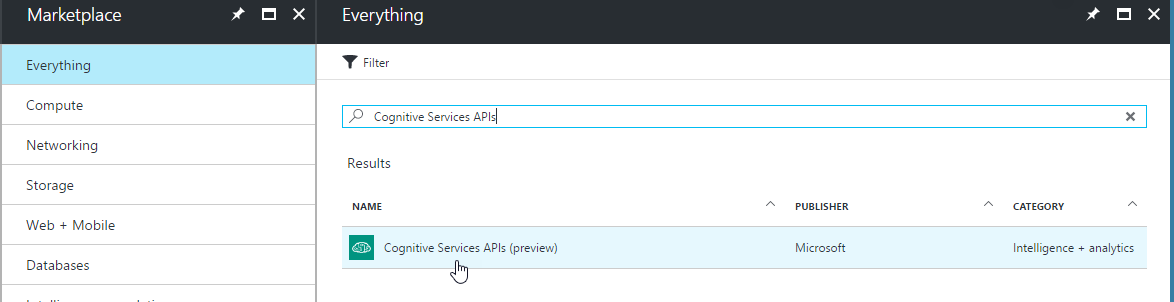
At the end of this lab you will have a Logic App that looks like this:



# Setting Up Cognitive Services

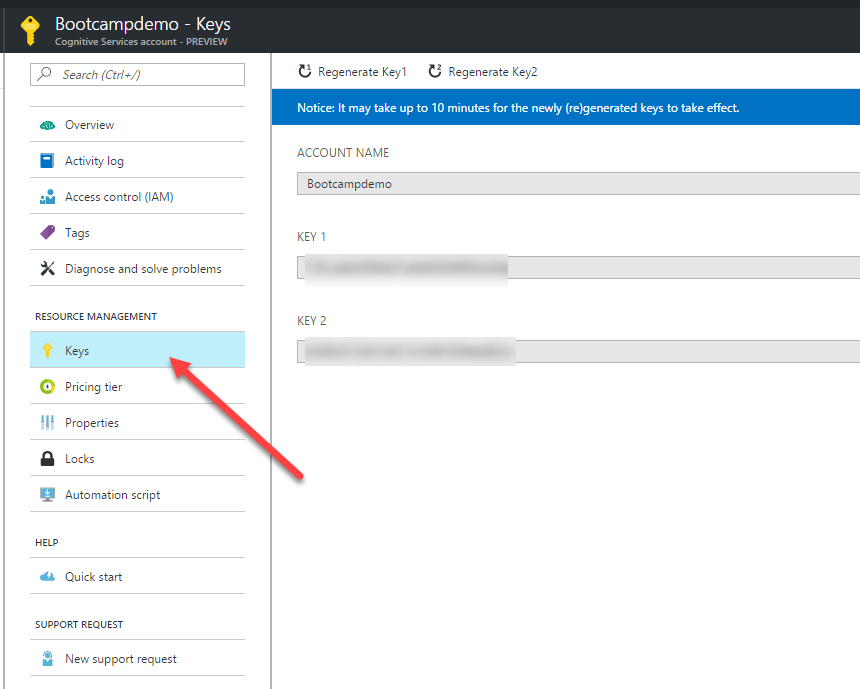
In this section, you will create the Cognitive Service to do the sentiment analysis of the emails.

1. Log into the Windows Azure Portal, click on the green plus on the left to create a new item, and search for **Cognitive Services APIs**. Select Cognitive Services APIs and click **Create** in the bottom right.



1. Configure the service by setting the following values:

|  |  |
| --- | --- |
| 1. Account name: **Bootcampdemo** 2. Subscription: ***Select your subscription*** 3. API type: **Text Analytics API (preview)** 4. Location: **West US** 5. Pricing tier: **F0** (if you can select it) 6. Resource group: **NYC-Bootcamp** 7. API Setting: **Enable** and **Save** 8. Click **Create** |  |

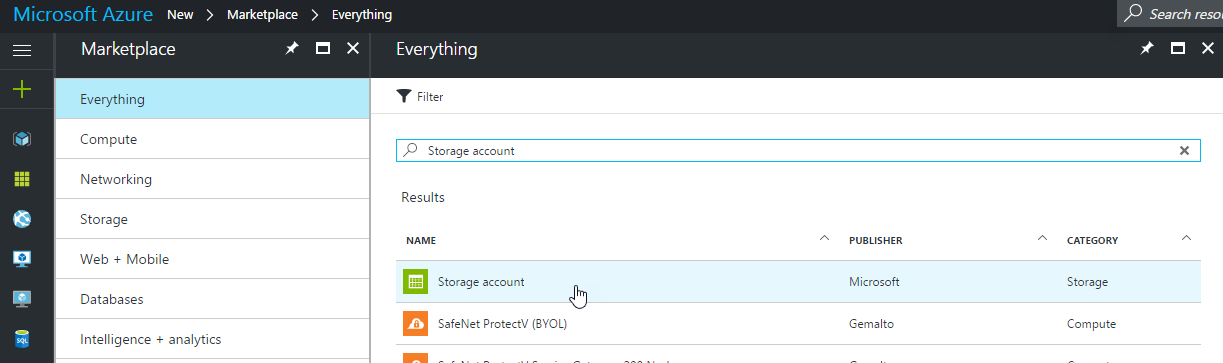
1. Navigate back into the newly created Cognitive Service by going to Cognitive Services inside the Azure Portal. Click on the newly created cognitive Service: **Bootcampdemo**.
2. Click on **Keys**. Copy **Key 1** to notepad for use later inside the Logic App.  
     
