**[Research Title]**

**Group Name:**

|  |  |
| --- | --- |
| Semester | Spring 2022 |
| Course Code | CBD 3384 |
| Section | 2 |
| Project Title | TWEET SENTIMENT ANALYSIS (TSA) using Azure Logic App and Function. |
| Group Name | Group F |
| Student names/Student IDs | Piyush Thakur (C0814266)  Parwinderjit Kaur (C0813818)  Vedank Shaileshkumar Solanki (C0820175)  Sailesh Paudel (C0813856)  Rahul Dankal (C0818803) |
| Faculty Supervisor | William Pourmajidi |

**Submission date: *[Insert submission date here]***

Contents

[Abstract 3](#_Toc110876301)

[A summary of the entire project and its results 3](#_Toc110876302)

[Introduction 4](#_Toc110876303)

[Methods 7](#_Toc110876304)

[Azure Cognitive Services Creation 7](#_Toc110876305)

[Azure Function App Creation 11](#_Toc110876306)

[Function App Configuration 13](#_Toc110876307)

[Azure Logic App Creation 16](#_Toc110876308)

[Azure Logic App Configuration 17](#_Toc110876309)

[Results 28](#_Toc110876310)

[Conclusions and Future Work 29](#_Toc110876311)

[References 29](#_Toc110876312)

# **Abstract**

## A summary of the entire project and its results

The main objective of this capstone project is to develop an easy and efficient way to perform Natural Language Processing to analyze Twitter Activity and determine the sentiment of Tweets that contain a particular hashtag. Using the computational power of Azure Functions along with Azure Cognitive Services as well as Azure Logic App to connect to Twitter and Office 365 API connections, our project aims to detect the sentiment of tweets and trigger an email informing the user with the sentiment as well as the author of the tweet.

The Logic App workflow can be configured to run at set intervals or manually when required, to check for a Hashtag using the Twitter API connection. The contents of the tweet are then fetched and passed into the Azure Cognitive which performs Sentiment Analysis on the body of the tweet. The Cognitive Service then returns a Confidence Score which comprises of a decimal value which ranges between 0 and 1 for all the three sentiments: Positive, Neutral and Negative.

The Azure Function App consumes these values and decides what the overall sentiment of the tweet is. The output of the Function App (The final Sentiment Value) is then stored which will later be consumed by an API connection from Office 365. This API connection contains the credentials to the outlook account and will sign-in the user at run time and trigger an email to notify the end user about the sentiment Analysis. The API connection receives information from the Twitter API about the author of the tweet and its body, fetches the Sentiment score from the function app and create an email from this dynamic information for each tweet.

The email is then triggered from the Office 365 API and will be delivered to all the recipients that are defined at the time of creation of the workflow.

It takes time to create this entire setup and configure it on Azure, hence we have also included an Automation technique by using ARM templates. This will allow potential users to deploy the entire Application on Azure without having to go through the effort of clicking on multiple UI components on the Azure Portal. The user can just use the template that we have uploaded on GitHub and run it on the *Deploy Custom Template* editor on Azure. This technique can also be beneficial if a user wants to deploy the same setup in multiple azure subscriptions and this helps the automate this entire deployment of the workflow which includes the creation and configuration of all the Azure resources associated with this project.

# **Introduction**

Sentiment analysis which can also be referred to as emotional AI or opinion mining, can essentially be defined as the process of analysis of pieces of writing, which helps us to determine the tone that they carry from an emotional perspective. In simple words, it helps us to analyze the school of thought of the author towards a particular topic. With the power of sentiment analysis, companies can:

* Have a better grasp to understand how their customers feel.

* Mold their business to become effective to feedback provided by customers.

* Observe and react efficiently to preserve brand reputation.

* Help their business to become more receptive and empathetic to customer feedback.

* Increase cognizance to enhance their products and services.

It is very hard to imagine how tedious a job it can be to browse the Internet, locate relevant textual information, read and analyze them manually to understand the sentiment of the authors. While it is doable, it is extremely inefficient and can take a lot of time especially in the era where Natural Language Processing technologies, Machine Learning techniques, Artificial Intelligence technologies are becoming more and more efficient and reliable every day. If this process can be automated, it can help save valuable resources as well as time which can be put to better use in improving the audience's sentiment for the product/information.

When it comes to the Internet, what is a better place than Twitter to understand what the consumers are talking about when it comes to brands, their products and services and the experience that the users have with these products. There are innumerable posts and threads which are boiling with comments and opinions of users which can be helpful for brands to gauge the public perception.

What if there is a sentiment analysis tool that can help you to monitor such information, analyze it in real time and notify you all at the same time. With access to such a tool, a company can react quickly to a problem before it gets escalated and becomes a problem for the reputation of

their brand. For that matter, it can even be used by PR specialists to get intimated whenever there is a pessimistic piece of information regarding their clients and help them to react in a timely manner.

This is exactly what we have tried to create in this project. We have created an Azure Logic App application which contains an entire workflow comprising of multiple Azure Services that work in conjunction to provide Sentiment Analysis of tweets. The application can be configured to poll twitter for a particular hashtag at regular intervals or manually when desired. The Application Connects to the Twitter API and performs a GET request and fetches tweets that contain the particular hashtag.

