

Final Project  
Social Media Monitoring via Logic Apps  
Prouty, Stephen

**Deep Azure@McKesson**

Dr. Zoran B. Djordjević

## Contents

Problem Statement.....	3
Overview of the Technology .....	3
Data Sources .....	3
Hardware / Software Used .....	3
High Level Project Overview .....	3
Project Step 1 – Azure SQL Database.....	5
Project Step 2 – Logic App with Social Media APIs .....	11
Project Step 3 – Create Aggregation Function App .....	17
Project Step 4 – Utilize Logic App to Execute the Function App.....	22
Project Step 5 – Generate Data Graph via Python.....	27
Summary .....	30
References .....	31

## Problem Statement

Social media applications have become part of our everyday lives. This is even more true for the younger generations. With smart phones becoming the next thing in the hands of a child shortly after they put down their baby bottles, parents are frequently left blind to the messages being sent by their children. Parents need a way to monitor the usage of these messaging applications to ensure appropriate and safe usage. Although there are commercial applications that can perform these tasks, they typically require software to be installed on the device being monitored. The goal of this project is to capture messages from these applications and load them into a central location where the information can be viewed by a parent.

"When past generations of parents let their children socialize, the setting was in-person and it was often easy to know what kids were doing. In fact, everyone kids knew either lived on their block or went to their school. Today, the internet and social media have evolved into the main social gathering spot for teens and kids. Forget the mall; Snapchat and Instagram provide the same social gratification as any real-life interaction. The problem? Who knows what's being sent, said...or saved."

~TeenSafe.com

<https://www.teensafe.com/blog/parents-monitor-childrens-social-med>

## Overview of the Technology

This project will use Azure Logic Apps to extract messages from social media applications. Logic App triggers will be defined to execute at regular intervals to retrieve messages from Instagram, Facebook, and Twitter via API connectors defined within the Logic App workflow. Message attributes will be inserted into tables in an Azure SQL database. A second Azure Logic app will be scheduled daily that will call an Azure Function App to aggregate the message data and send a notification email stating the data is ready. A Python script will then connect to the Azure SQL database and render a bar chart showing the daily message distribution.

## Data Sources

Instead of using a pre-generated sample set of data, this project will extract data directly from accounts setup in personal social media applications. Accounts will need to be created for Instagram, Facebook, and Twitter.

## Hardware / Software Used

All work was performed from a Windows 7 Professional 64-bit processor laptop with 16GB RAM. The following software components were installed to support the building and deploying of the applications within this document.

- Python 2.7.14 (<https://www.python.org/downloads/>)
- Matplotlib 2.1.2 (<https://matplotlib.org/>)
- Visual Studio Code 1.18 (<https://code.visualstudio.com/>)
- Microsoft SQL Server Management Studio 2017 (<https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms>)

## High Level Project Overview

The remainder of this report will detail the steps taken to solve the defined problem statement. Each of the following step will be further detailed in the next sections.

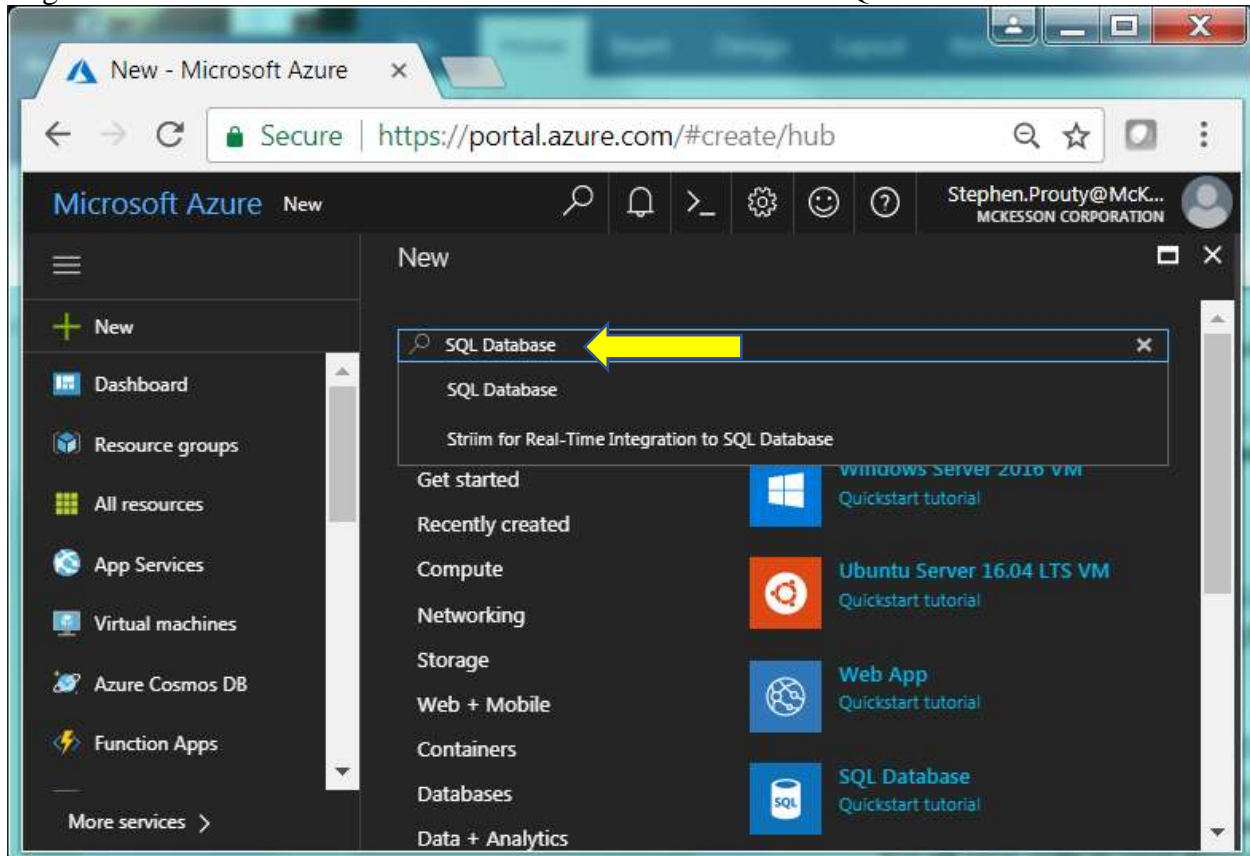
- 1) Create Azure SQL database and the message detail and summary tables.
- 2) Create Logic App with three API connectors: Instagram, Facebook, Twitter.
- 3) Create Function App to connect to Azure SQL database and aggregate the message data.
- 4) Define second Logic App to call Function App and send notification of completion via email.
- 5) Utilize Python client script to connect to Azure SQL database and generate trend graph.



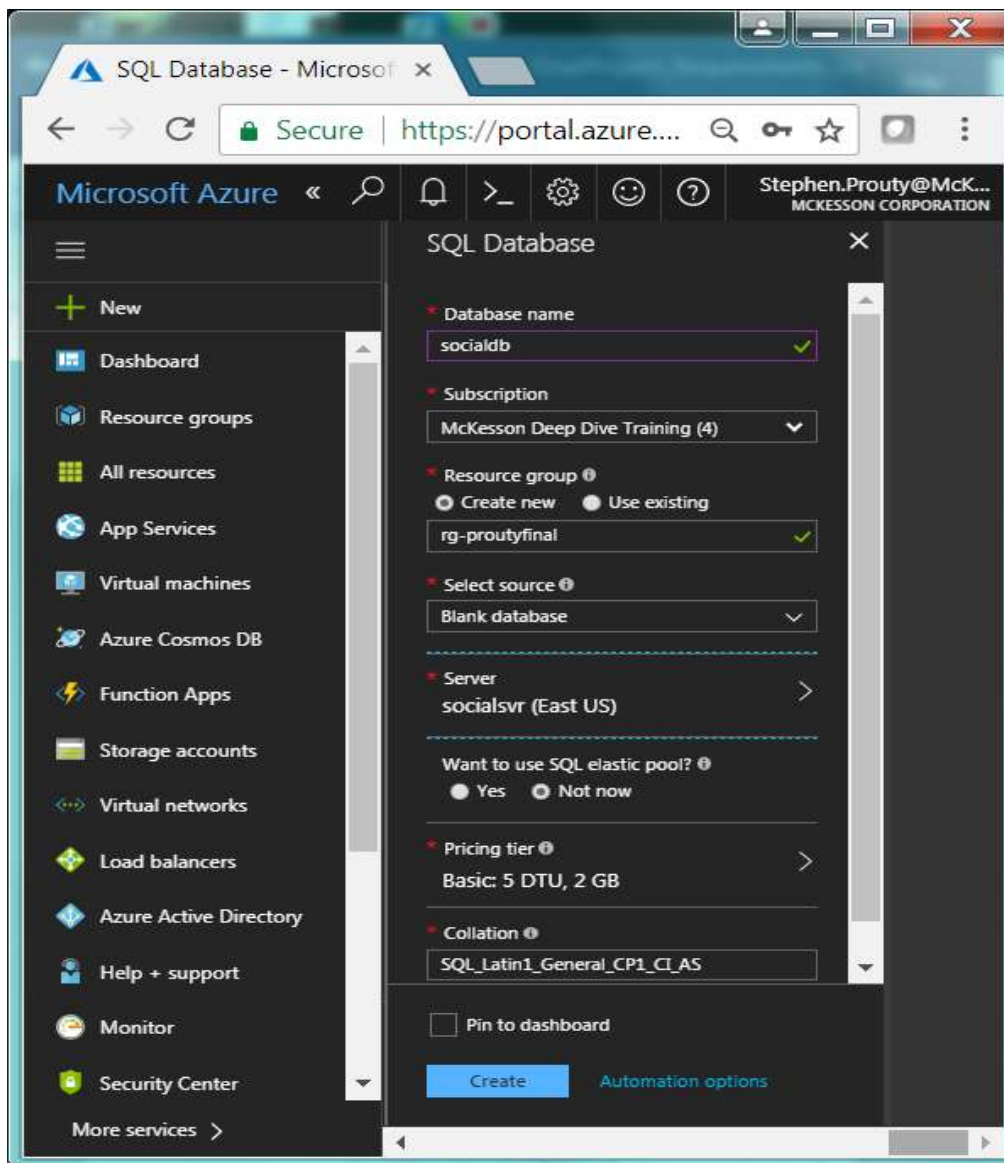
### Project Step 1 – Azure SQL Database

One of the requirements for this problem solution was to centralize all incoming social media messages into a central repository. This central repository needs to be easily accessible by the end user, the parent, for reviewing messages. The Azure SQL database was selected to fulfill this requirement. The following steps describe creating a new Azure SQL database.

Login to the Azure Portal and search for a new resource called “SQL Database”.



Select “Create” and fill in the SQL Database information. Since this project is only for demonstration purposes, select the “Basic” pricing tier to keep the cost low.



After creating the Azure SQL database, the table structures need to be created that will store the incoming messages from the social media sites. There will be two tables created, one for detailed records and one for aggregated data. Each table definition is shown below.

#### MESSAGE DETAIL

MSG_DATE	DATE
SOURCE	TEXT
MSG_TEXT	TEXT

#### MESSAGE SUMMARY

SUMMARY_DATE	DATE
SOURCE	TEXT
MSG_COUNT	INTEGER

The following tables show example data for each table.

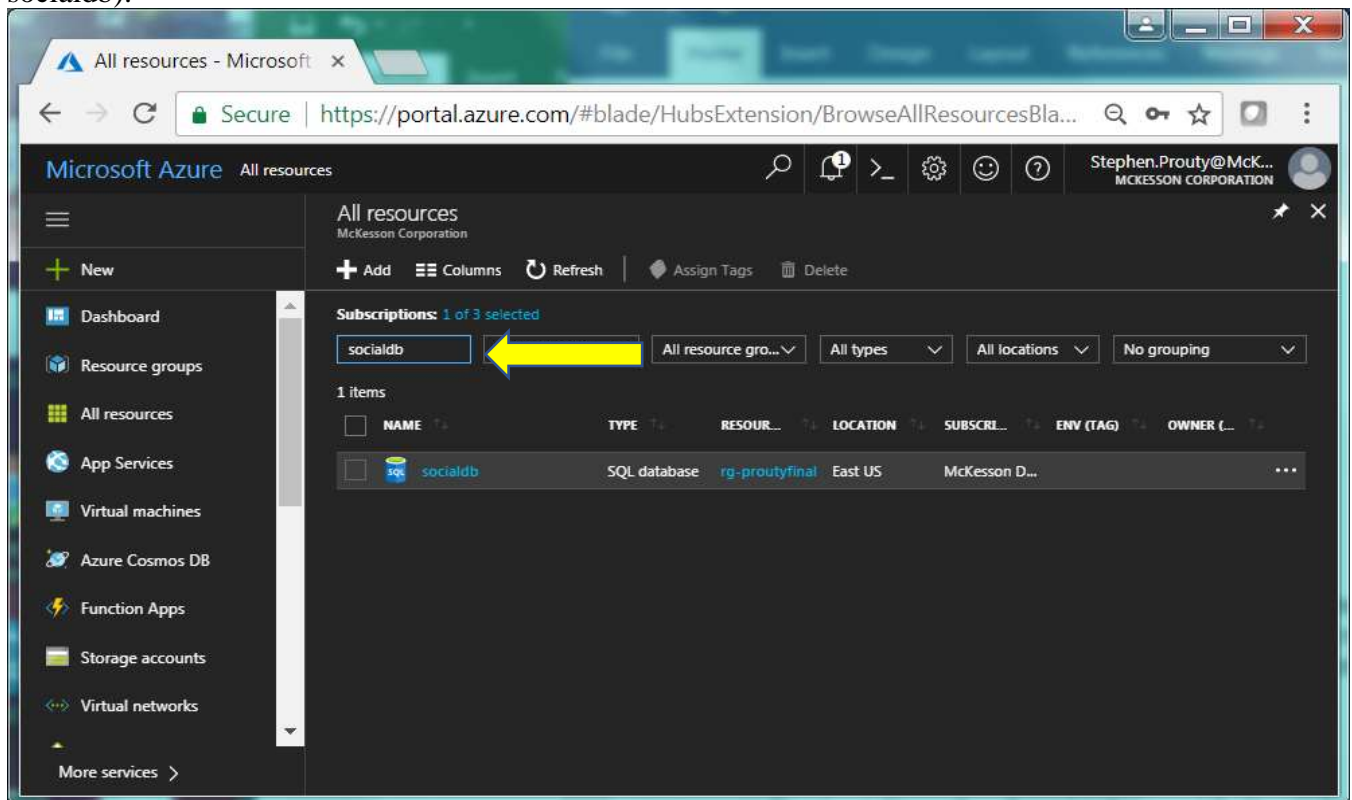
**MESSAGE\_DETAIL**

MSG_DATE	SOURCE	MSG_TEXT
2/6/2018	Instagram	Check out my vacation
2/6/2018	Twitter	On my way to work
2/6/2018	Twitter	Working on Logic App homework
2/6/2018	Facebook	Super Bowl weekend!!
2/7/2018	Instagram	Another cold day in Pittsburgh
2/7/2018	Facebook	Ski slopes of PA
2/9/2018	Twitter	Working late tonight. Very tired
2/9/2018	Twitter	TGIF. Work week is over
2/8/2018	Instagram	Friday night dinner plans