   

# Setting Up Azure Storage

Azure Storage will be used to store email attachments. In future work, we will want to review and possibly convert attachments to text for analysis.

Note: When using storage accounts with Function they must be of type General Purpose. All other types are not supported.

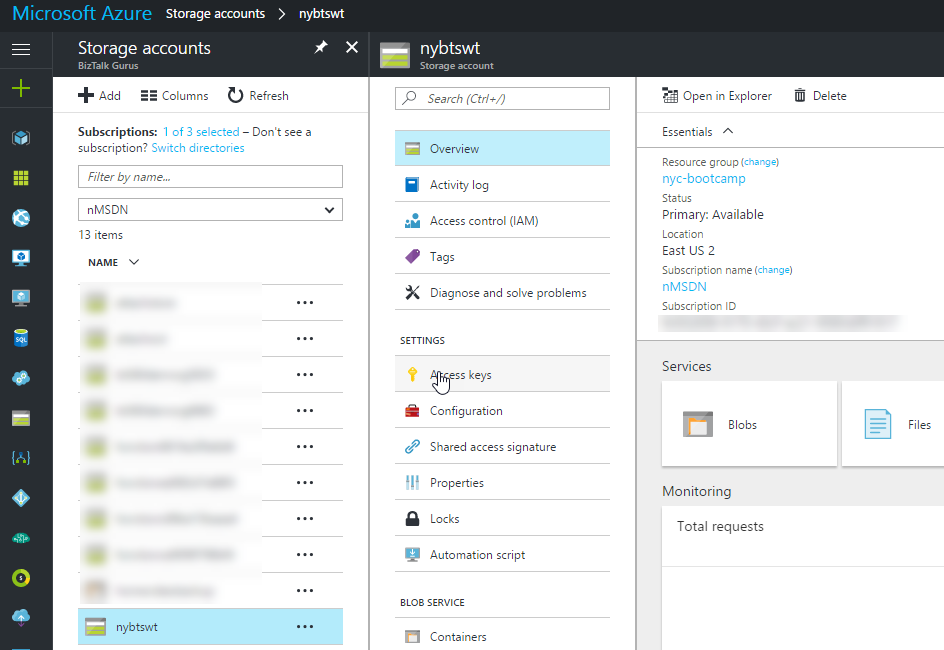
1. Log into the Windows Azure Portal, click on the green plus on the left to create a new item, and search for **Storage account**. Select Storage account and click **Create** in the bottom right.

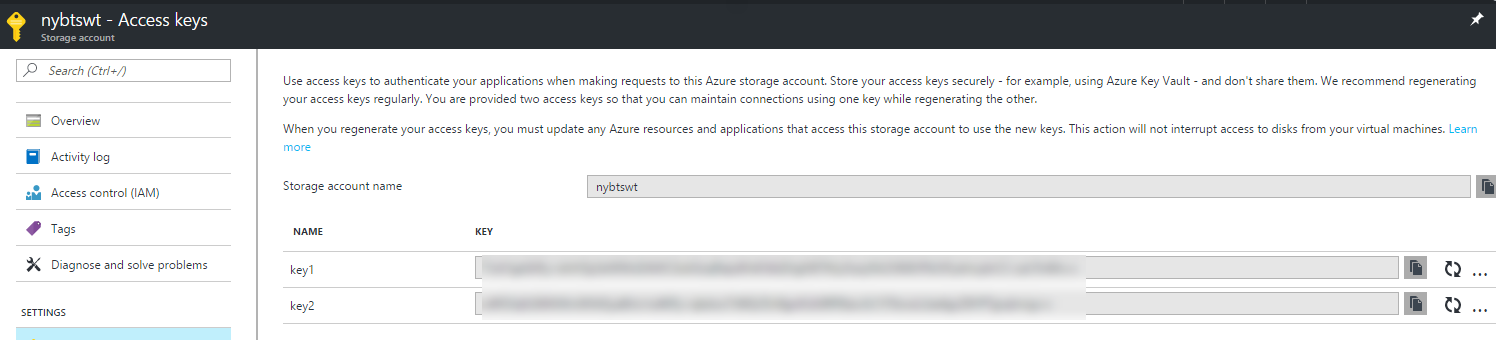


1. Configure the storage account with the following settings:

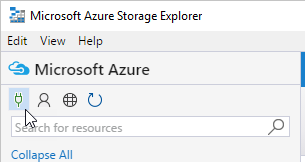
|  |  |
| --- | --- |
| 1. Name: **nycbt<Your Initials>**  (Note this needs to be globally unique) 2. Deployment model: **Resource manager** 3. Account kind: **General purpose** 4. Performance: **Standard** 5. Replication: **Locally-redundant storage (LRS)** 6. Storage service encryption: **Disabled** 7. Subscription: ***Select your subscription*** 8. Resource group: *Select resource group already created* (**NYC-Bootcamp)** 9. Location: **East US 2** 10. Click **Create** |  |