The tweets are then passed on to an Azure Cognitive Service that performs Sentiment Analysis on these tweets and returns the Confidence Score which is a decimal value between 0 and 1 for all the three sentiments: Positive, Neutral and Negative. These Confidence scores are then fed as an input to an Azure Function App which decides the overall sentiment of the tweet and declares it to be either Positive, Negative or Neutral.

The next Component in the workflow is an API connection that we are using to connect the Logic App to a SaaS service in the form of Office 365. This API connection contains information that is needed to connect to the Office 365 SaaS service. Once the function inside the Function App is triggered and the sentiment analysis is completed, the Logic App will provide information like the tweet author’s name, tweet’s body content along with sentiment of the tweet from the Function App to the Office 365 API connection.

The Office 365 API connection then shoots an email to the configured recipient along with all this information and thus completing this workflow by intimating the recipient of any tweets that are being posted along with their sentiment analysis.

Detailed below is some information regarding all the different components that we have used in this project.

**Azure Cognitive Services:** Azure Cognitive Services is a special product from Microsoft that help the user incorporate Cognitive intelligence into their applications. These services can be useful for automation and integration between several Azure resources. It provides Natural Language Processing capabilities for understanding and analyzing textual information. This service can be used to create intelligent applications using REST API’s, the web based Language studio provided by Azure or using the dedicated client libraries. We are using the sentiment analysis feature of this product to gain granular information regarding the opinions that can be devised from words that appear in tweets.

**Azure Functions:** Azure Function is a serverless compute service provided by Microsoft that allows users to run a piece of code that runs when an event is triggered. This Event can be a Timer trigger, an HTTP Trigger, Queue Trigger, etc. Unlike Azure AppServices, we do not need to

have dedicated Server’s allocated to our applications. The advantage of being serverless means that the user only gets billed for each execution of the function app and not for the hosting of

the application. The Azure platform is intelligent enough to deallocate any resources during the time that the application does not run. This was another important factor in us choosing Function apps instead of Azure AppServices.

**Azure Logic Apps:** Azure Logic Apps is a solution provided by Microsoft to develop and run automated workflows that can be integrated with services, applications, systems and data. Logic Apps is a part of Azure Integration Services that simplifies the way that we can connect legacy devices as well as Cloud and on-premises environments with each other. In our project it helps us to monitor the tweet’s activity and analyze their sentiment. We use triggers and actions that run inside the Logic App and can include conditional statements, iterative statements, and can call other Azure services like API apps and Azure Functions as needed for the workflow of the Logic App.

**API Connection:** API connections for Azure services like Office 365 or to communicate with third part services like twitter work as connectors. These connectors work like a proxy around an API that is used to communicate with the Logic App. These connectors provide access to various operations that can be used to complete a Logic App workflow and these operations can be in the form of a trigger which allow the workflow’s different components to initiate the processing.

**Azure Storage Account:** Whenever a function app is created, it is necessary to link it with an Azure Storage account. This is because functions need the storage account to log function executions, manage the various triggers that are configured for the app, storage of function keys necessary for its invocation, etc. Azure functions used a host ID value which is used to store per-app correlation and its tracking information. If there is a collision between duplicate host ID values, it becomes impossible to uniquely identify the linked function app, hence it is recommended to use separate storage accounts for multiple functions. It is also recommended to create the storage account in the same Azure region as the function app to reduce latency.

# **Methods**

Below is outlined how we have created and configured all the Azure resources necessary for creating our environment. The first resource that we will need to create is Azure Cognitive Services for Language. Azure Cognitive Services is a powerful tool provided by Microsoft which comes equipped with a lot of capabilities like Key Phrase Extraction, Prebuilt Question Answering, Sentiment Analysis etc. There are a lot of other customizable features as well that require provisioning of additional services like Azure BLOB storage and Azure Cognitive Search etc. however, we will not be using them for the environment that we intend to create.

## Azure Cognitive Services Creation

While creating this resource it's better to create a resource group with a unique name and deploy this resource along with all the other resources associated with this project in the same resource group. This will help us for better management, governance, and cost analysis of the resources. While choosing the region, we have chosen to go with the East US region as it's one of the major data centers in Azure and it's the best choice as this data center allows us to deploy most if not all Azure resources, hence the latency and the cost related to the communication between different Azure resources will almost negligible as Azure does not charge for dataflow between Azure resources that are hosted in the same region.

Coming to the pricing tier for this resource, it's best to go with the free tier, as it provides us with 5000 TXT records processing per month for free. If somebody wishes to enhance and scale this project for huge amount of processing, there is a standard rising tier provided by Microsoft as well which provides us with 10 million and above text record processing at a very low price. For more information regarding the pricing details for this resource you can refer the [link](https://azure.microsoft.com/en-us/pricing/details/cognitive-services/language-service/).

However, if anyone want to try this out for testing purposes, they need to remember that it is only possible to create one azure cognitive service for language that is in the free tier. Any subsequent resource needs to be in the standard tier (This is a limitation imposed by Azure as the free tier is intended for testing purposes only).