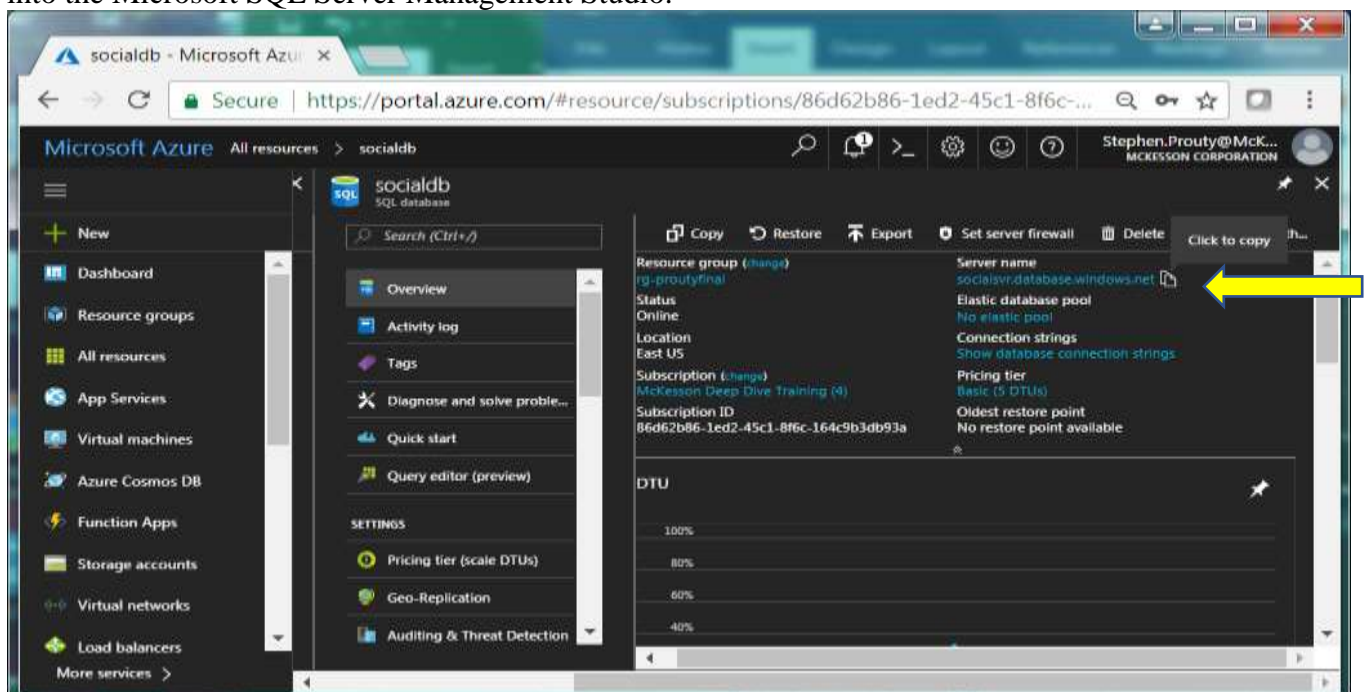
**MESSAGE\_SUMMARY**

MSG_DATE	SOURCE	MSG_COUNT
2/6/2018	Instagram	1
2/6/2018	Twitter	2
2/6/2018	Facebook	1
2/7/2018	Instagram	1
2/7/2018	Facebook	1
2/7/2018	Twitter	0
2/9/2018	Instagram	1
2/9/2018	Facebook	0
2/8/2018	Twitter	2

With the Azure SQL database now created and the logical tables defined, use the Microsoft SQL Server Management Studio 2017 software to create the tables. First, the connection string for the new database must be determined. Connect to the Azure portal and search for the database (ex. socialdb).

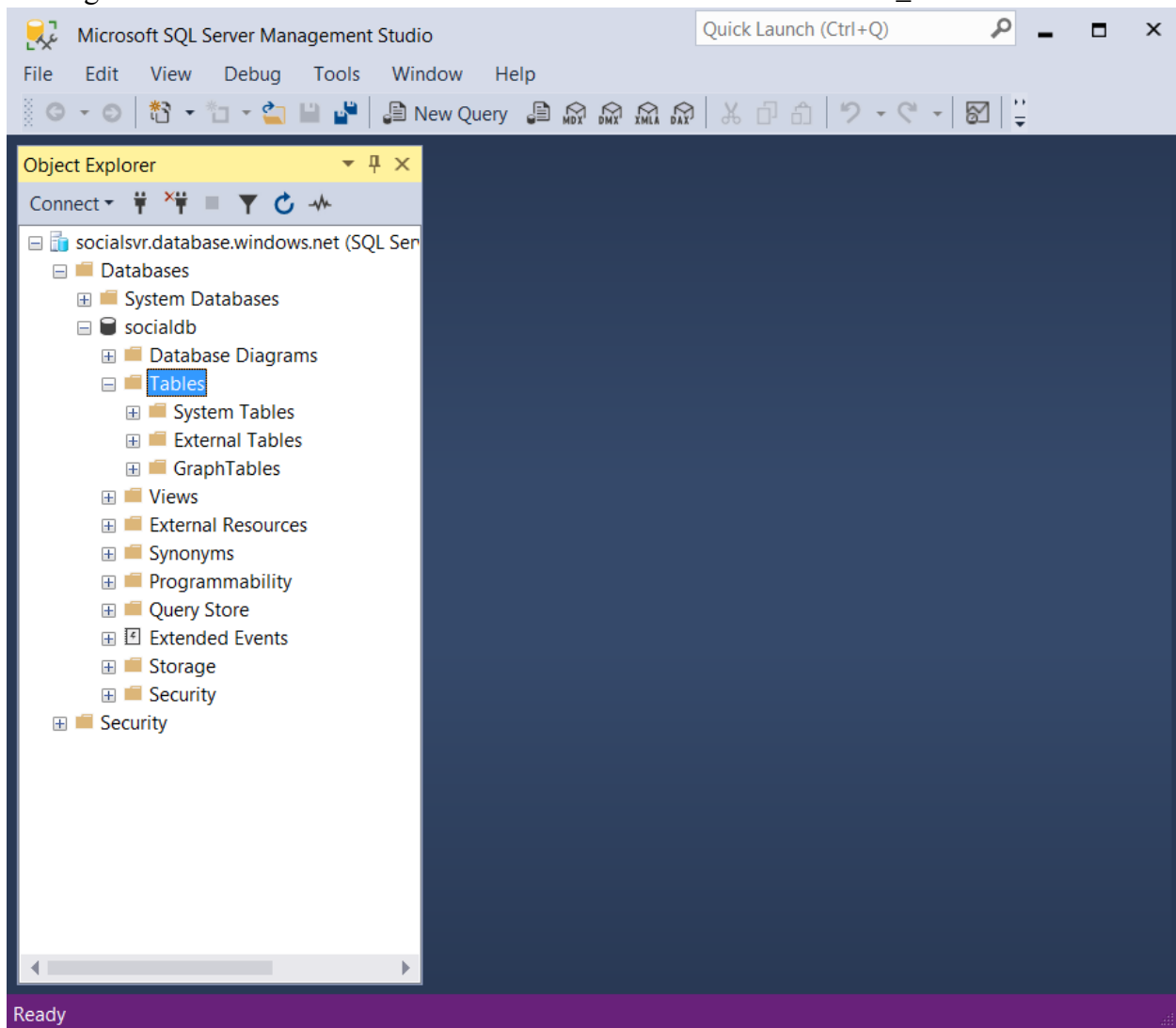


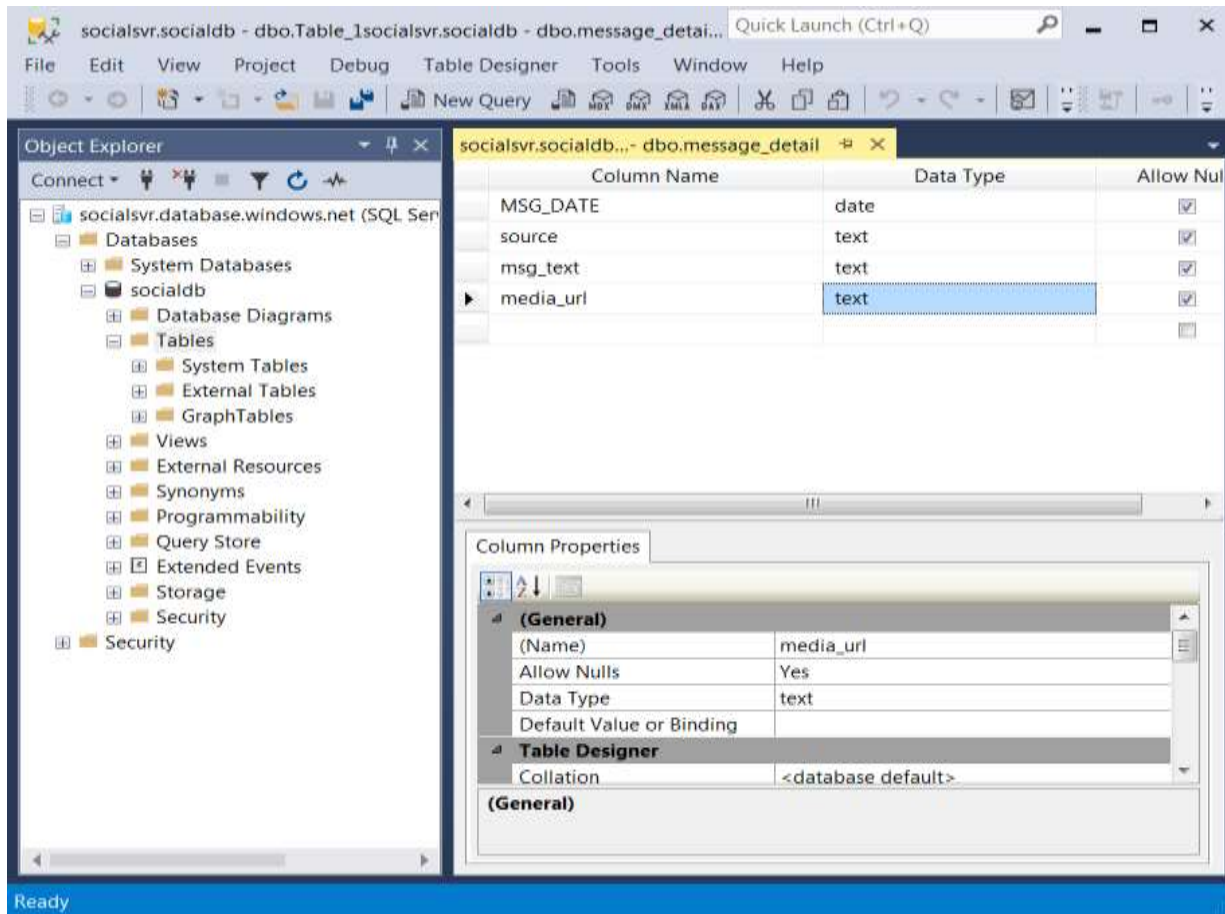
From the overview tab, copy the “Server Name”. This Server Name will be used when logging into the Microsoft SQL Server Management Studio.





After logging into the SQL Server Management Studio, browse to the newly created database then right click on “Tables” and select “New” to create a new MESSAGE\_DETAIL table.





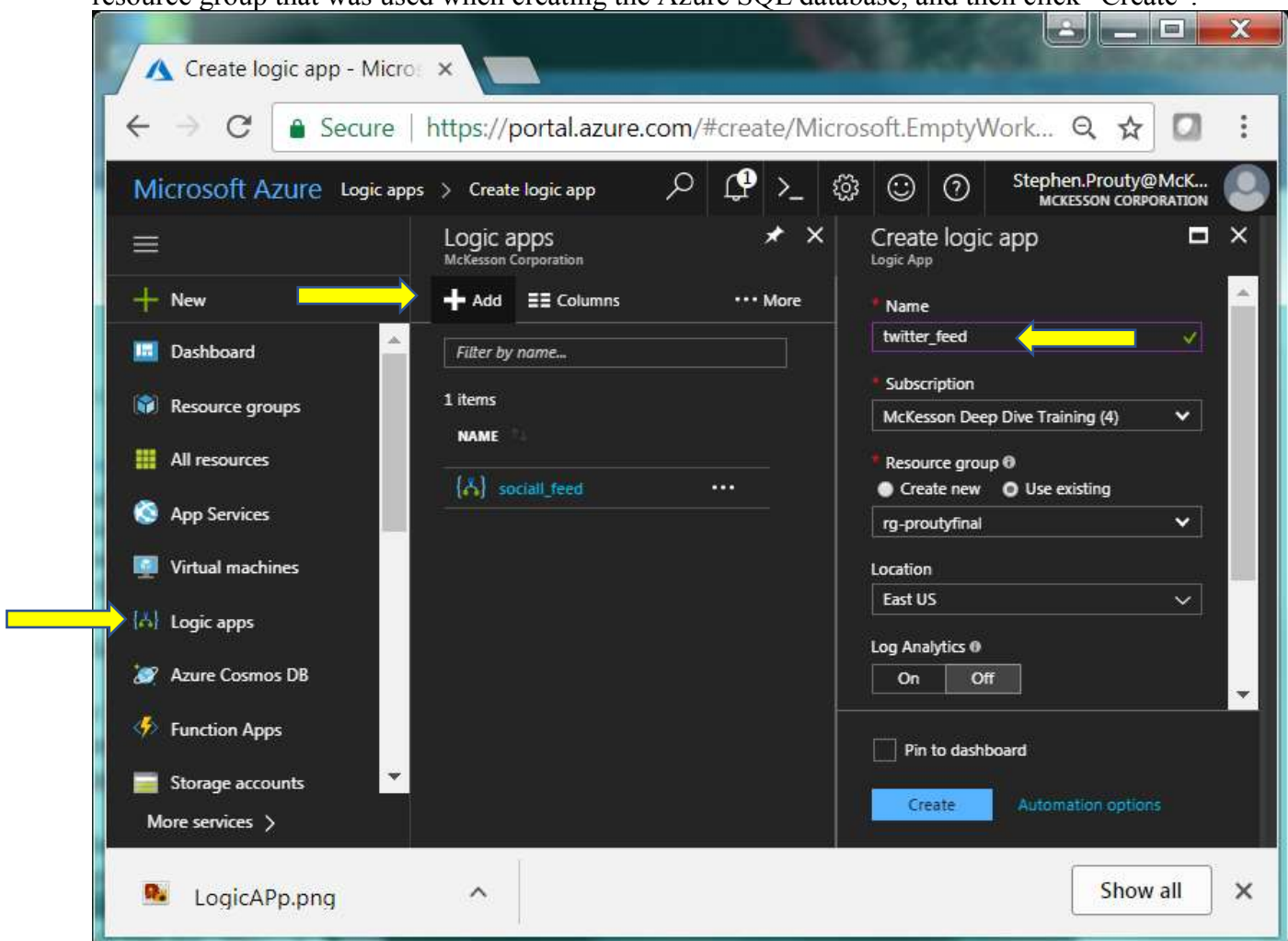
Repeat this same process for the MESSAGE\_SUMMARY table.