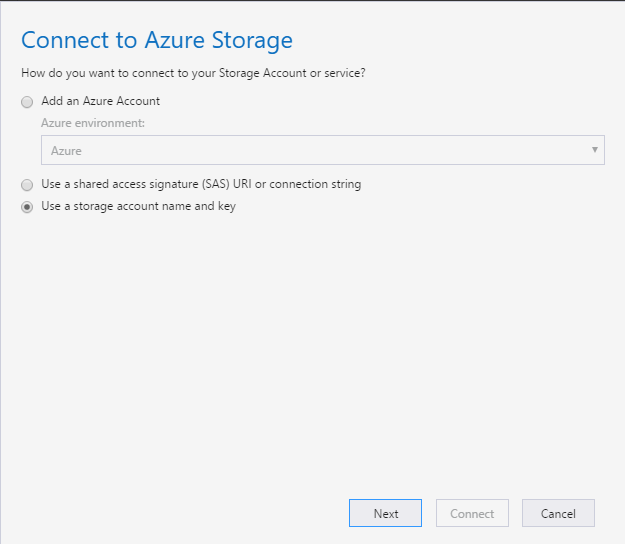
1. While we are at our storage account, lets set it up inside our storage explorer tool (this case using Microsoft Azure Storage Explorer).   
   Navigate back into the newly created Storage account. Click on **Access Keys**.

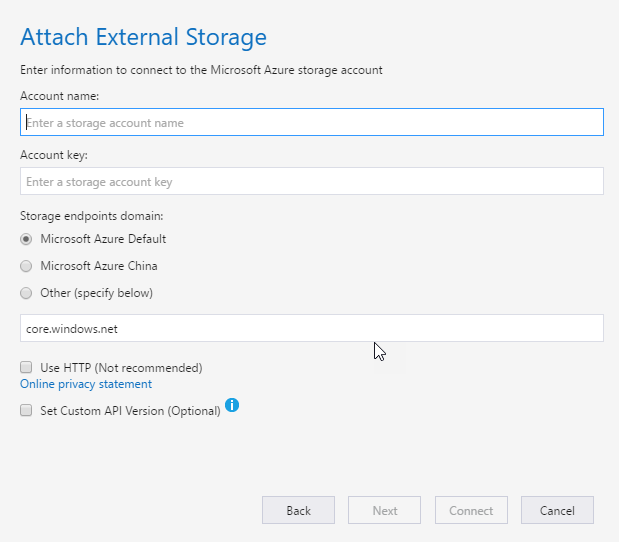




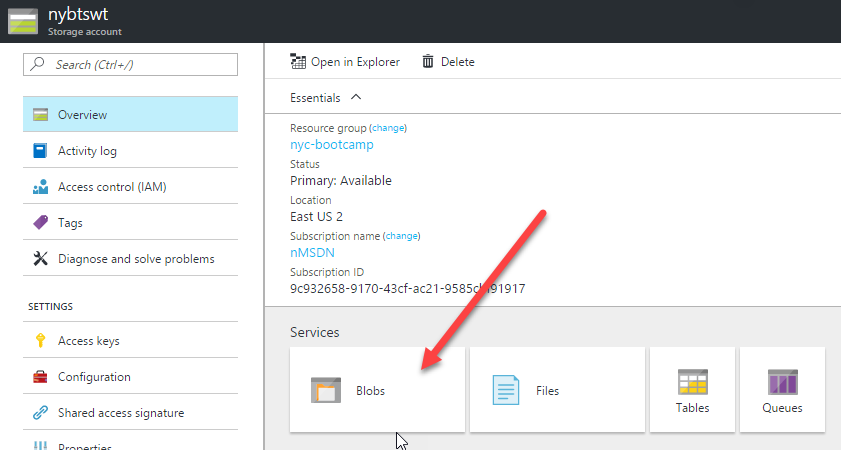
1. Copy and Paste the **Storage account name** and **key1** into Azure Storage Explorer by clicking on the green plug-in. Select **Use a storage account name and key**. Enter your Name and Key. Click **Connect**.



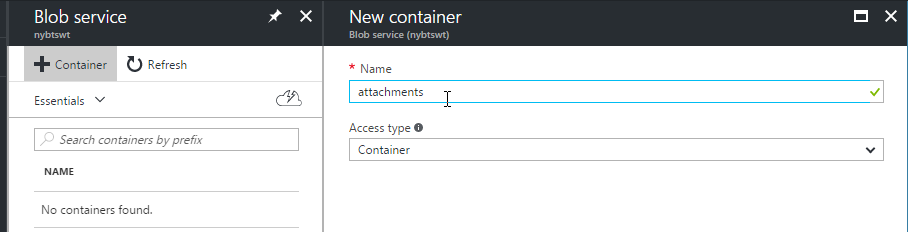




1. Now let us create a Container to store the email attachments. Back in our Azure Portal, click on **Overview** for your storage account. Under Services, click on **Blobs**.