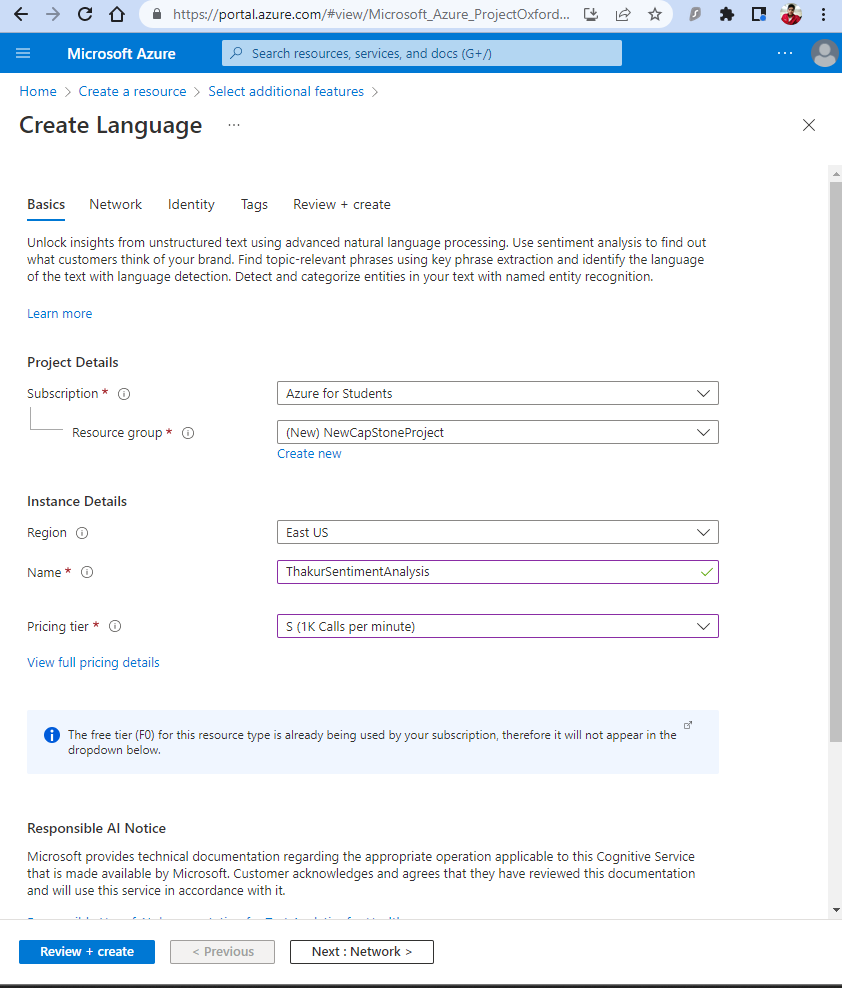


Figure 1: Azure Cognitive Service creation

Coming to the Networking blade, here we can choose how much network access should be provided to this resource. What that means is, should resources throughout the Internet, only resources in particular selected network, or no resource from any network should be allowed to access this resource.

There is a functionality provided by Microsoft called Private Endpoints, which will allow other Azure resources to privately connect (via the private Azure backbone) to this resource. In the next tab labeled as identity, Microsoft provides us the functionality to use Azure Managed Identity to grant access to this resource by a group of other resources other people etc. However, we will not be creating any such restriction for our environment.

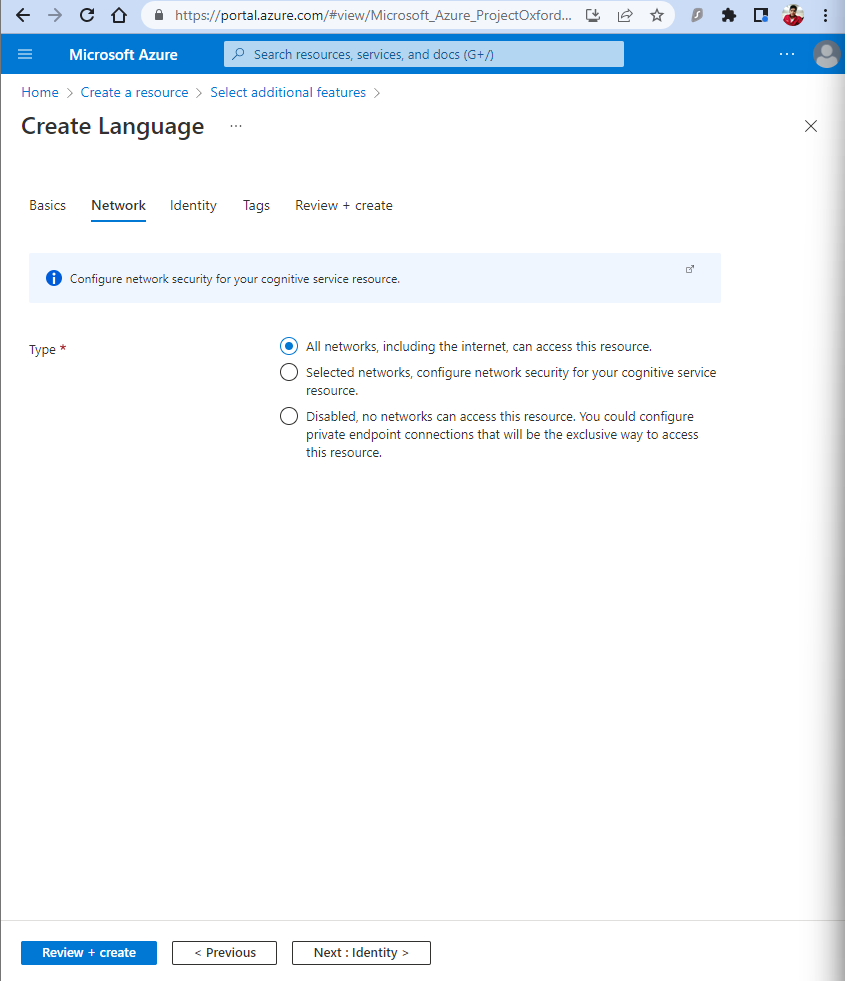


Figure 2: Networking Configuration for Cognitive Service

Navigating to the newly created resource, we will head to the keys and endpoint plate for the time and copy the values for key one and keep this safe for the time being. Along with this, we will also be copying the value of the endpoint. These values will be used later to access the API of our cognitive service and it is important to keep these keys safe, as unauthorized access to these keys can mean unauthorized access to this resource.