## Project Step 2 – Logic App with Social Media APIs

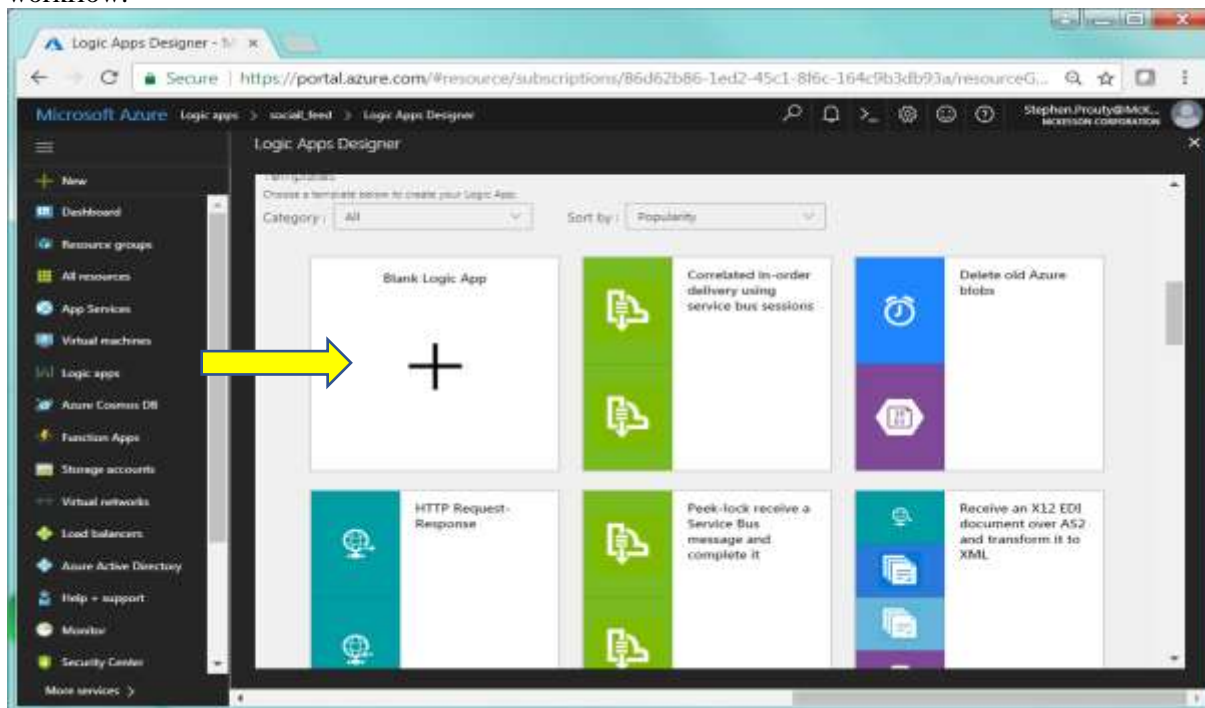
With the central data storage now defined, the Logic Apps can be created to extract data from the social media applications. For this solution three social media applications will be utilized: Instagram, Facebook, and Twitter. Prior to creating the Logic Apps be sure that you have the account information for each of these applications. The build process will require access to each. For this solution, the Logic App will be built using the Azure Portal. However, Logic Apps can also be designed and deployed from Visual Studio.

**Note:** The following actions to create new Logic Apps will be repeated for each of the social media connections. At the end of Step 2 you will have created three new Logic Apps (twitter\_feed, Instagram\_feed, and facebook\_feed).

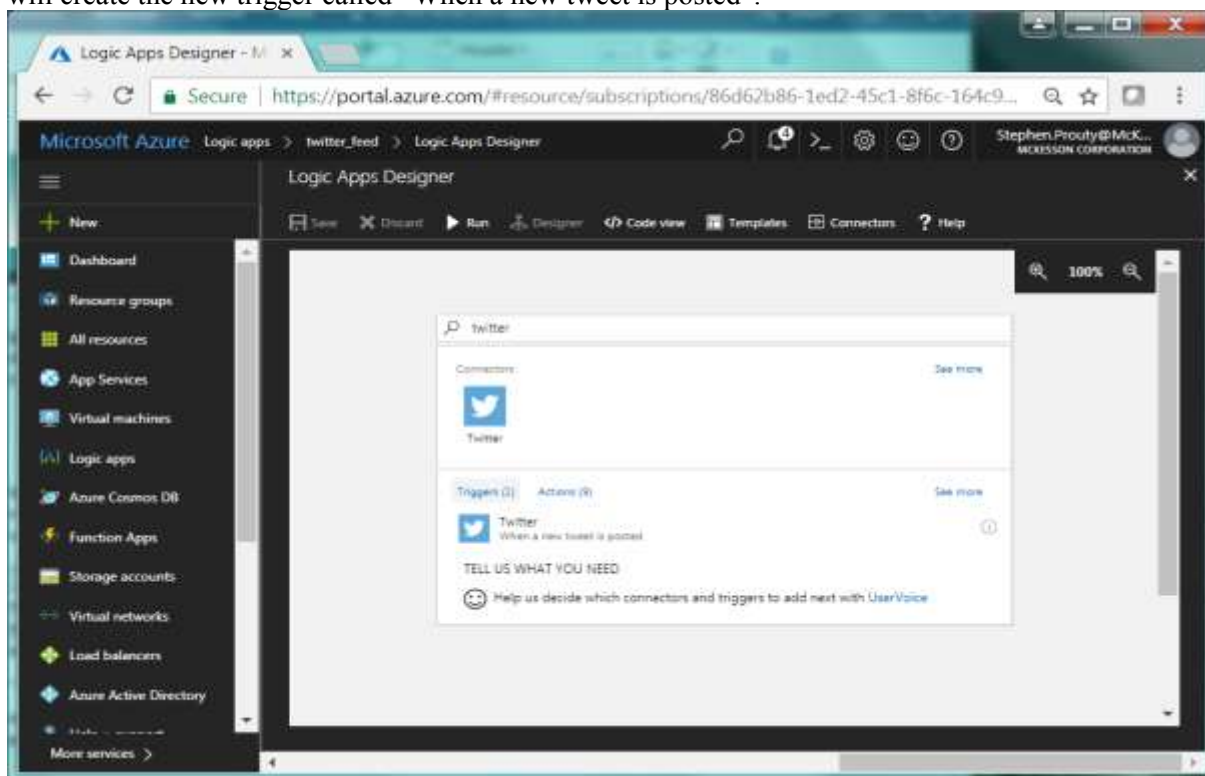
Login to the Azure Portal and add a new Logic App called “twitter\_feed”. Use the same resource group that was used when creating the Azure SQL database, and then click “Create”.



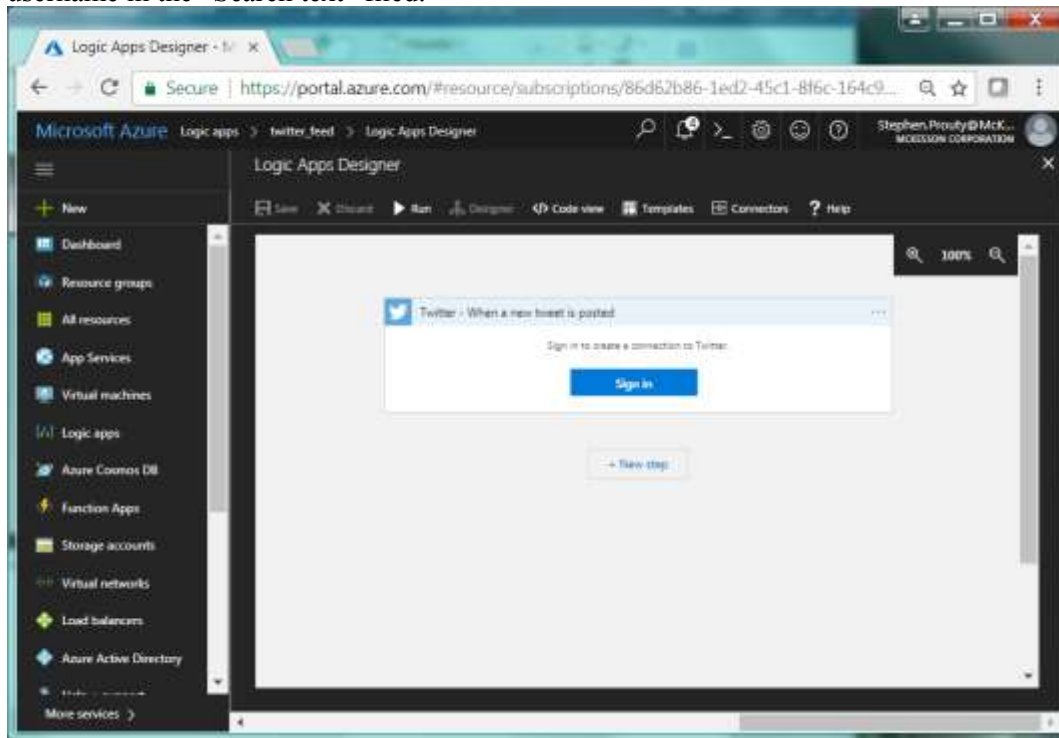
Once the Logic App resource is created, select the “Blank Logic App” template to begin building the new workflow.



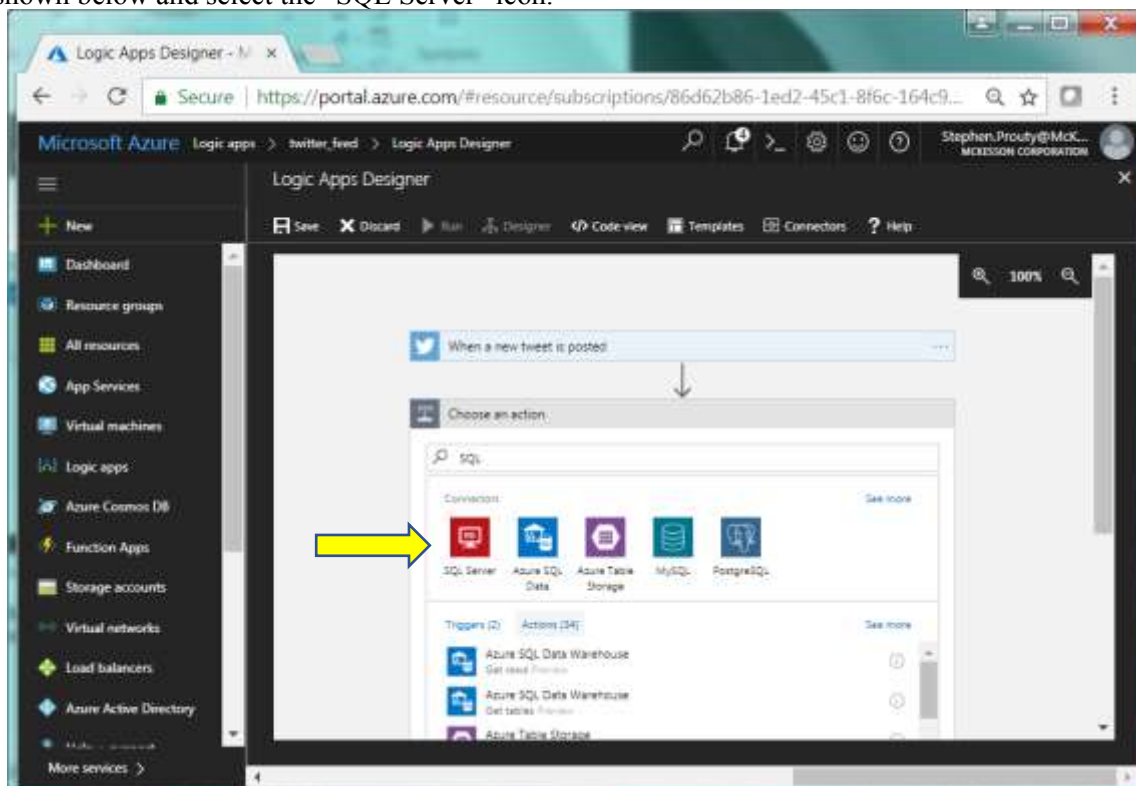
Selecting the “Blank Logic App” will open the Logic Apps Designer as shown below. This is the main interface that will be used to design the remainder of the workflow process. Start by searching for the desired trigger event that will launch the Logic App. In this case, search for the word “Twitter”. This will create the new trigger called “When a new tweet is posted”.



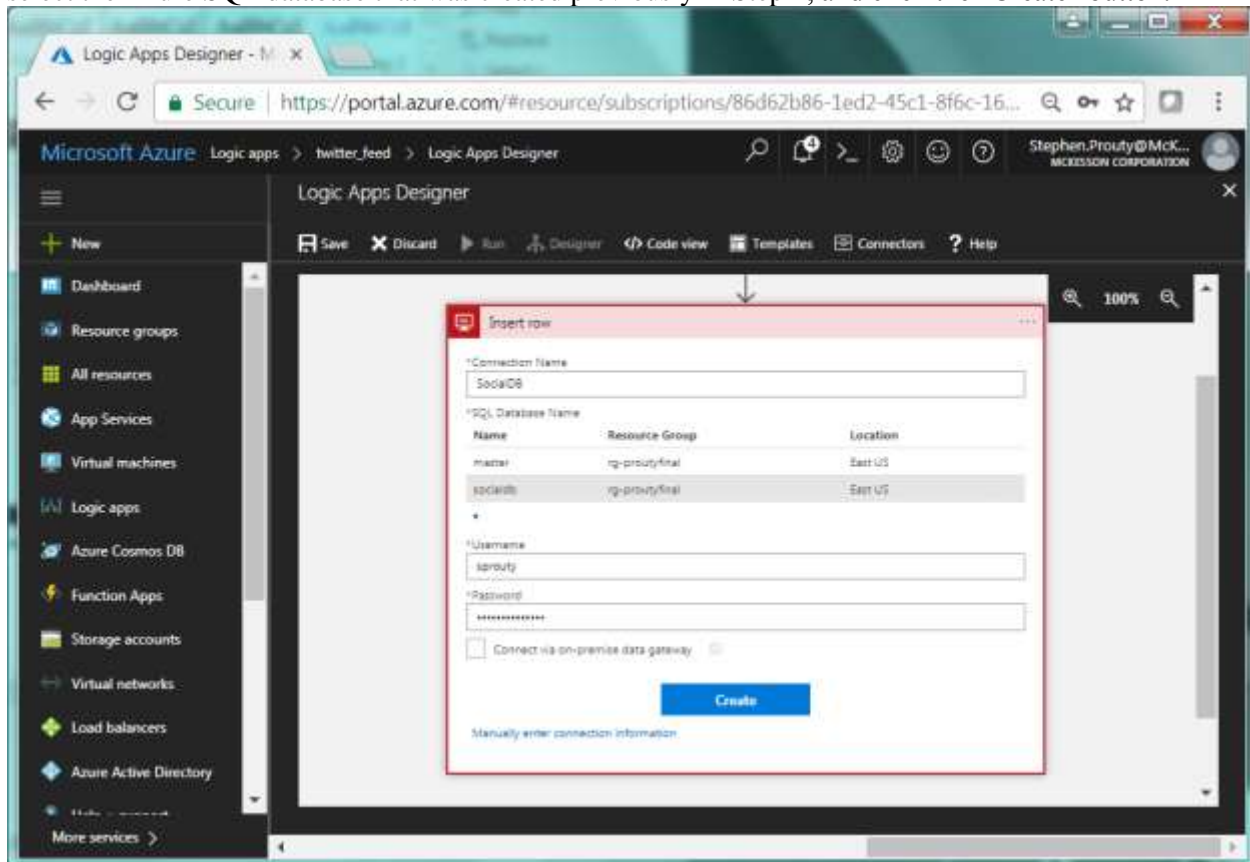
If the connector API has not already been created then the following authentication screen will be displayed. Click on the “Sign In” button to authenticate the service. Once authenticated add your Twitter username in the “Search text” field.



Next, add the Action to load the new Twitter messages into the previously created Azure SQL database. This is done by clicking on the “+ New step” button and selecting “Add an action”. Search for “SQL” as shown below and select the “SQL Server” icon.