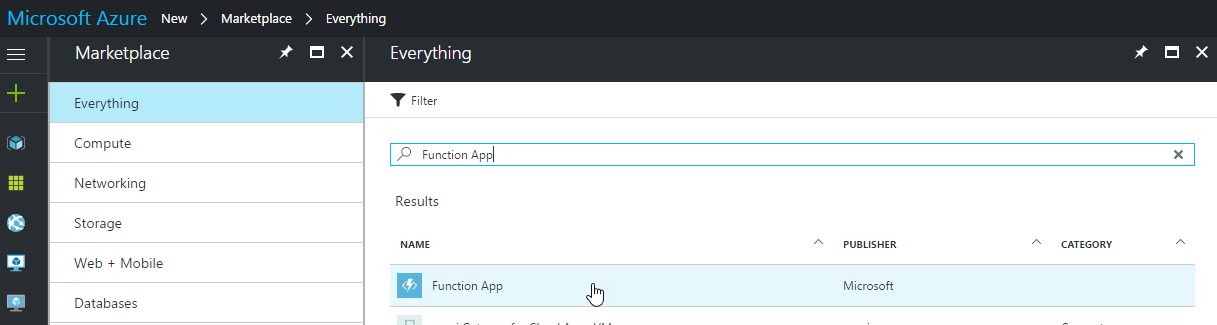
1. Click on the **+ Container**, name the container **attachments** and set Access type to **Container**. Click **Create**.



# Creating the Function

As part of the solution, the email body needs to be cleaned of any HTML code so the sentiment analysis can be as accurate as possible. This cleaning will be done inside an Azure Function.

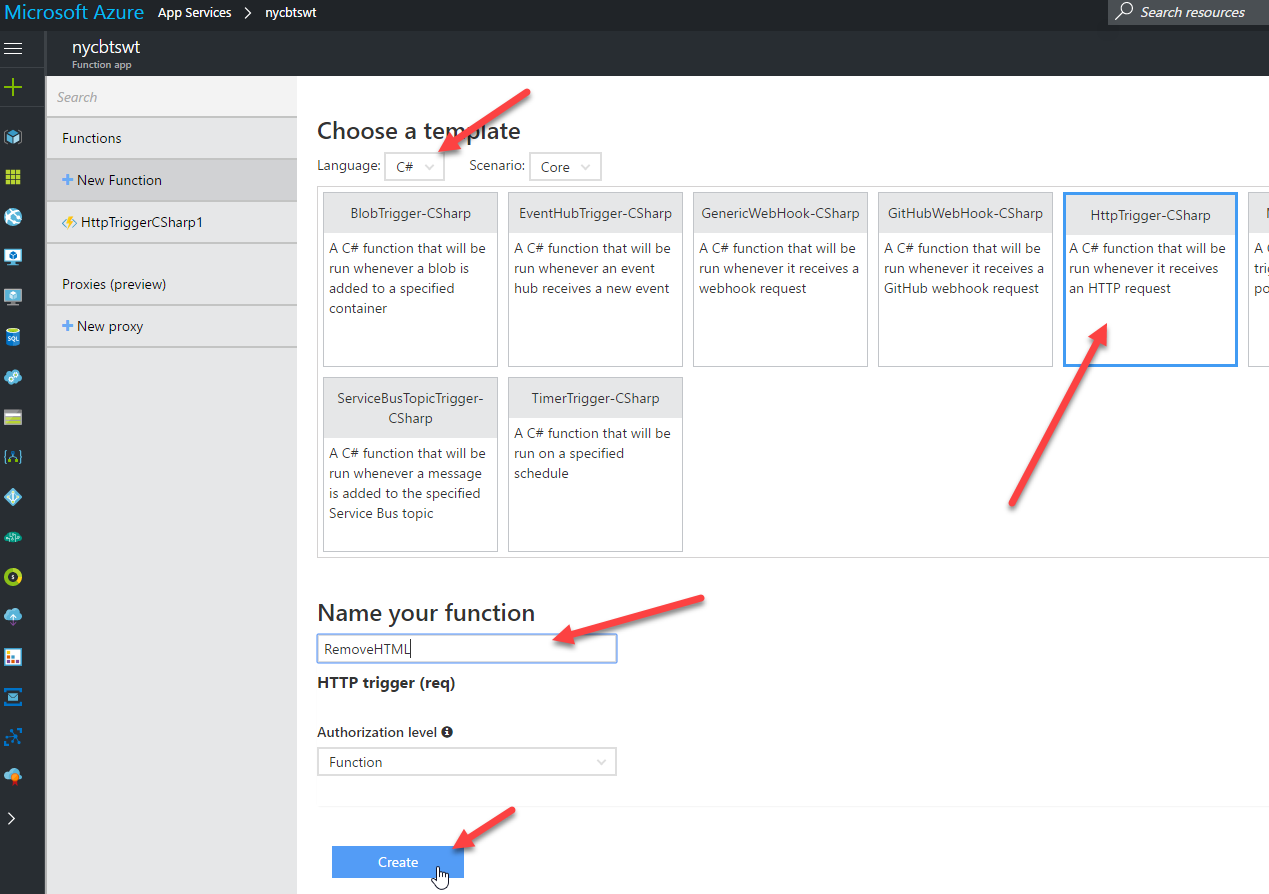
1. Log into the Windows Azure Portal, click on the green plus on the left to create a new item, and search for **Function App**. Select Function App and click **Create** in the bottom right.