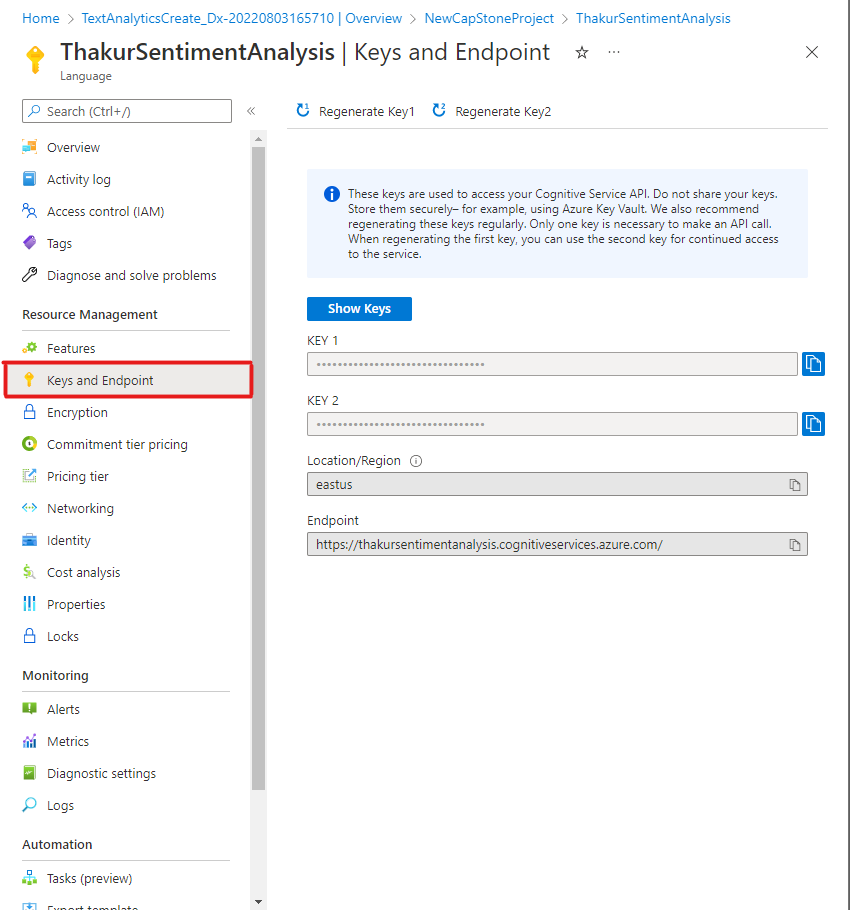


Figure 3: Keys and Endpoint Configuration

## Azure Function App Creation

We will now have to create the next important component of our environment which is an Azure function app. Azure functions is an on-demand cloud PaaS surface which provides the user with continuously updated infrastructure under the hood as well as resources required to run your application.

The biggest advantage of using an Azure Function App instead of hosting your application on an Azure VM or an Azure App Service is that the user only needs to focus on the piece of code that is intended to be ran and the user does not need to worry about the underlying infrastructure, it's updates, it's scalability nor its availability. The Azure infrastructure is smart enough to scale our function app to run on zero instances when there is no traffic or no application execution all the way to 100 instances when the application needs it.

This behavior or architecture is also referred to as a serverless architecture because the end user does not need to be concerned about how, when and where the application is running, just configure your application and its trigger (when the application should run) and let Azure functions handle the rest.

We had initially chosen to run our application on Azure App Service, which is another PaaS offering from Microsoft, however it does not provide us with a serverless architecture and we need to make sure that reconfigure all the underlying architecture (in regards of the number of instances, the size of instances, the scaling of instances) on which the application will be running. Another disadvantage of choosing Azure web apps instead of Azure functions was that the user constantly gets billed for the app service even if the application is not running.

This is because Azure app services run on dedicated instances provided by Microsoft and even if the application is in a stopped state or even if it is running and receiving no traffic, because the underlying server instance it's still allocated to the application the user continues to get charged for it regardless. This was one of the biggest reasons that drove us towards using function app instead.

As we can see below, while creating the function app we choose the app Service plan to be a consumption (serverless ) plan .