After choosing the SQL Server connection, select the “Insert Row” action. Name the new connection and select the Azure SQL database that was created previously in Step 1, and click the “Create” button.

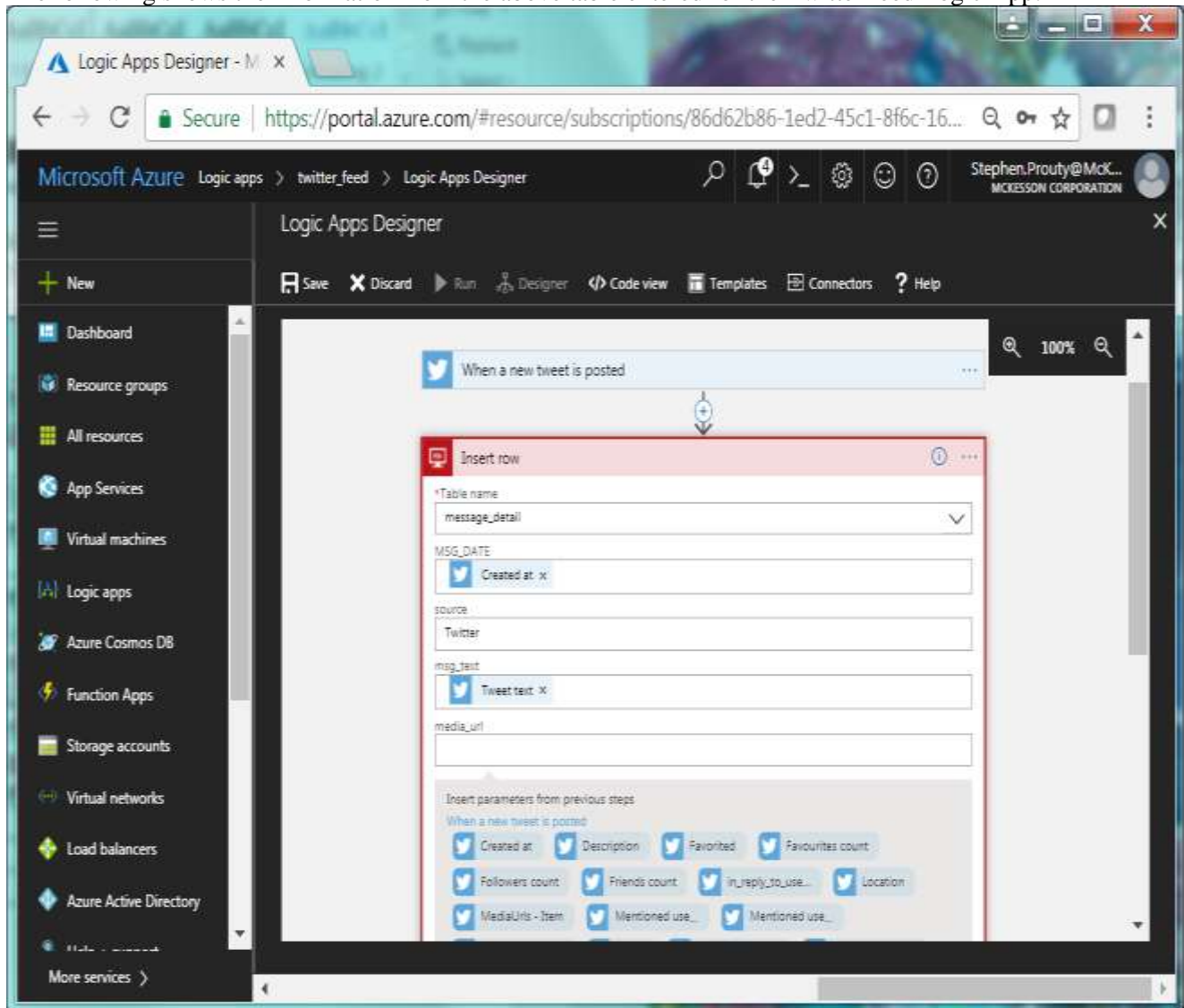


Once the SQL connector is created, the next step is to define the table to receive the inserted data as well as the message attributes that will be inserted. The following table shows the values to be entered for each social media application. Each application will insert data into the MESSAGE\_DETAIL table

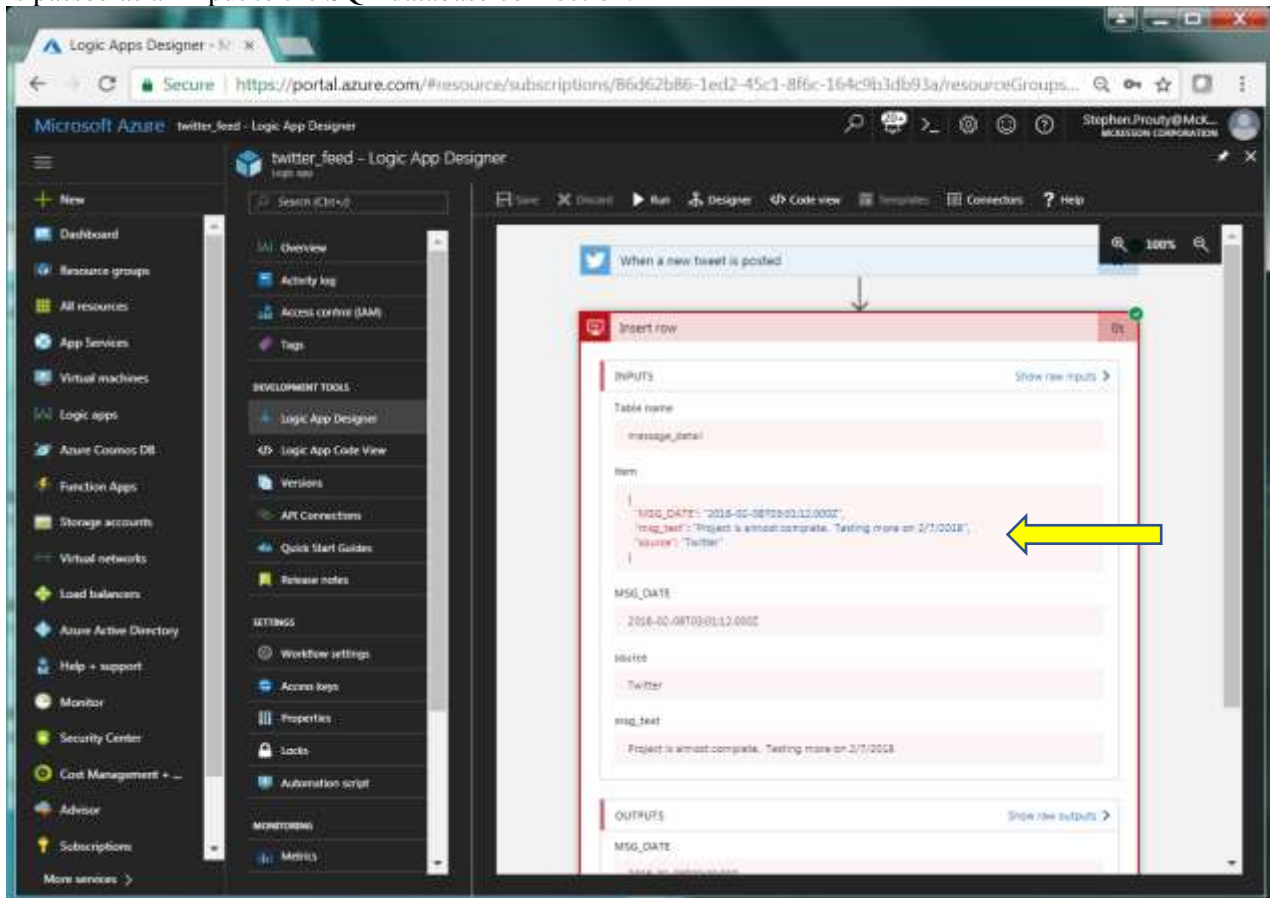
Media	Database Column	Media Attribute
Twitter	msg_date	CreatedAt
	source	Twitter
	msg_text	Tweet Text
	media_url	Null (future use)
Instagram	msg_date	@utcnow() – Instagram API doesn't provide the date/time of posting.
	source	Instagram
	msg_text	Media List Media Caption Text
	media_url	Null (future use)
Facebook	msg_date	Timeline Feed Feed Item Created Time
	source	Facebook
	msg_text	Timeline Feed Feed Item Status Message
	media_url	Null (future use)



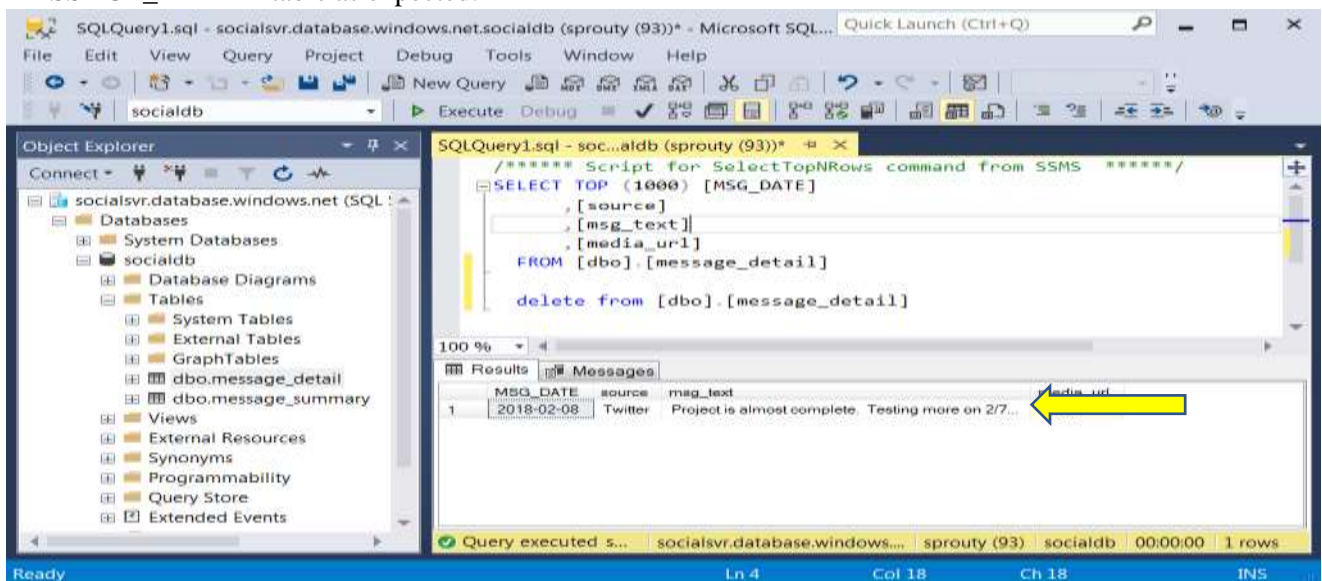
The following shows the information from the above table entered for the Twitter feed Logic App.



Save the Logic App and test it by selecting the “Run” option. Next, tweet a new message from the Twitter app or web site. The Logic App twitter trigger will fire and execute the workflow actions as shown below. A new Twitter message with text “Project is almost complete. Testing more on 2/7/2018” is passed as an input to the SQL database connection.



The data can also be seen using SQL Server Management Studio in the Azure SQL database MESSAGE\_DETAIL table as expected.



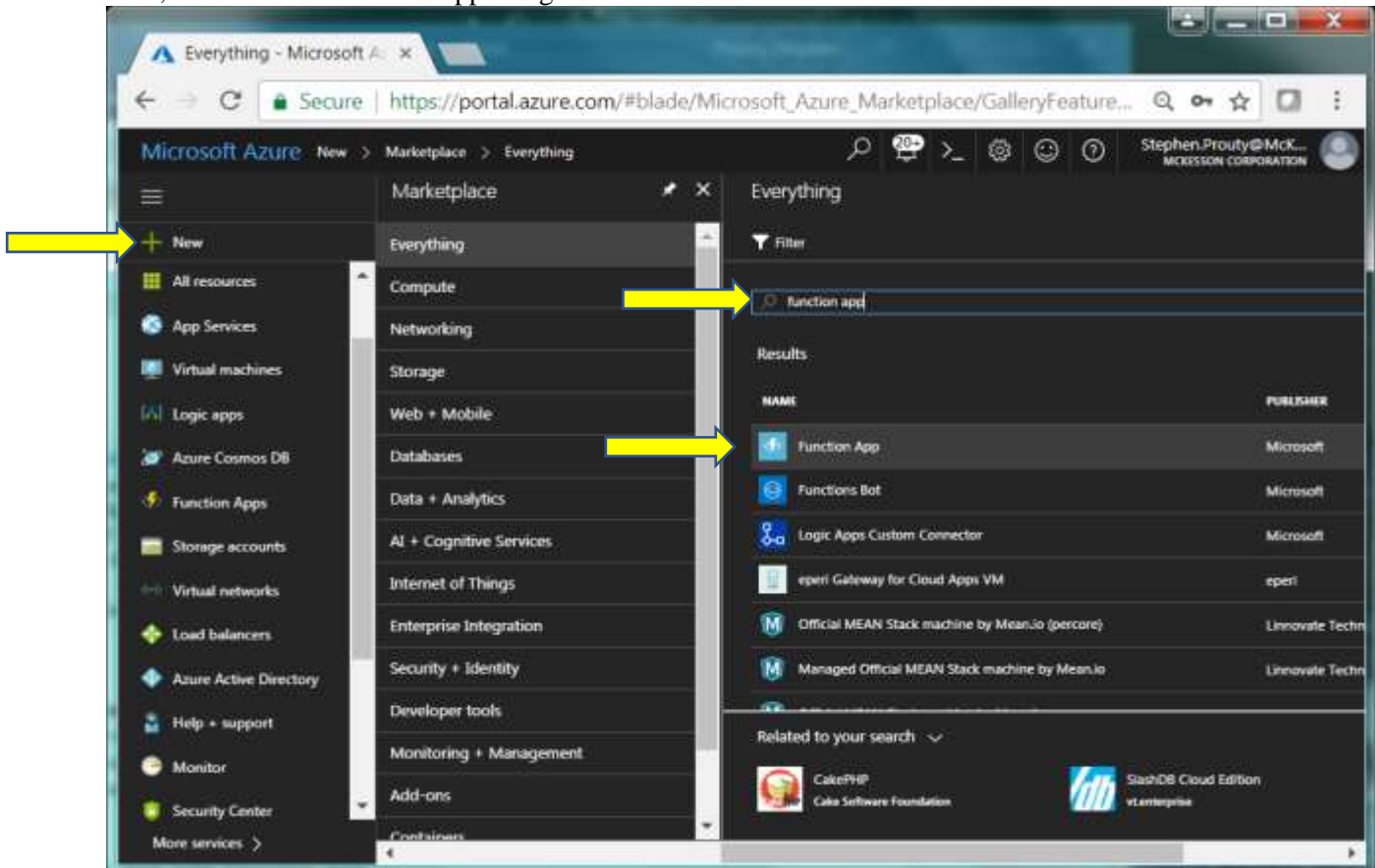
Repeat these same processes in Step 2 to add the social media connectors for Facebook and Instagram.