1. Configure the function app with the following settings:

|  |  |
| --- | --- |
| 1. App Name: **nycbt<Your Initials>**  (Note this needs to be globally unique) 2. Subscription: *Select your subscription* 3. Resource group: *Select resource group already created* (**NYC-Bootcamp)** 4. Hosting Plan: **Consumption Plan** 5. Location: **East US 2** 6. Storage Account: *Use the default value* 7. Click **Create** |  |

1. Navigate back into the newly created Function App by going to App Services. Click on the newly created function app: **nycbt<Your Initials>**.
2. Do not create the function using the main welcome screen. Oddly, it do not let you set a custom name for the function.   
   Select **+New Function** on the left. Change the language to **C#**. Select **HttpTrigger-CSharp**. Name the function **RemoveHTML**.



1. Replace the existing code with the following (new lines are highlighted):

using System.Net;

using System.Text.RegularExpressions;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter log)

{

log.Info($"Webhook was triggered!");

string emailBodyContent = await req.Content.ReadAsStringAsync();

string updatedBody = Regex.Replace(emailBodyContent, "<.\*?>", string.Empty);

updatedBody = updatedBody.Replace("\\r\\n", " ");

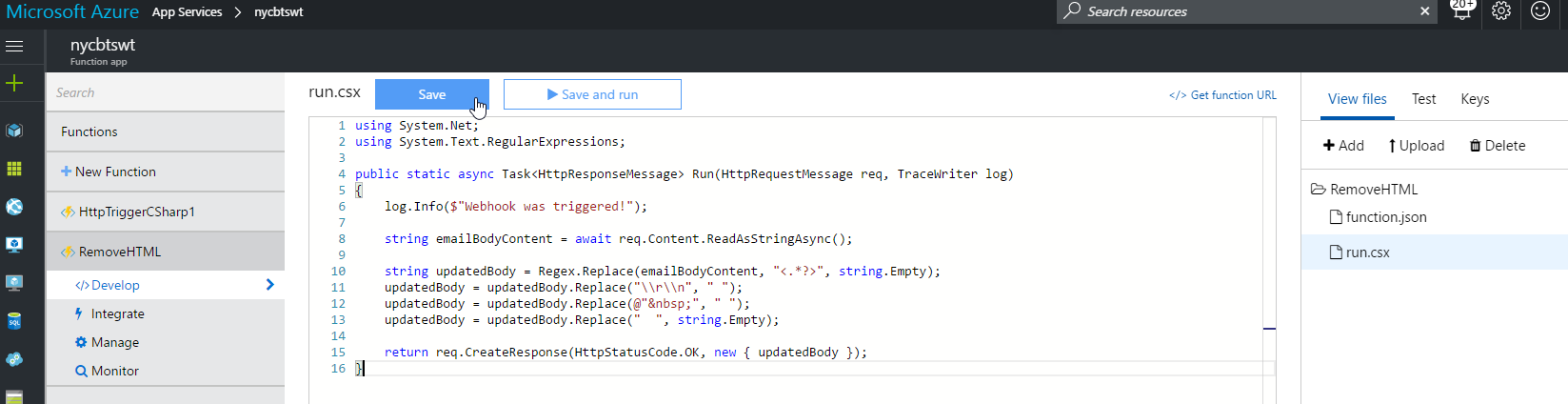
updatedBody = updatedBody.Replace(@"&nbsp;", " ");

updatedBody = updatedBody.Replace(" ", string.Empty);

return req.CreateResponse(HttpStatusCode.OK, new { updatedBody });

}

1. At the end, it should look like this:



1. Make sure you press **Save** on your function.
2. You can test your function by clicking on **Test** on the right. It might be under the **<** icon on the right.   
     
   Use the following to test the function:

{

"name": "<P><P><BR>This is a test</BR></P></P>"

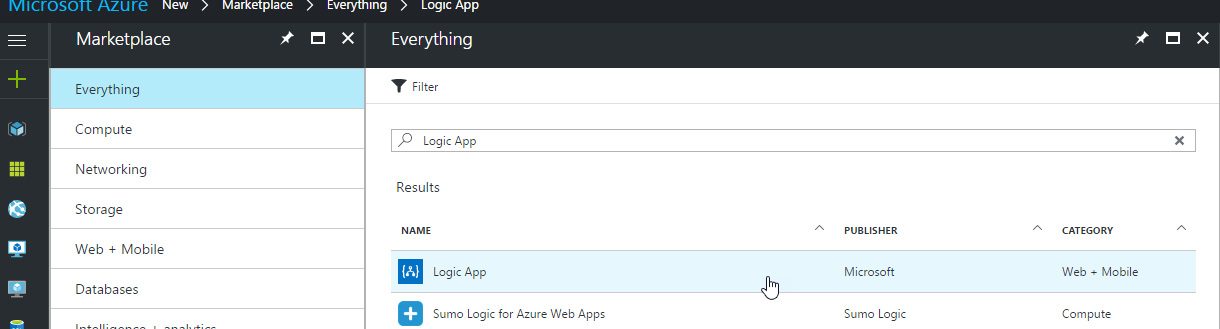
}

The output should look like:  
{"updatedBody":"{\n\"name\": \"This is a test\"\n}"}

# Creating the Logic App

In this section, you will put all the pieces together inside a Windows Azure Logic App. It is important to save your work often. I usually save after each shape.

1. Log into the Windows Azure Portal, click on the green plus on the left to create a new item, and search for **Logic App**. Select Function App and click **Create** in the bottom right.