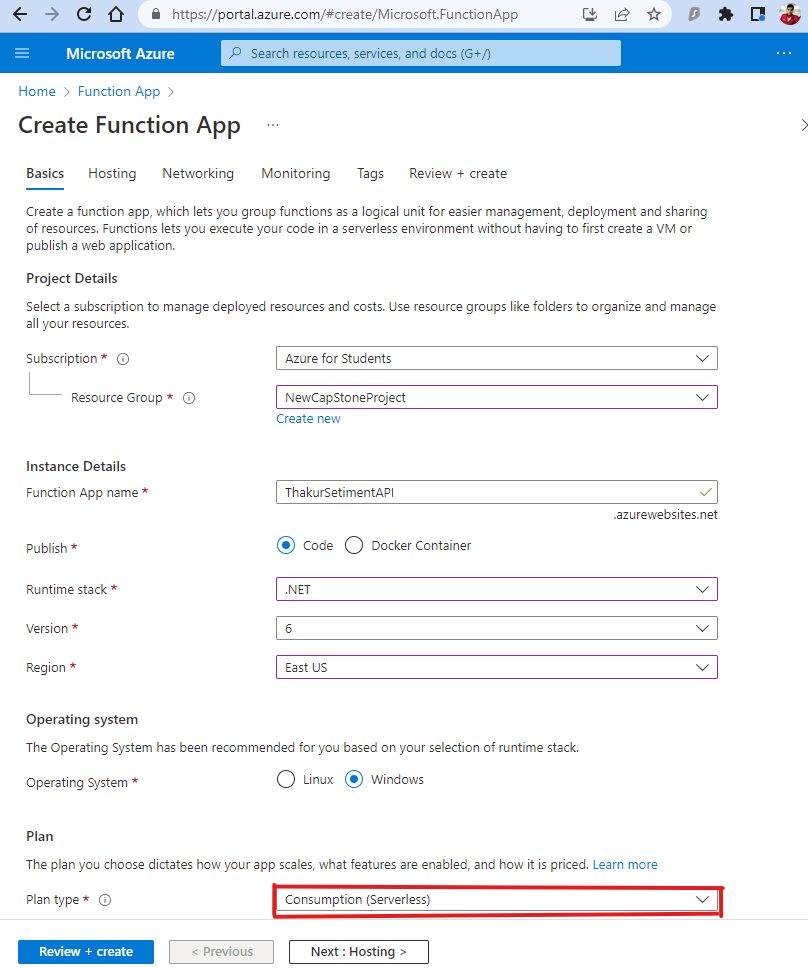


Figure 4: Function App Creation and Configuration

Another thing to note here is that when a function app is created a storage account is also created automatically along with it. This storage account stores the necessary keys that are required for the invocation of the function app. We will not be delving into the networking blade while creating the function app as it is again required in scenarios where we want to restrict access to the function app from public Internet and want only resources in our own private network to be able to connect to as well as invoke the function app.

## Function App Configuration

Once the function app has been created, we will proceed with creating a function inside this function app which will run whenever it receives an HTTP request based on data in the body or query string.

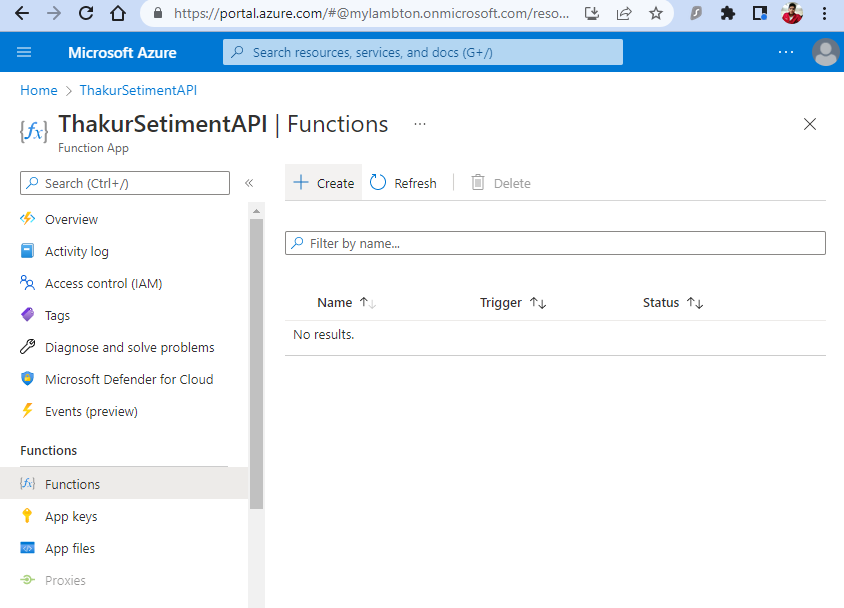


Figure 5: Creation of function inside Function App

When creating the function, we will need choose a trigger which will describe the event that causes the function to be invoked. We will choose the HTTP Trigger Function option when creating the Function.