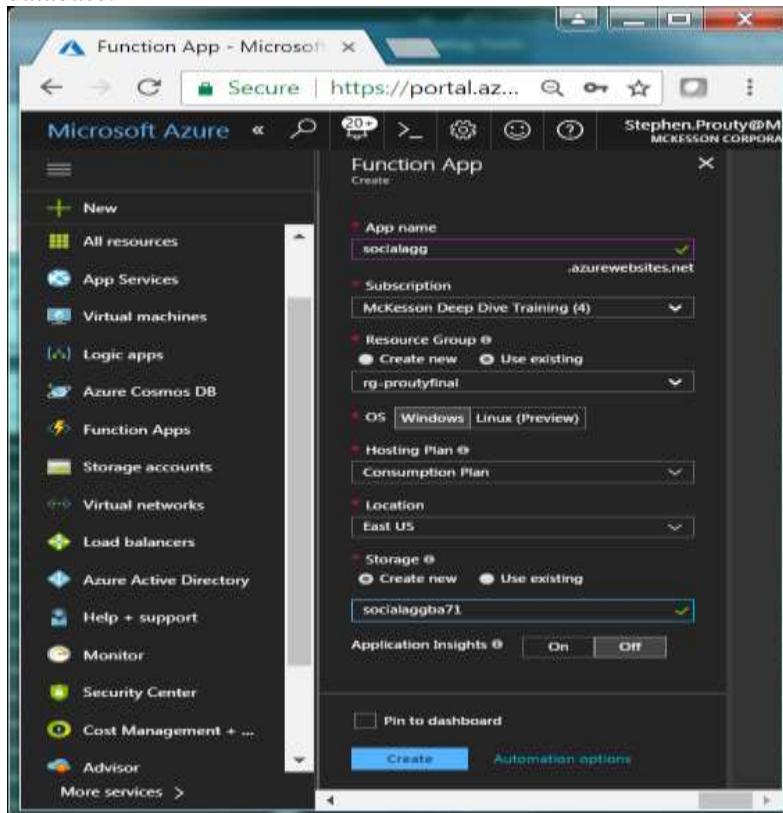
### Project Step 3 – Create Aggregation Function App

Now that we are able to extract messages from the three social media applications, the next step is to aggregate the data from the MESSAGE\_DETAIL table into the MESSAGE\_SUMMARY table. The MESSAGE\_SUMMARY table will store the total messages sent from each application by date. This data will later be presented to the end user via a Python script that graphs the data into a bar chart.

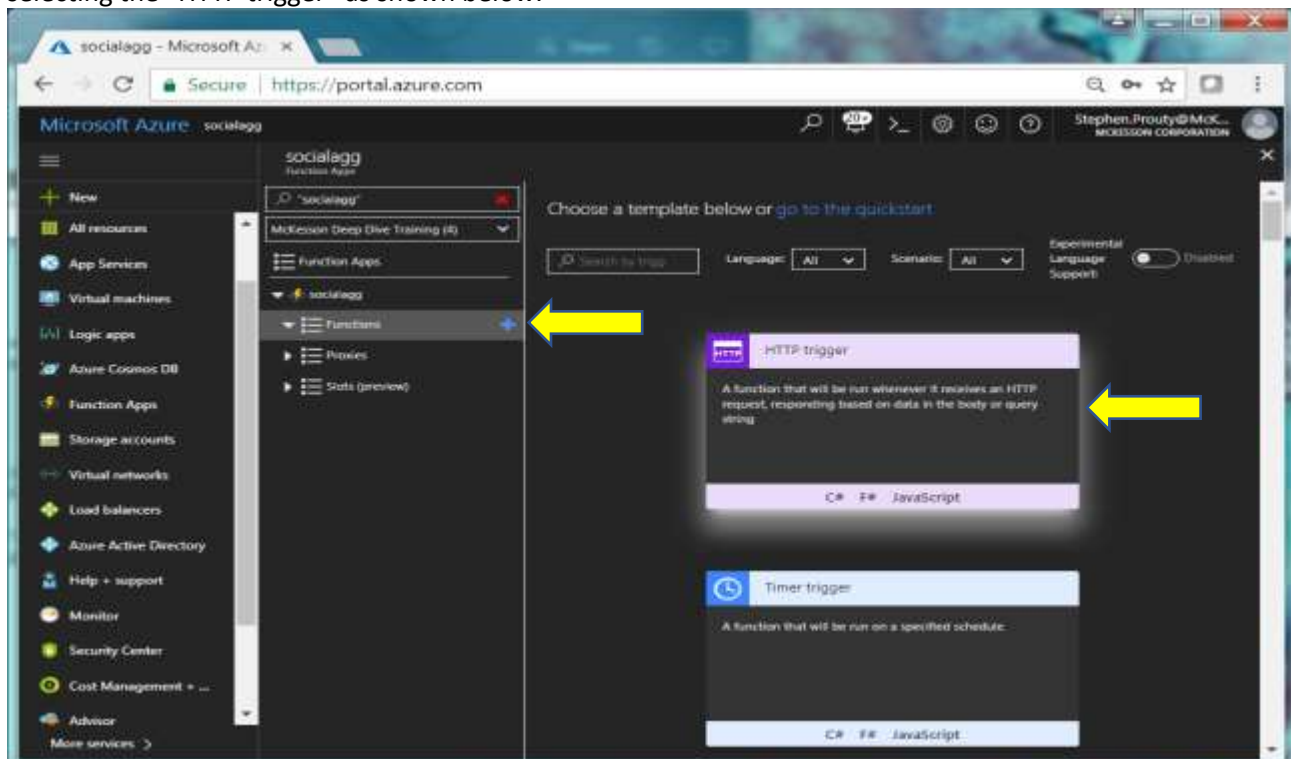
First, create a new Function App using the Azure Portal as shown below.



Give the new Function App a name and assign it to the resource group previously used for the Azure SQL database.



After the new Function App is built, add a new C# function by clicking on “Custom functions” and selecting the “HTTP trigger” as shown below.



Add the following code to the Function App. This code will connect to the Azure SQL database and insert aggregated data into the MESSAGE\_SUMMARY table from the MESSAGE\_DETAIL table.

```
#r "System.Configuration"
#r "System.Data"

// Load the required libraries
using System.Net;
using System.Configuration;
using System.Data.SqlClient;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter
log)
{
    // Define SQL statement to aggregate data into MESSAGE_SUMMARY table
    string sqlStmt = @"MERGE message_summary AS ms
        USING ( SELECT msg_date,
                        CONVERT(VARCHAR(8000), source) as source,
                        COUNT(*) as msg_cnt
                    FROM message_detail
                    GROUP BY msg_date,
                        CONVERT(VARCHAR(8000), source)) AS md
        ON (CONVERT(VARCHAR(8000), ms.source) = md.source AND
            ms.summary_date = md.msg_date)
        WHEN NOT MATCHED
            THEN INSERT(summary_date, source, msg_count)
            VALUES(md.msg_date, md.source, md.msg_cnt)
        WHEN MATCHED
            THEN UPDATE SET ms.msg_count = md.msg_cnt;";

    // Build SQL connection string based on Logic App connection configuration
    string connectionString =
        ConfigurationManager.ConnectionStrings["socialdb"].ConnectionString;

    // Connect to the SQL DB and prepare the statement handler
    using (SqlConnection dbCon = new SqlConnection(connectionString))
    using (SqlCommand dbCmd = dbCon.CreateCommand())
    {
        // Define the SQL command to be executed.
        dbCmd.CommandText = sqlStmt;

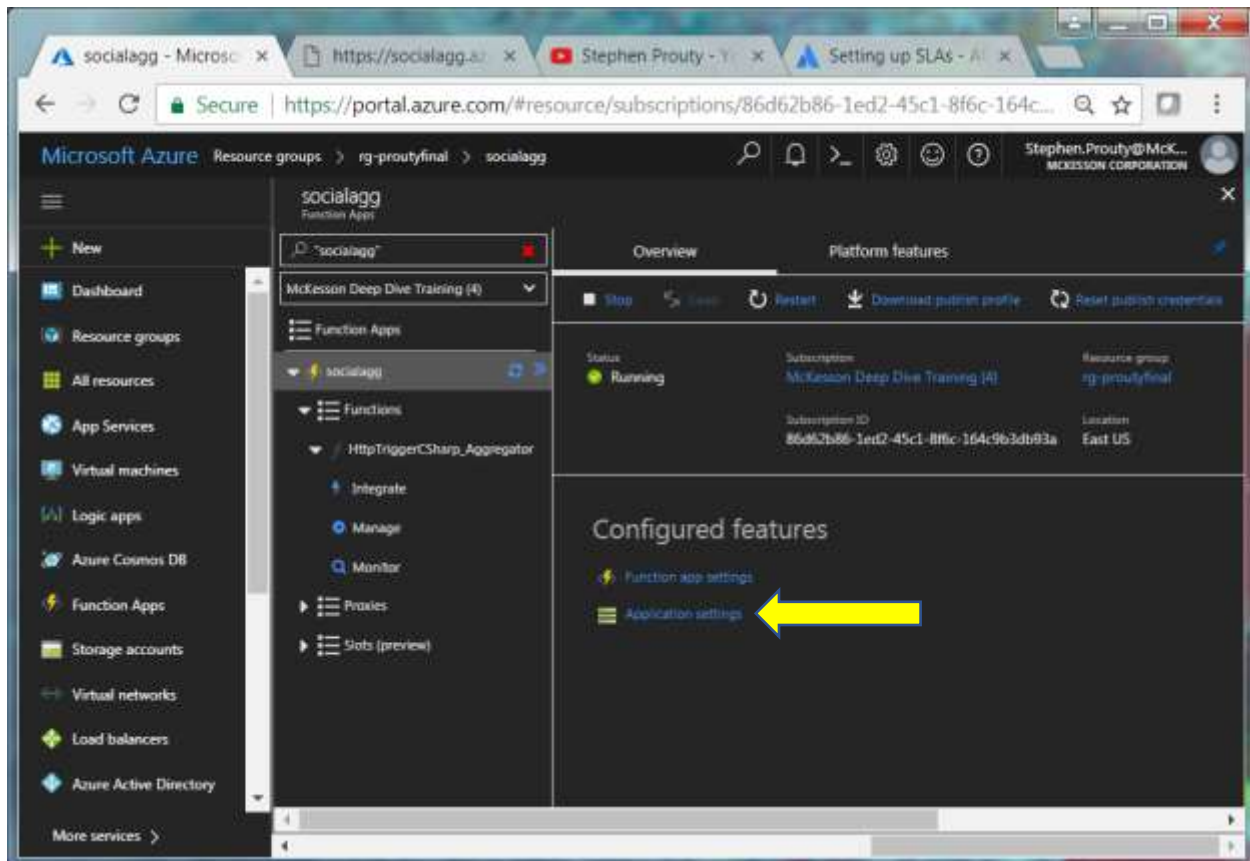
        // Open the SQL cursor for execution
        dbCon.Open();

        // Execute the SQL Insert statement
        dbCmd.ExecuteNonQuery();

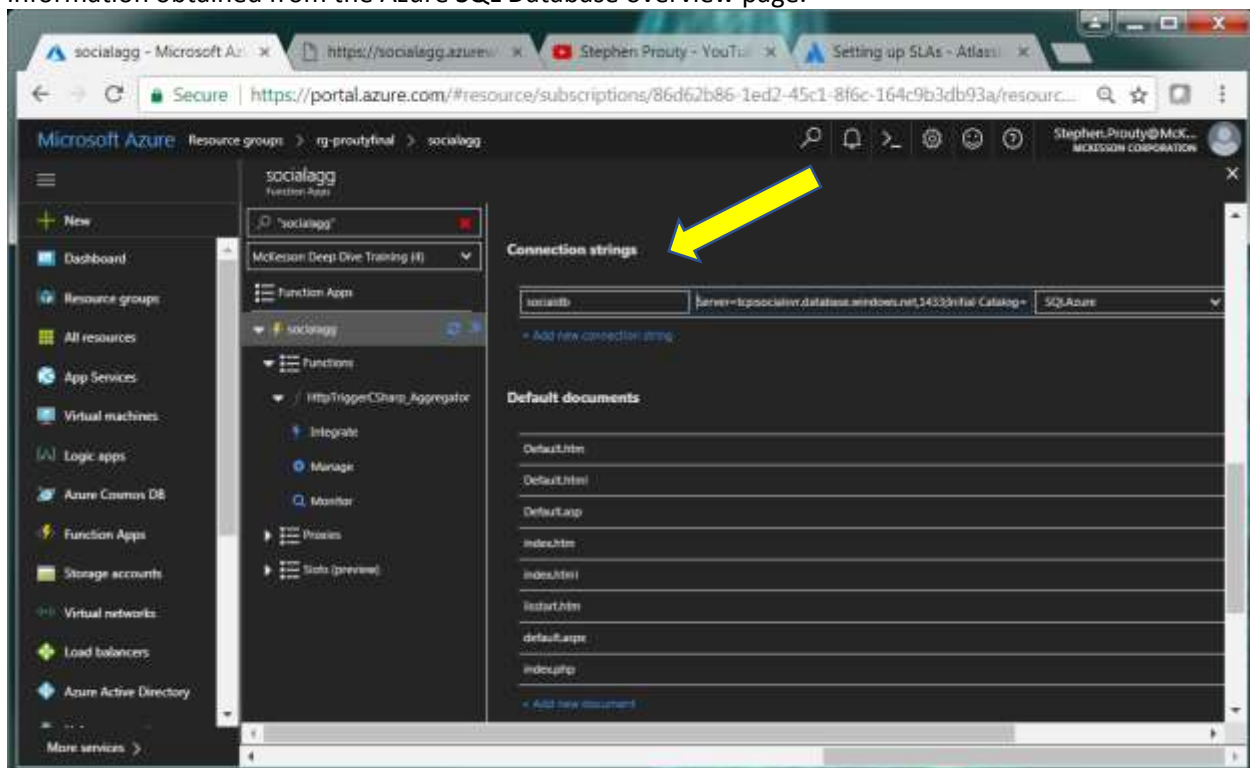
        // Close the SQL cursor
        dbCon.Close();
    }

    return req.CreateResponse(HttpStatusCode.OK, "Data Aggregation Complete");
}
```

In the above code, notice the connection string does not define the actual connection information or the username and password. This information is instead defined in the “Application settings”. By using this approach, the code does not need to make visible the credentials for the Azure SQL database. In order to enter the connection information, click “Application settings” as shown below.