1. Configure the logic app with the following settings:

|  |  |
| --- | --- |
| 1. Name: **FeedbackEmailProcessing** 2. Subscription: *Select your subscription* 3. Resource group: *Select resource group already created* (**NYC-Bootcamp)** 4. Location: **East US 2** 5. Click **Create** |  |

1. Navigate back into the newly created Logic App by going to Logic Apps inside the Azure Portal. Click on the newly created logic app: **FeedbackEmailProcessing**.
2. Click the **Blank Logic App** template to set started.
3. From the Trigger drop down, select **Office 365 Outlook**. Then select **When a new email arrives** trigger. Then click **Sign In** to create the connection using the credentials provided or using your own account.

|  |  |
| --- | --- |
|  |  |

1. Expand the **Show advanced options** to finish the configuration. Check the following settings:

|  |  |
| --- | --- |
| 1. Include Attachments: **Yes** 2. Subject Filer: *Something specific to you like a key word or initials*   Note: The adapter is a bit silly. It will tell you that you have email attachments but it doesn’t give you access to them unless you set Include Attachments = Yes.  Note: Each Logic App will check all new messages in the account. No emails are removed from the account. Only emails that match the Subject Filter are processed. |  |

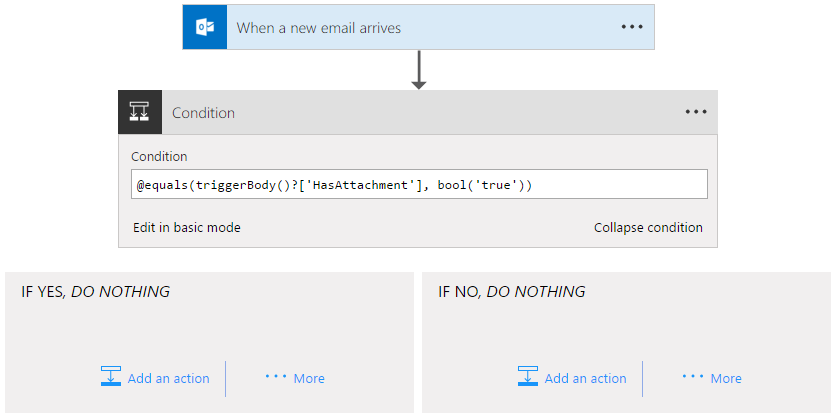
1. After receiving an email message, we are going to check to see if the email has any attachments.   
   Click on **+ New Step**, select **Add a condition**.

Click on **Edit in advanced mode**.

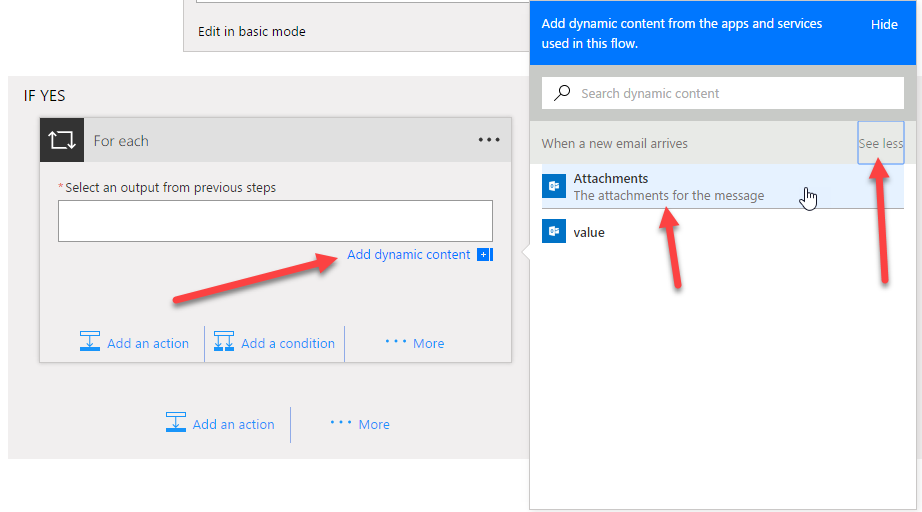
Set the condition to: **@equals(triggerBody()?['HasAttachment'], bool('true'))**

Note: This gets the values of the HasAttachment property of the triggerBody (which is the email) and compares it to the Boolean object of True.

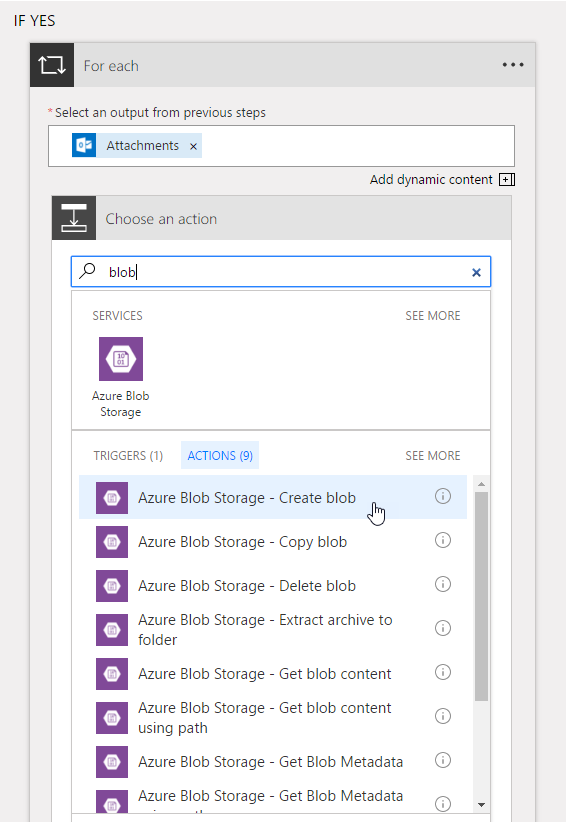
The Logic App should now look like:



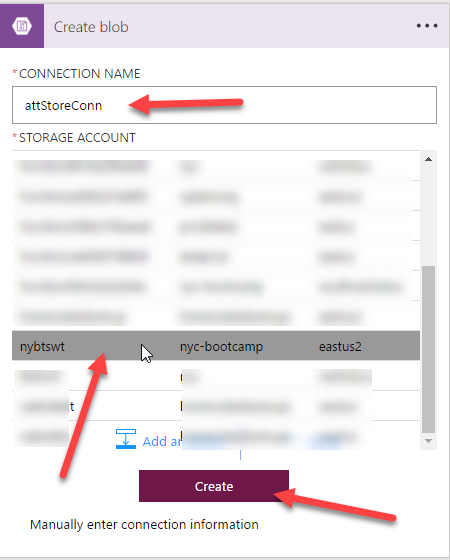
1. On the Yes side, we want to loop over each attachment and save them to blob storage in Azure. To get started with this, click on **… More** under If Yes, DO NOTHING. Select **Add a for each**.
2. Click on **Add dynamic content**. Then on the right click **See more** (picture below says See less because I already clicked on it). Select **Attachments**. This will loop over all the attachments in the email.



1. Click on **Add an action** inside the For each. In the search box, type **Blob**. Select **Azure Blog Storage – Create blob**.



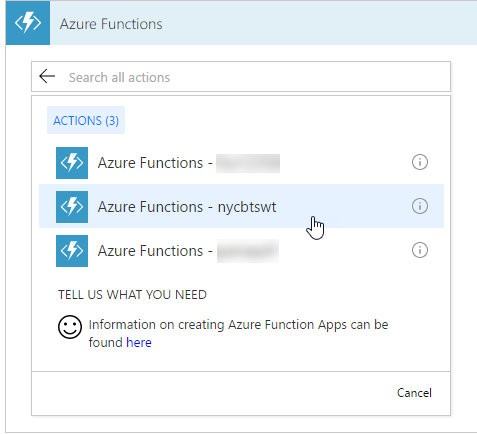
1. Set Connection Name to **attStoreConn** and select your storage account from the list. Click **Create**.



1. Set the following properties on the shape:

|  |  |
| --- | --- |
| 1. Folder path: **/attachments** 2. Blob name: **Dynamic Content: Name** (name of the attachment) 3. Blob content: **Dynamic Content: Content** (content of the attachment) |  |

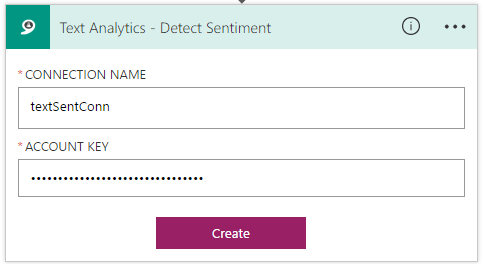
1. Save your Logic App and test the process thus far. Inside the designed, click on **Run**. Send an email to your email account with the matching subject. Include 2 or more attachments. Using your Azure Storage tool, check to ensure your attachments are saved to storage.
2. Now we will remove HTML from the body of the email and check the sentiment. To get started with this, we need to call our function. Click the **+ Add an action**. Select **Azure Functions** from the top menu. Select **Azure Functions – Choose an Azure function**. Select your function created above. If you have more than one function in this Function App, you will them be promoted to select the one you want to call with this shape. Select **RemoveHTML**.



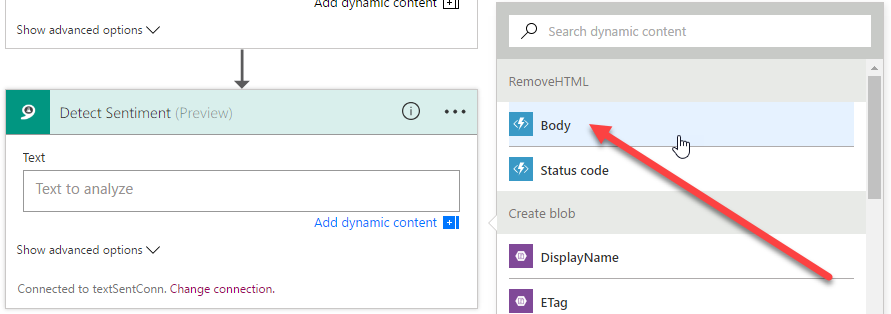
1. The Request Body needs to be set for the Function. The input to the function is JSON. So we need to format the request as such. We want to pass in the Dynamic Content of the email body. To do so, type the following: **{ “emailBody”:** Then select from the Dynamic Content on the right. Select the **Body** of the email. Add the trailing **}** after you select it. See below.