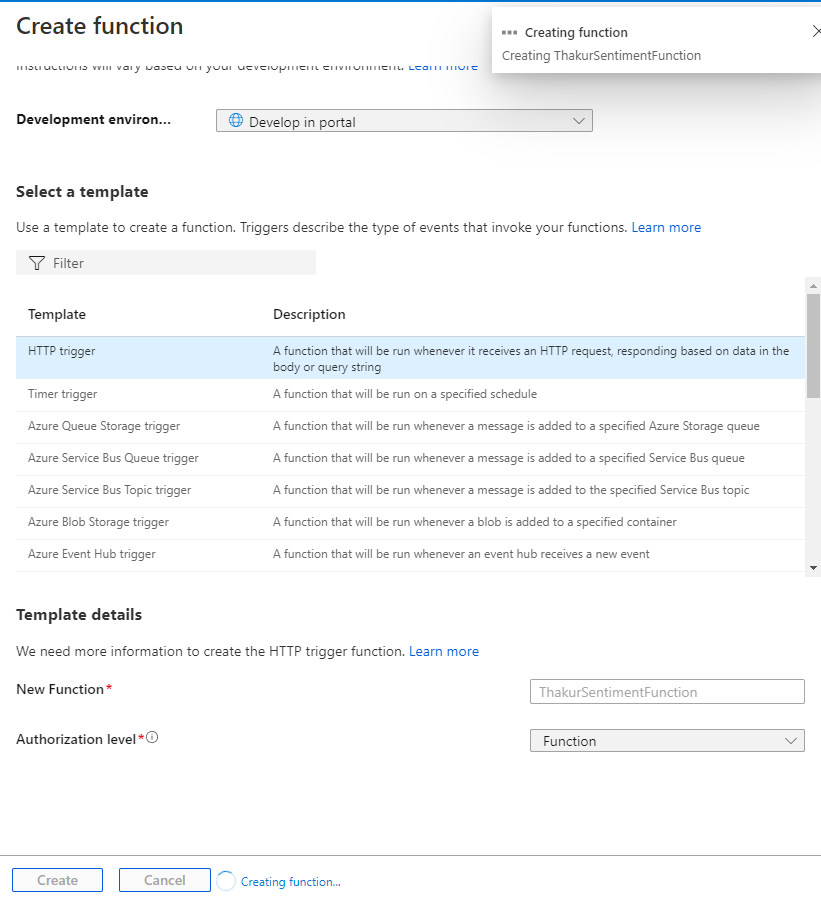


Figure 6: Choosing the Function Trigger

Going to the Code + Test blade of the function, this is where we put the code of the function. Here the Confidence score gets passed into the function from the Cognitive Service and once that is done the Function App decides the overall sentiment of the tweet and declares it to be either Positive, Negative or Neutral.

If we want to test the working of the function, Azure provides us a functionality by choosing the Test / Run button where we can manually enter a value in the body input box and choose to run it. Depending on the value we get the value positive or negative for the sentiment of the tweet.

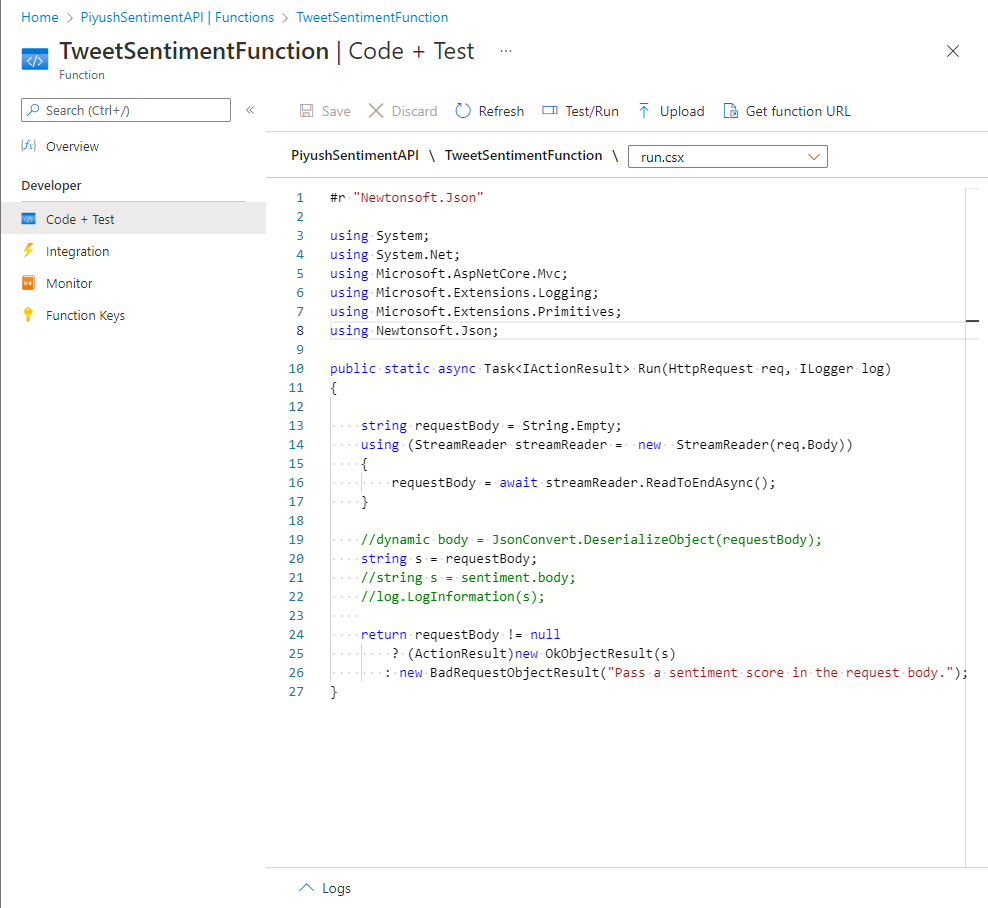


Figure 7: Code + Test blade of the Function

The code of the function app is available [here](https://github.com/PiyushThakur266/PiyushRepo/blob/main/CapStone%20FunctionApp%20Code) which can be copied directly into the function of the function app and it will be ready to run.