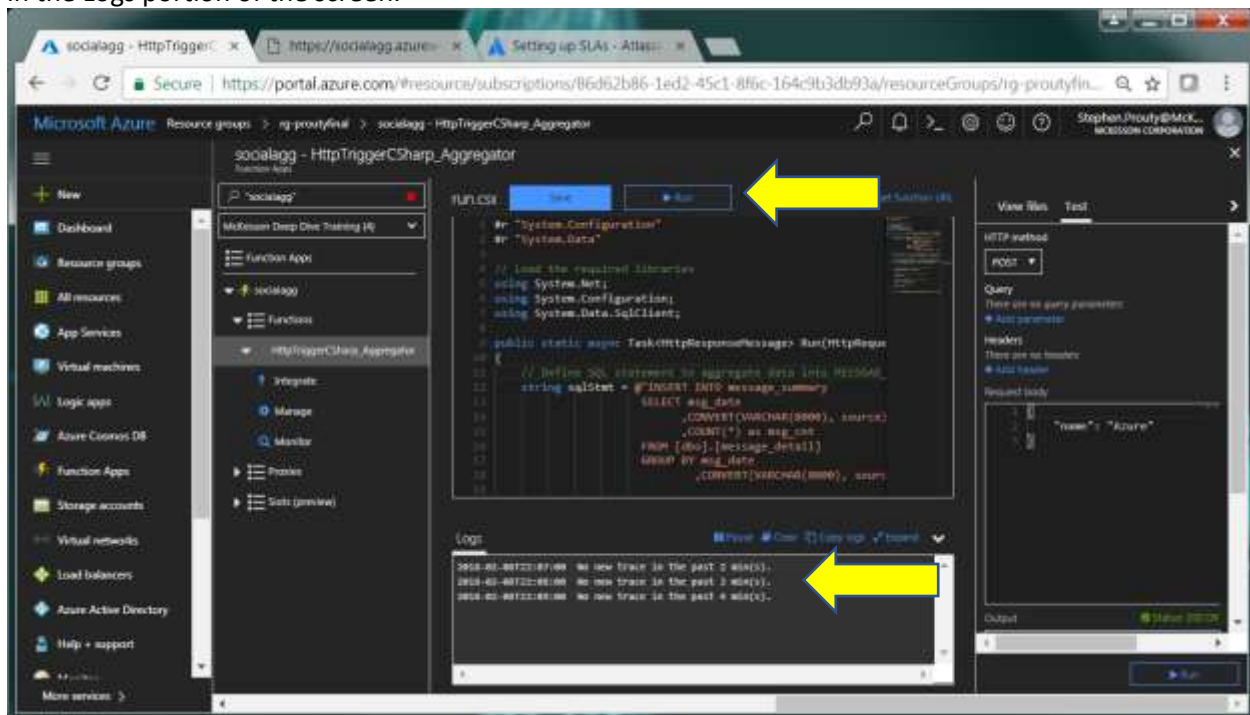


Scroll to the “Connection strings” portion and enter the connection name and the connection string information obtained from the Azure SQL Database overview page.



At this point the Function App is completed and ready to be included in the Logic App in Step 5 .

The Function App can be tested by using the “Run” button. If there are any errors they will be displayed in the Logs portion of the screen.

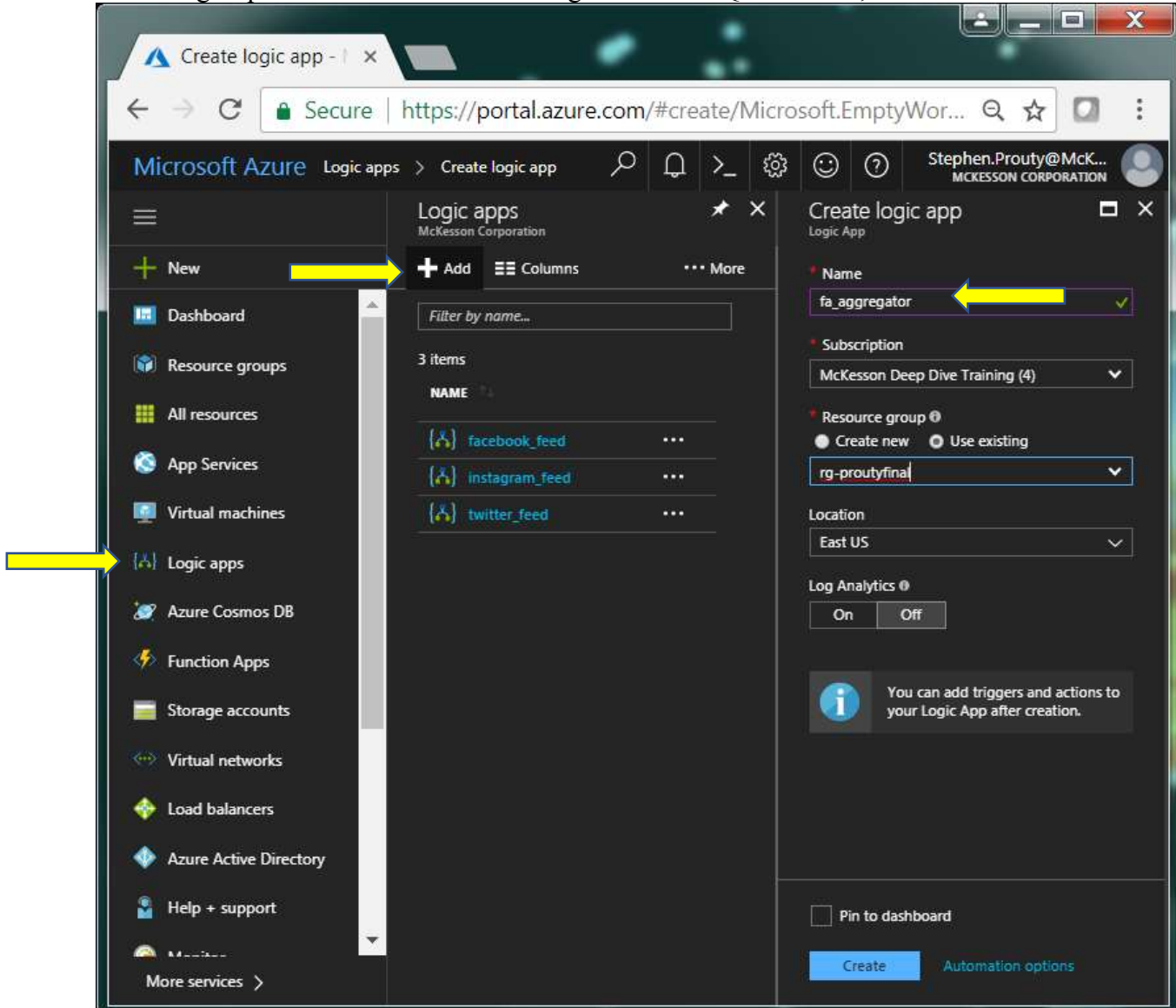




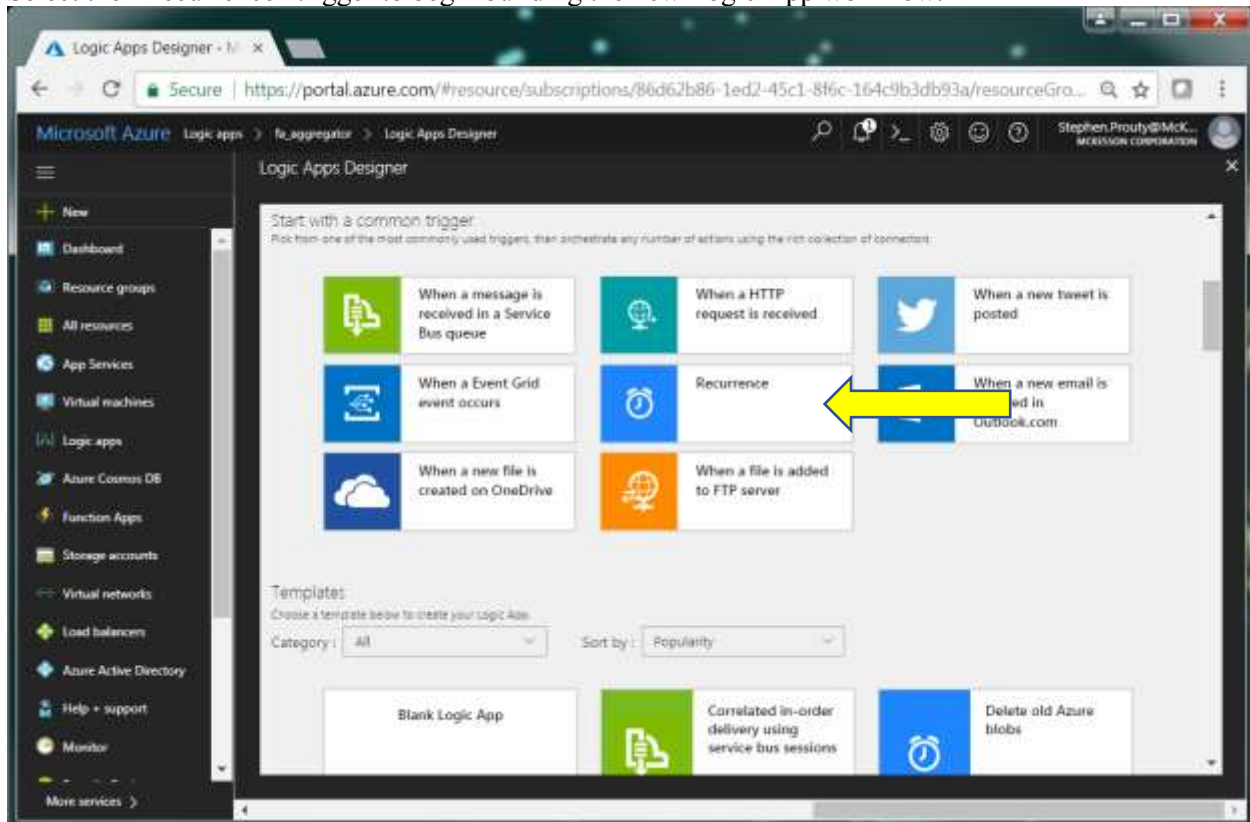
### Project Step 4 – Utilize Logic App to Execute the Function App

Next, create another Logic App that will utilize the “Recurrence” trigger to daily execute the new aggregation Function App and send a notification of completion via email.

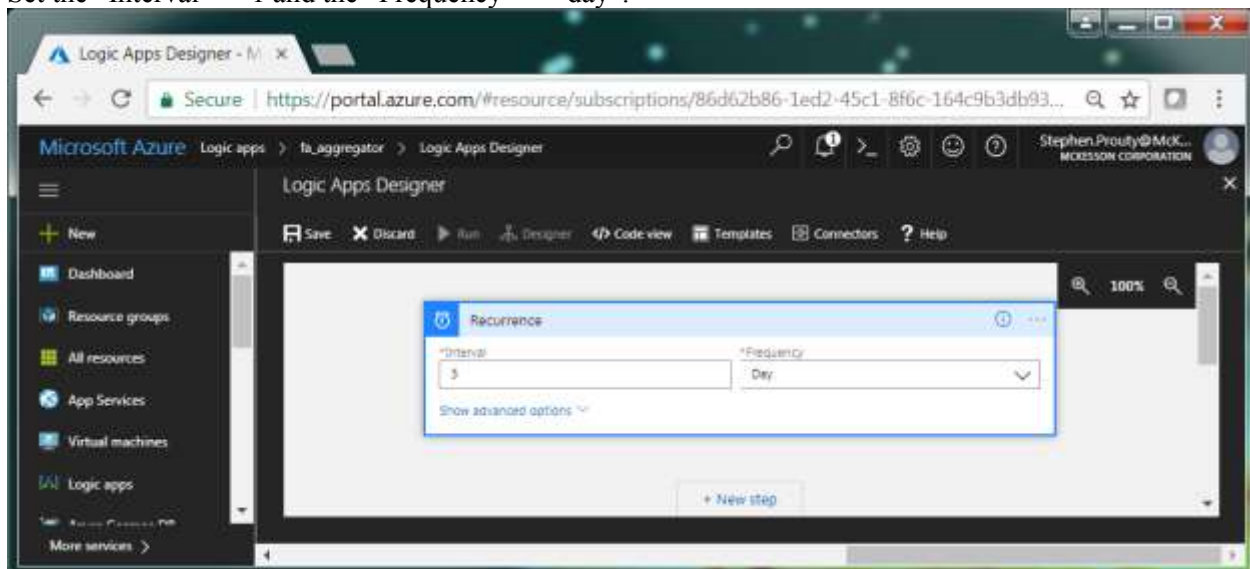
Login to the Azure Portal and add a new Logic App called “fa\_aggregator”. Use the same resource group that was used when creating the Azure SQL database, and then click “Create”.



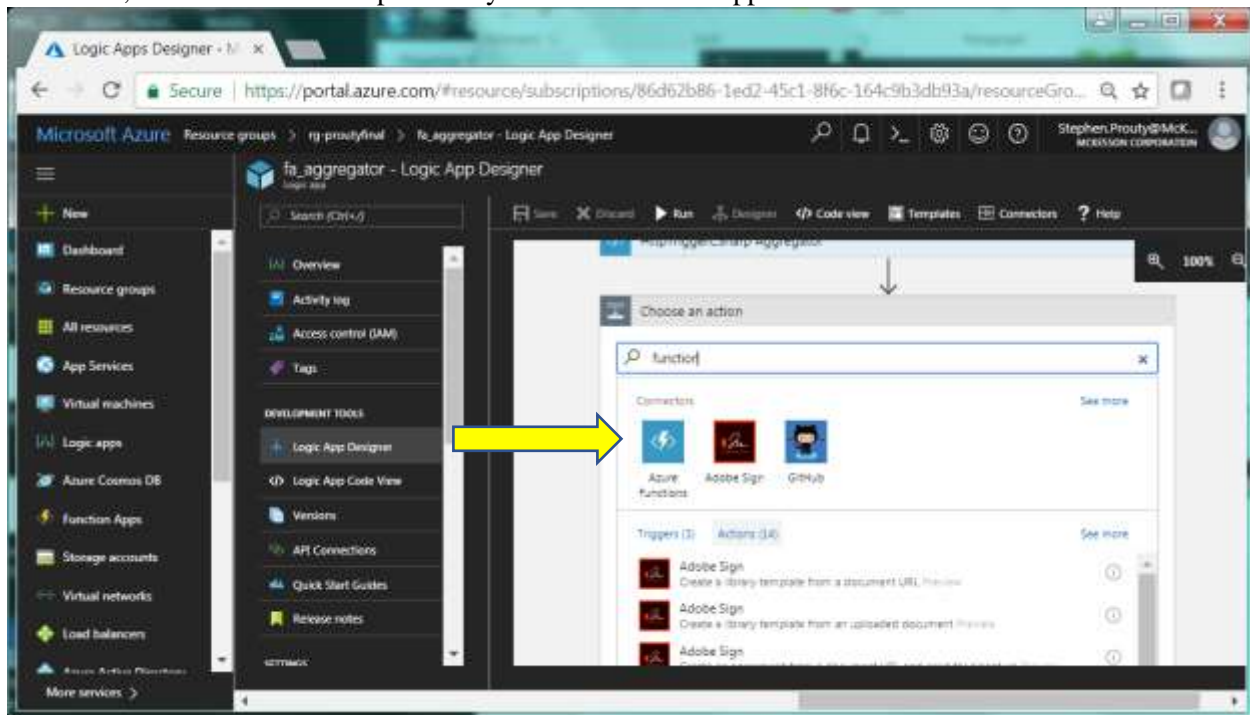
Select the “Recurrence” trigger to begin building the new Logic App workflow.