|  |  |
| --- | --- |
|  |  |

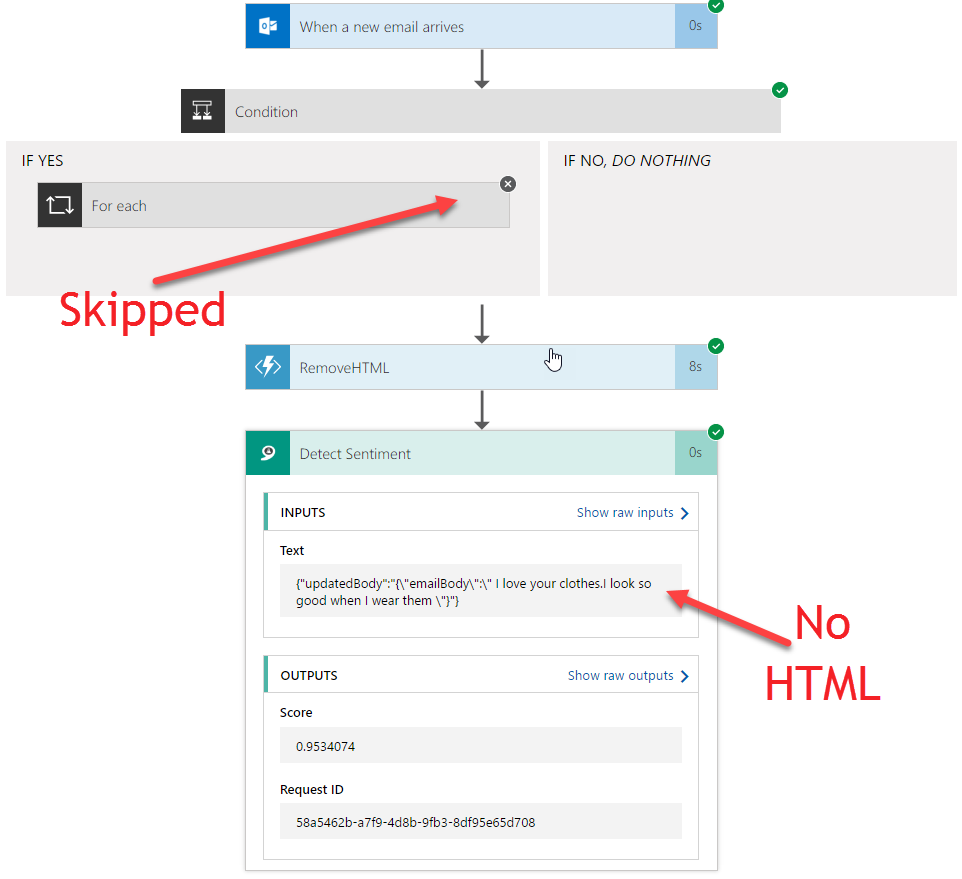
1. Now we need to take the output of the Function and run it through Sentiment Analysis. Click the **+ New Step** and select **Add an action**. Search for **Test Analytics**. Select **Text Analysis – Detect Sentiment**. You will be promoted to create a new connect. Name the connection **textSentConn**. Enter your **Account Key** (connection key) for you text analytics service from above. Click **Create**.



1. In the Text box, select Dynamic Content – RemoveHTML – Body. This will use the output from the function as the input to this API call.



1. Save your Logic App and test the process thus far. Inside the designed, click on **Run**. Send an email to your email account with the matching subject. Include the following body “I love your clothes. I look so good when I wear them”. Make sure you remove any signatures from the email. Once it runs, the output should look like below. Click on the **Detect Sentiment** to open it up and see the Inputs and Outputs. Once done, click on **Designer** on the top to return to building your Logic App.



1. The last step in the process is to send a text if the score is too low. In order to do that, we will use Twilio. If you already have an account you can use your connection details. If not, ensure you get your phone number registered and use the API provided for todays lab.
2. Click on **+ New Step**, select **Add a condition**.

Click on **Edit in advanced mode**.

Set the condition to: **@less(body(Detect\_Sentiment')?['score'], float('.50'))**

Note: This gets the values of the Score property of the 'Check\_email\_body\_sentiment’ shape and compares it to the Float object of ‘.50’. If you do not cast the ‘.50’ to float you will get an invalid comparison exception at runtime.

1. On the Yes side, we want to send a text message. To do this, click on **Add an action**. Search for **Twilio**. Select **Twilio – Send Text Message (SMS)**. You will be promoted to create a new connection. Name the connection **twilioConn**. Enter your **Twilio Account Id** and **Twilio Access Token** (use the ones provided if you do not have a Twilio account). Click **Create**.
2. Configure the shape as follows:

|  |  |
| --- | --- |
| 1. From Phone Number: +1**4695027528** (if using the provided account) 2. To Phone Number: **Your Number** (with the +1) 3. Text: **“They have a score of" Dynamic Content: Score** |  |

1. Save your Logic App and test the end result. Inside the designed, click on **Run**. Send an email to your email account with the matching subject. Include the following body “I hate your clothes. I do not look good at all plus the quality is poor”. Make sure you remove any signatures from the email. Once it runs, you should see the path through the Logic App showing a text was sent. You should receive a text message with the sentiment scope.