## Azure Logic App Creation

Now for the main component, we have created a Logic App. This logic app will help us to integrate with Azure functions, our Twitter API Connection, the API for Office 365 and the Cognitive Services. Just like the Azure Function App, our Azure logic app can also run on the serverless platform (consumption plan).

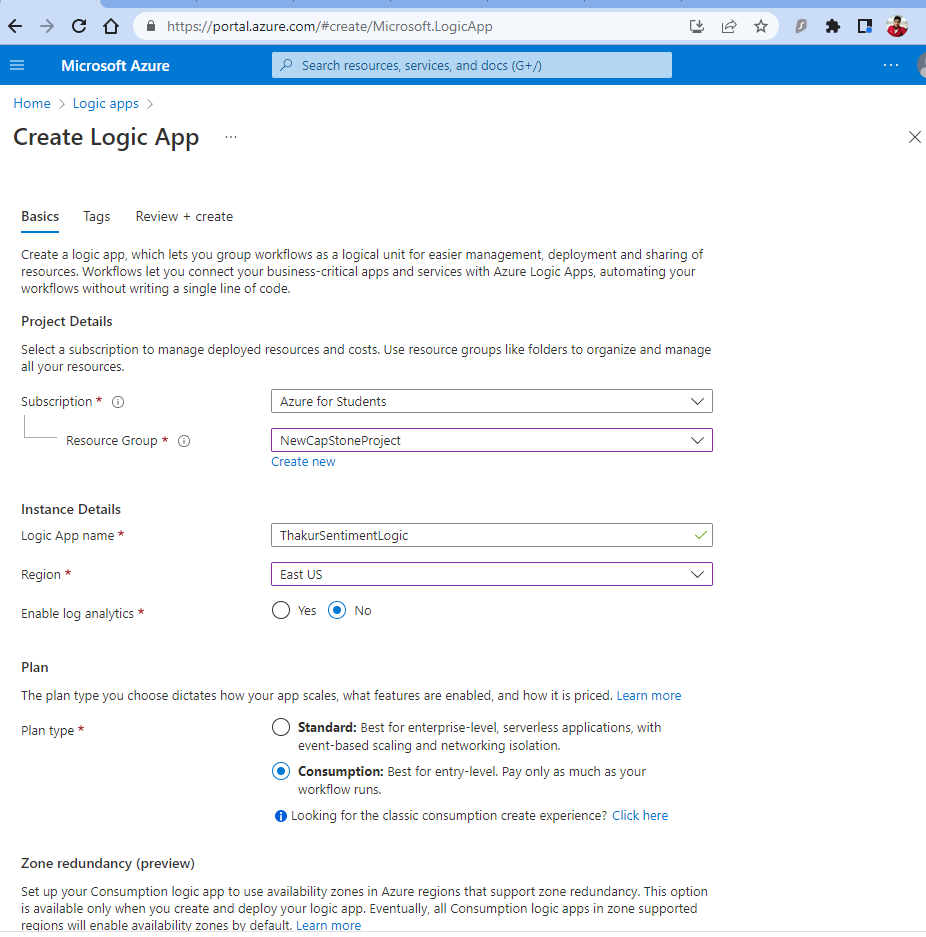


Figure 8: Azure logic App Creation

## Azure Logic App Configuration

Once the deployment of the Logic app has been successful, we head to the Logic App designer blade to integrate the Twitter API connection with the app. After this we will get an option to sign into a Twitter account and upon its completion, we will configure the values for the search text for a hashtag. We can also set a time frame for how often we want to check for the tweets.

There are restrictions for querying the Twitter API's for the number of connectors per account, number of post operations and other operations per hour, maximum number of new tweets tracked within one polling interval etc. These can be referred in this [link](https://docs.microsoft.com/en-us/connectors/twitterconnector/#limits) for more information.