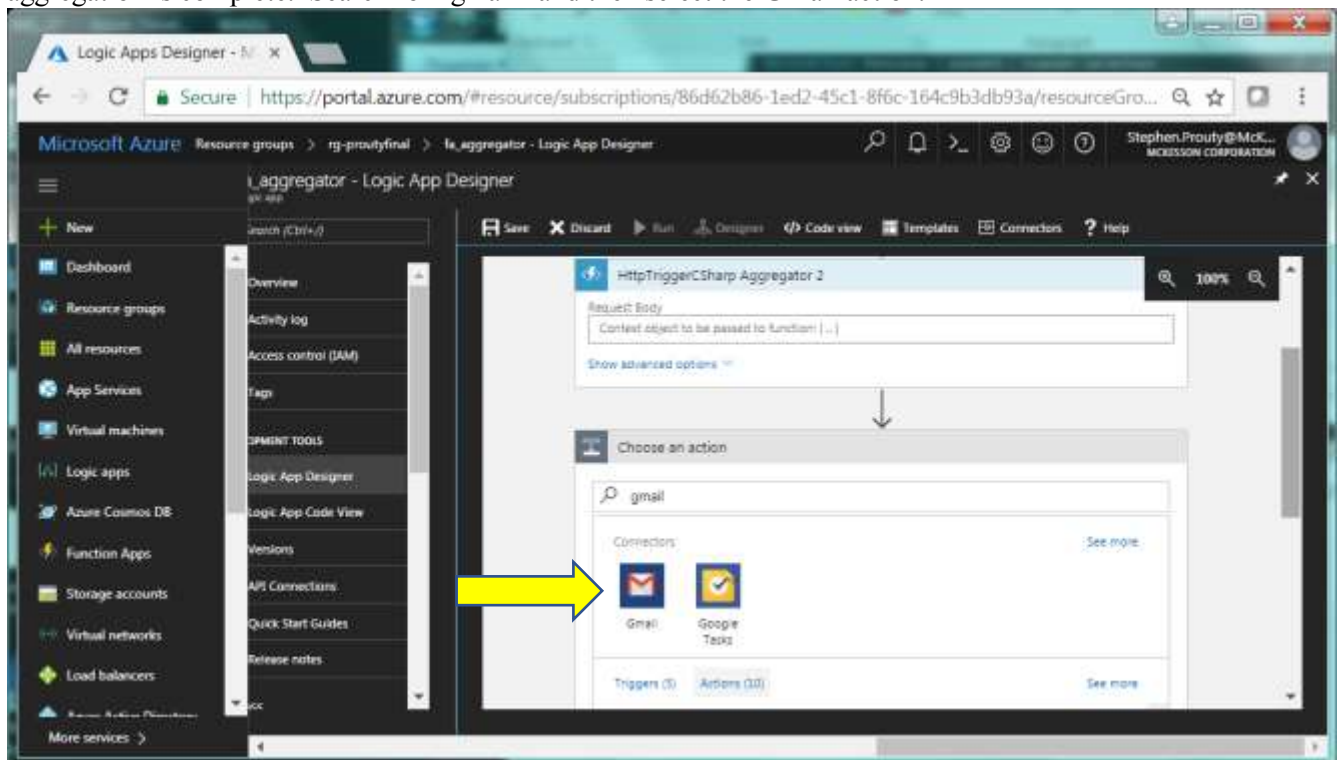
Set the “Interval” = 1 and the “Frequency” = “day”.



Next, add a workflow step that calls the aggregator Function App. Search for “Function”, select Azure Functions, and then choose the previously created Function App and C# function.

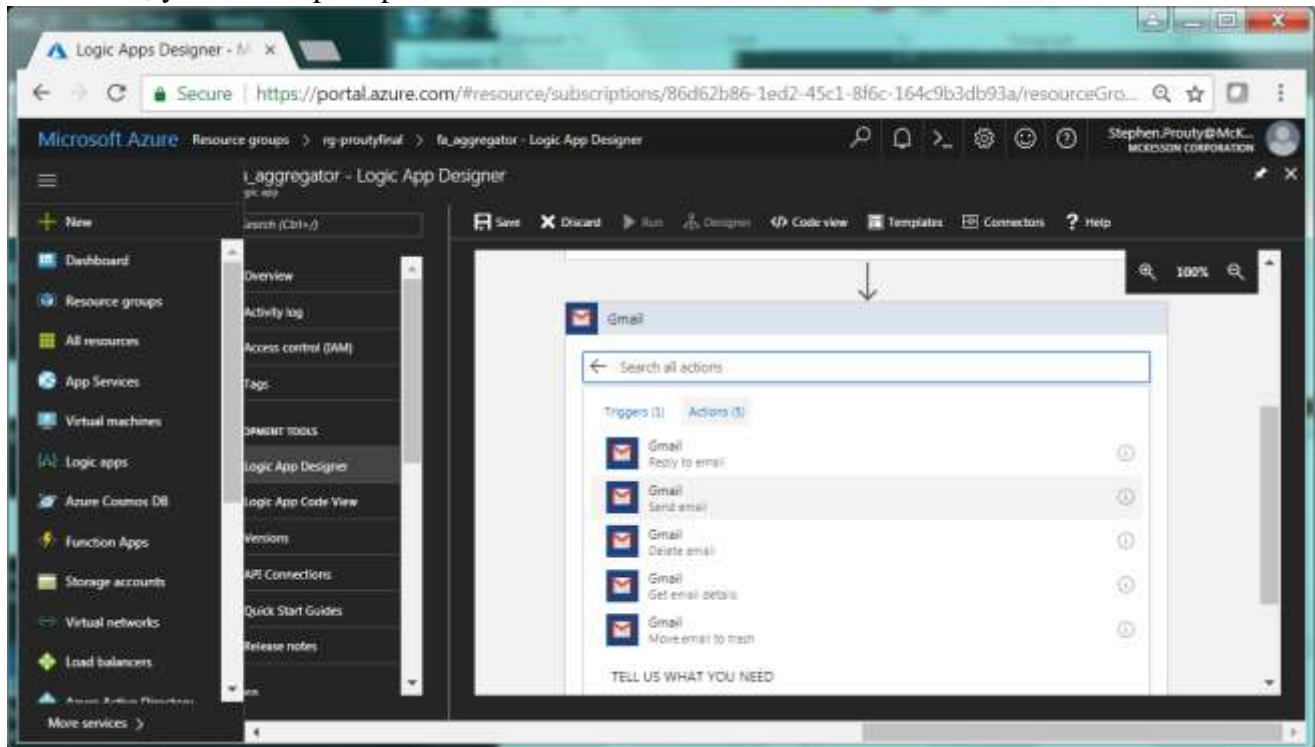


Finally, add one more workflow step to send a GMail connector to send a notification stating the aggregation is complete. Search for “gmail” and then select the GMail action.

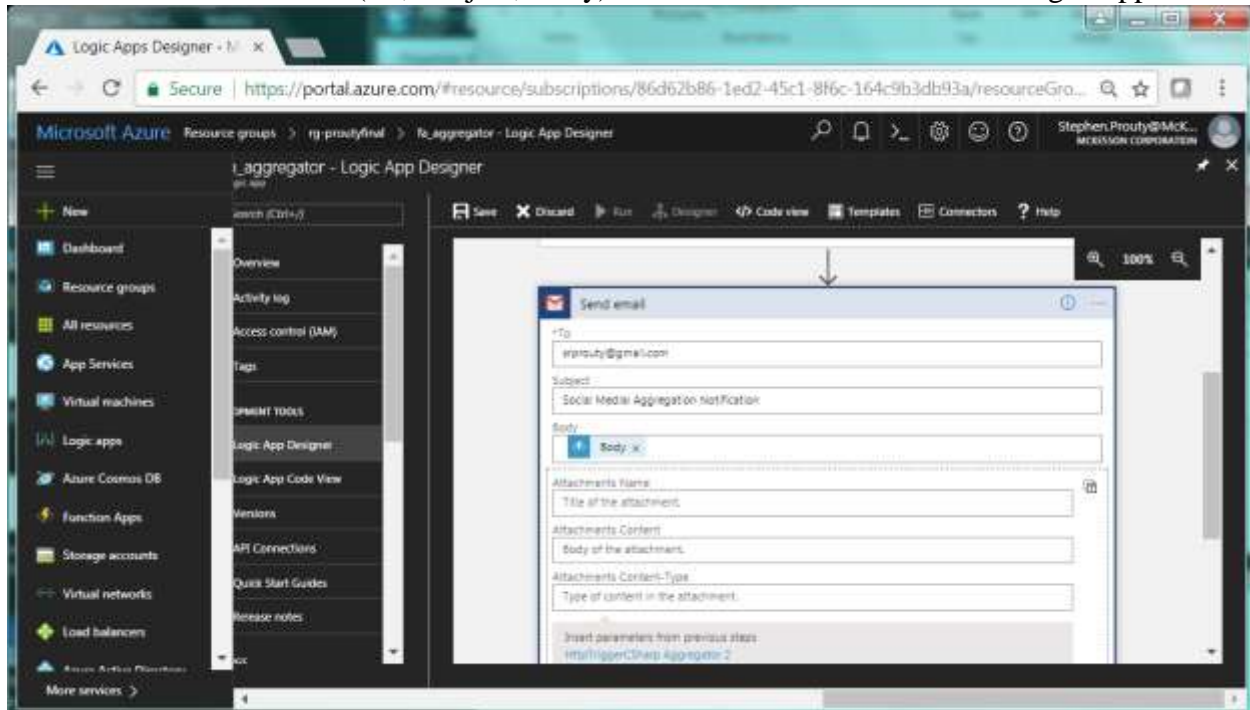




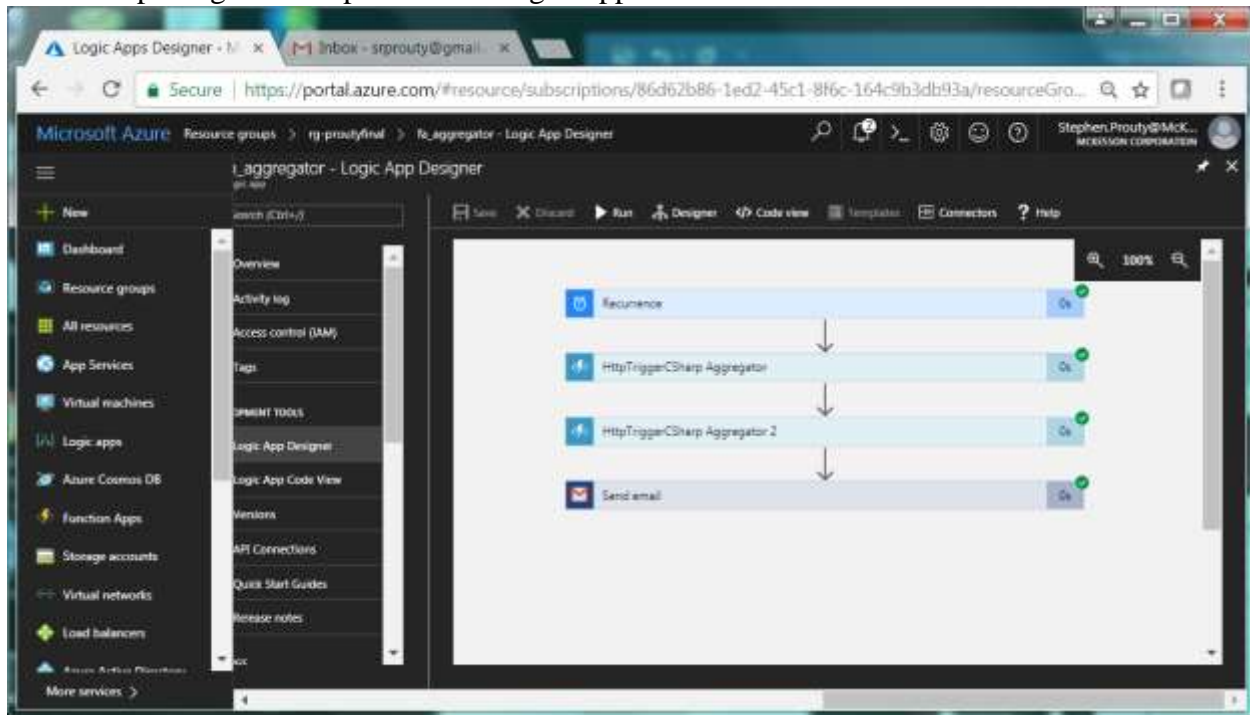
After selecting the GMail icon, choose the “send email” action. Like the social media connectors, you will be prompted to enter authentication credentials for GMail.



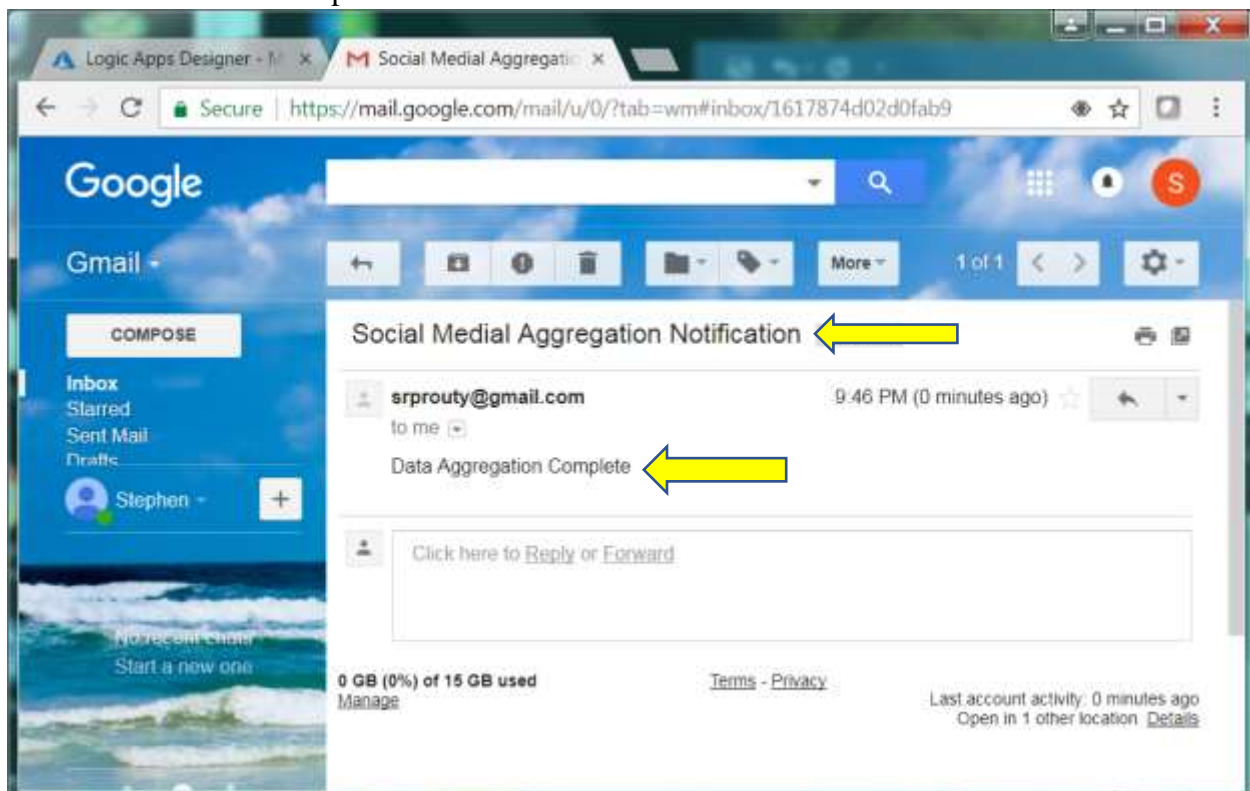
Add the email information (To, Subject, Body) to the action and then save the Logic App.



After completing these steps the final Logic App will look as follows.



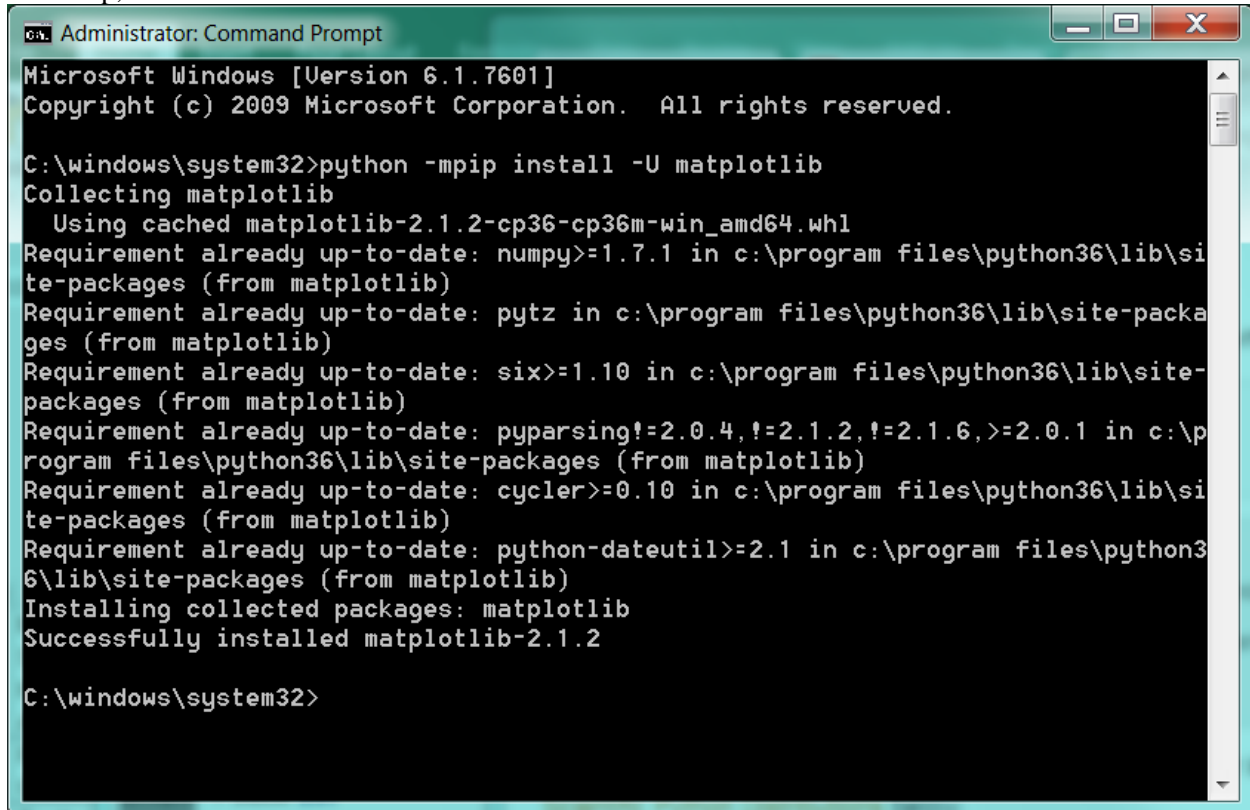
Running the Logic App updates the MESSAGE\_SUMMARY table and sends the following email notification of completion.



## Project Step 5 – Generate Data Graph via Python

The final step for this project will generate a bar chart displaying the message counts located in the MESSAGE\_SUMMARY table. In order to accomplish this task, a Python client script with the Matplotlib module will be used.

First step, install the Matplotlib module.



```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\windows\system32>python -mpip install -U matplotlib
Collecting matplotlib
  Using cached matplotlib-2.1.2-cp36-cp36m-win_amd64.whl
Requirement already up-to-date: numpy>=1.7.1 in c:\program files\python36\lib\site-packages (from matplotlib)
Requirement already up-to-date: pytz in c:\program files\python36\lib\site-packages (from matplotlib)
Requirement already up-to-date: six>=1.10 in c:\program files\python36\lib\site-packages (from matplotlib)
Requirement already up-to-date: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\program files\python36\lib\site-packages (from matplotlib)
Requirement already up-to-date: cyclur>=0.10 in c:\program files\python36\lib\site-packages (from matplotlib)
Requirement already up-to-date: python-dateutil>=2.1 in c:\program files\python36\lib\site-packages (from matplotlib)
Installing collected packages: matplotlib
Successfully installed matplotlib-2.1.2

C:\windows\system32>
```

Next open Microsoft Visual Studio Code and create a new python file with the following content. This python script will query the MESSAGE\_SUMMARY table in the Azure SQL database and use that information to plot a bar chart for the date specified. Be sure to replace the following in the python script:

<DATE> – The date you wish to create the bar chart for. Format “YYYY-MM-DD”

<SQLSERVER> - Azure SQL Server name

<SQLDB> - Azure SQL DB name

<USERNAME> - The Azure SQL DB username

<PASSWORD> - The Azure SQL DB password

```
import pyodbc
import numpy as np
import matplotlib.pyplot as plt

inputDate = "2018-02-09" # Date to select from MESSAGE_SUMMARY table

fig, ax = plt.subplots() # Define new Plot axes
ind = np.arange(1) # the x locations for the groups
width = .33 # the width of the bars
```

```

# Define bar chart variables to be assigned in SQL for loop
instagram_bar = None r'SERVER=<SQLSERVER>;'
    r'DATABASE=<SQLDB>;'
    r'UID=<USERNAME>;'
    r'PWD=<PASSWORD>;')

facebook_bar = None
twitter_bar = None

# Connect to the Azure SQL database
dbConn = pyodbc.connect(
    r'DRIVER={ODBC Driver 13 for SQL Server};')

# Open the SQL cursor and fetch the records into the appropriate bar chart variable
cur = dbConn.cursor()
for row in cur.execute("SELECT summary_date, source, msg_count "
    "FROM message_summary "
    "WHERE CAST(summary_date as DATE) = ?", inputDate):
    msgSource = row[1]
    msgCnt = row[2]
    print(msgSource + ":" + str(msgCnt))
    if msgSource == "Instagram":
        instagram_bar = ax.bar(ind, msgCnt, width, color='r')
    elif msgSource == "Facebook":
        facebook_bar = ax.bar(ind + width, msgCnt, width, color='b')
    elif msgSource == "Twitter":
        twitter_bar = ax.bar(ind + 2*width, msgCnt, width, color='g')
    else:
        print("Invalid Source Found" + msgSource)

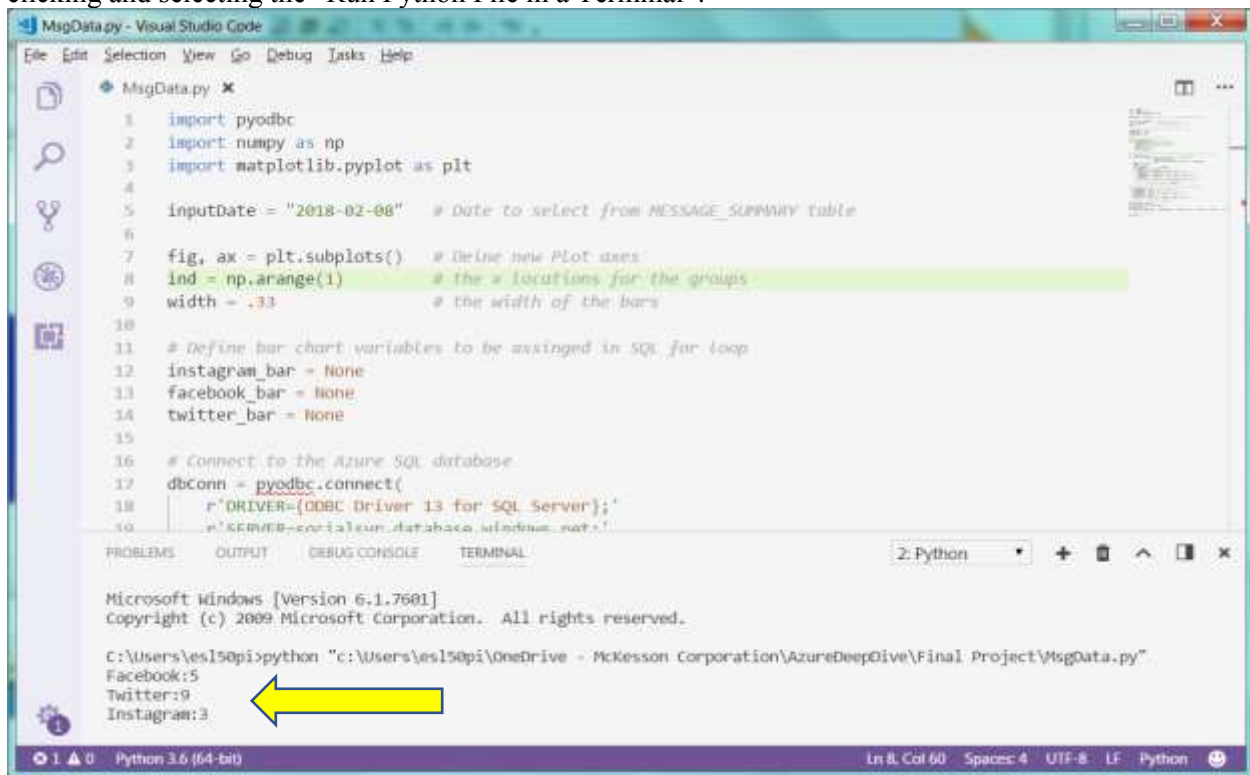
# Set the Bar Chart value to 0 if no records are returned from SQL DB.
if instagram_bar is None:
    instagram_bar = ax.bar(ind, 0, width, color='r')
if facebook_bar is None:
    facebook_bar = ax.bar(ind + width, 0, width, color='b')
if twitter_bar is None:
    twitter_bar = ax.bar(ind + 2*width, 0, width, color='g')

# Add Bar Chart labels, title, and legend
ax.set_ylabel('Messages')
ax.set_title('Social Media Usage')
ax.set_xticks([])
ax.legend((instagram_bar[0], facebook_bar[0], twitter_bar[0]), ('Instagram', 'Facebook', 'Twitter'))

# Plot the Bar Chart
plt.show()

```

After creating the new python script, save the file as a Python type and then execute the script by right clicking and selecting the “Run Python File in a Terminal”.



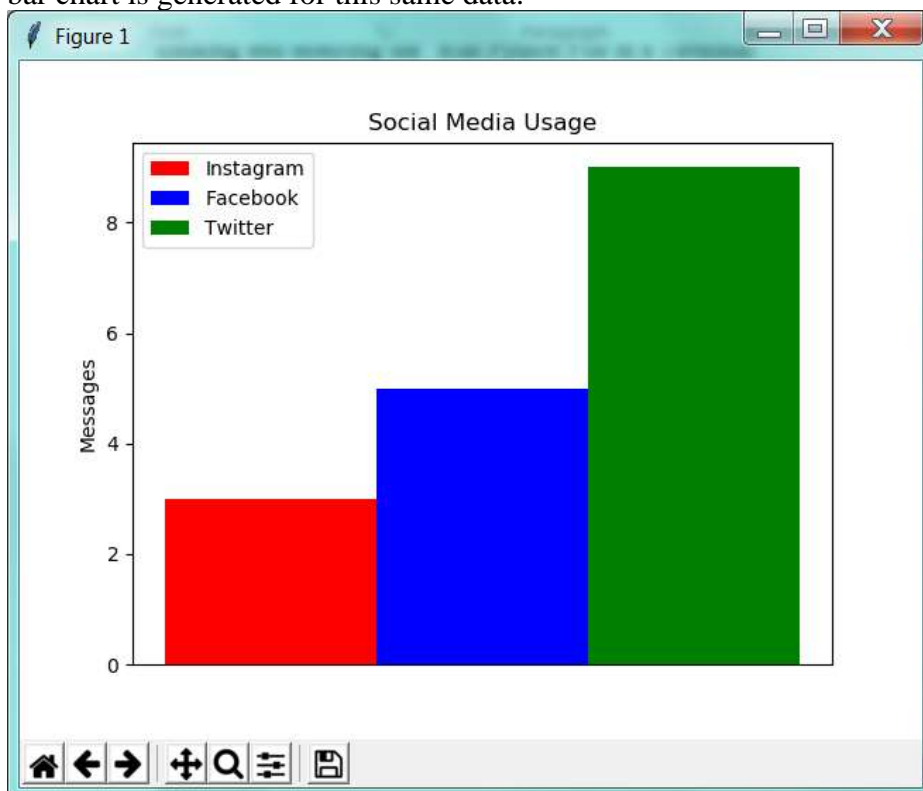
```
1 import pyodbc
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 InputDate = "2018-02-08" # Date to select from MESSAGE_SUMMARY table
6
7 fig, ax = plt.subplots() # Derive new Plot axes
8 ind = np.arange(1) # the x locations for the groups
9 width = .33 # the width of the bars
10
11 # Define bar chart variables to be assigned in SQL for loop
12 instagram_bar = None
13 facebook_bar = None
14 twitter_bar = None
15
16 # Connect to the Azure SQL database
17 dbconn = pyodbc.connect(
18     r'DRIVER={ODBC Driver 13 for SQL Server};'
19     r'SERVER=cs150pi.azurewebsites.net;'
20 )
21
22 # Execute SQL query
23 sql = "SELECT media, count(*) as count FROM MESSAGE_SUMMARY WHERE InputDate = '%s'" % InputDate
24 cursor = dbconn.cursor()
25 cursor.execute(sql)
26
27 # Print counts
28 for media in ['Instagram', 'Facebook', 'Twitter']:
29     count = 0
30     for row in cursor.fetchall():
31         if row[0] == media:
32             count += row[1]
33     print(media + ': ' + str(count))
34
35 # Create bar chart
36 for media in ['Instagram', 'Facebook', 'Twitter']:
37     count = 0
38     for row in cursor.fetchall():
39         if row[0] == media:
40             count += row[1]
41     bar = plt.bar(ind, count, width)
42     if media == 'Instagram':
43         bar.fillcolor = 'red'
44     elif media == 'Facebook':
45         bar.fillcolor = 'blue'
46     elif media == 'Twitter':
47         bar.fillcolor = 'green'
48     ind += 1
49
50 plt.show()
```

Microsoft Windows [Version 6.1.7601]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\es150pi>python "c:\Users\es150pi\OneDrive - McKesson Corporation\AzureDeepOlive\Final Project\MsgData.py"

Facebook:5  
Twitter:9  
Instagram:3

In the “Terminal” window above the counts are displayed for each media. In addition, the below bar chart is generated for this same data.



## Summary

Logic Apps is a very flexible architecture capable of delivering complex workflows with minimal coding required. The built in workflow features allow for parallel processing and conditional logic such that the applications can be customized to meet the needs of the project being implemented. Logic Apps currently supports ~200 different connection APIs for both Microsoft resources (blob storage, SQL Server, event grids, etc) as well as interfaces to external resources (GMail, Twitter, Facebook, HipChat, etc).

In this project, some basic functionality was displayed using Logic Apps to integrate with three mainline social media applications (Facebook, Twitter, and Instagram). The basic integration with each media was relatively simple. However, I found that the documentation around each interface was lacking. Each media provided numerous data attributes that could be captured. However, few of them used names or were adequately described. As a result, I spent a significant amount of time performing trial and error exercises to capture the desired data. In addition, I found that the Instagram API did not expose an attribute for the date/time of the posted message. This was a critical piece of information that was needed to properly aggregate the data into the summary database table. As a replacement, the program utilized a basic Logic App date function ( @utcnow() ) to insert the current date/time.

Given more time, there are many improvements that could be implemented that would make this application more beneficial to the end user. A few of these improvements are listed below.

- Capture images from the Social Media applications.
- In addition to the graph of messages usage, display the actual message text for a given day.
- Produce historical multi-day bar charts instead of a single day.
- Create a web front end instead of a Python script.
- Integrate with more media connectors (HipChat, GMail, SLACK, etc)

## References

### YouTube Links:

2 Min: <https://youtu.be/Nq21domeXjc>

15 Min: <https://youtu.be/Hf0IBo7nDko>

### GitHub Repository:

<https://github.com/scmprouty/AzureDeepFinal>