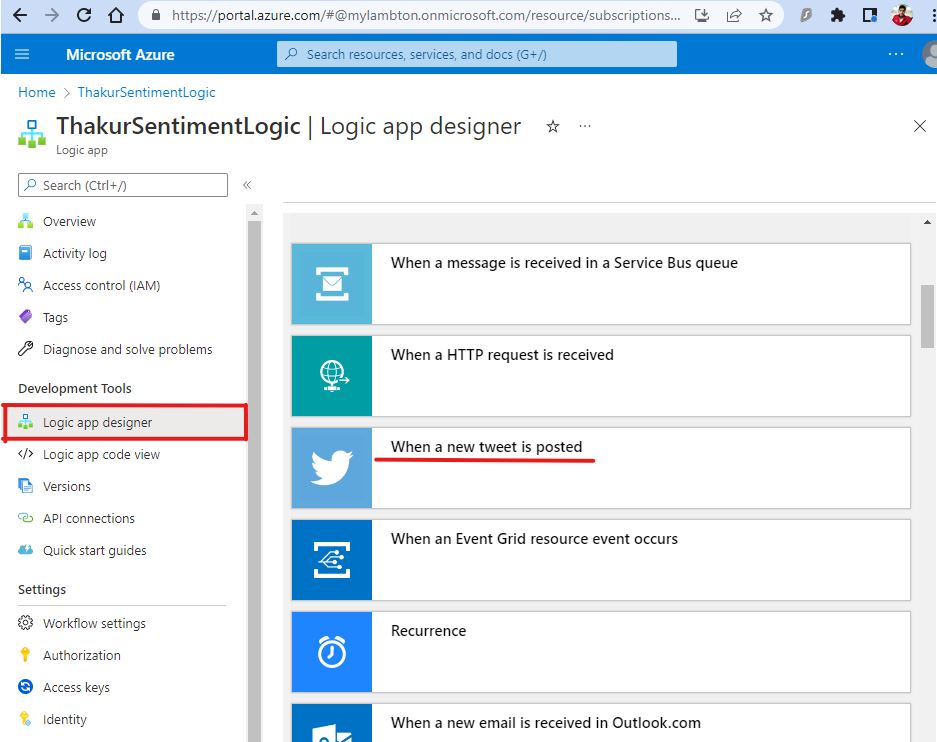


Figure 9: Logic App connection to Twitter API

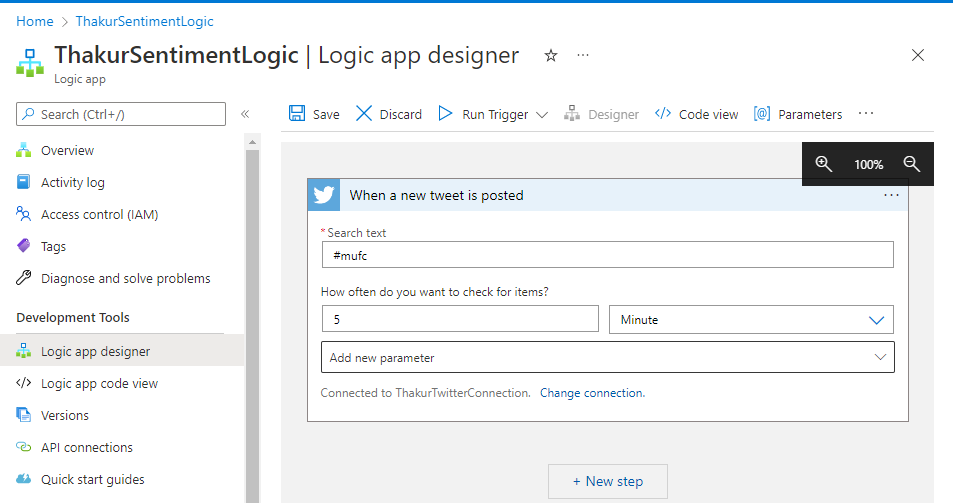


Figure 10: Configuration for the Tweet hashtag to be searched

The Next step is to configure the Text Analytics sentiment detection for our logic app and for that we can click on add new step and then configure Sentiment (v3.0).

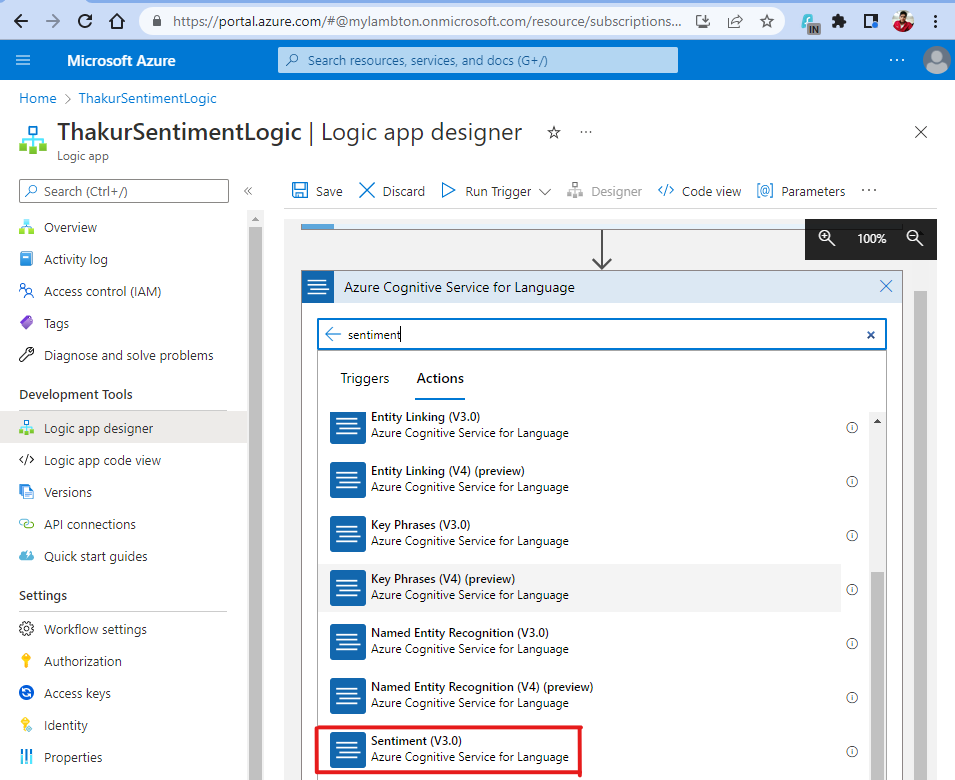


Figure 11: Configuring Text Analytics for sentiment analysis.

While configuring this we will need to enter the values for account key and site URL that we had initially set aside when creating the Azure Cognitive Service resource. The Key 1 value needs to be used as Account key and the Endpoint URL will be pasted in Site URL. Once this has been done, we can proceed and click on create.

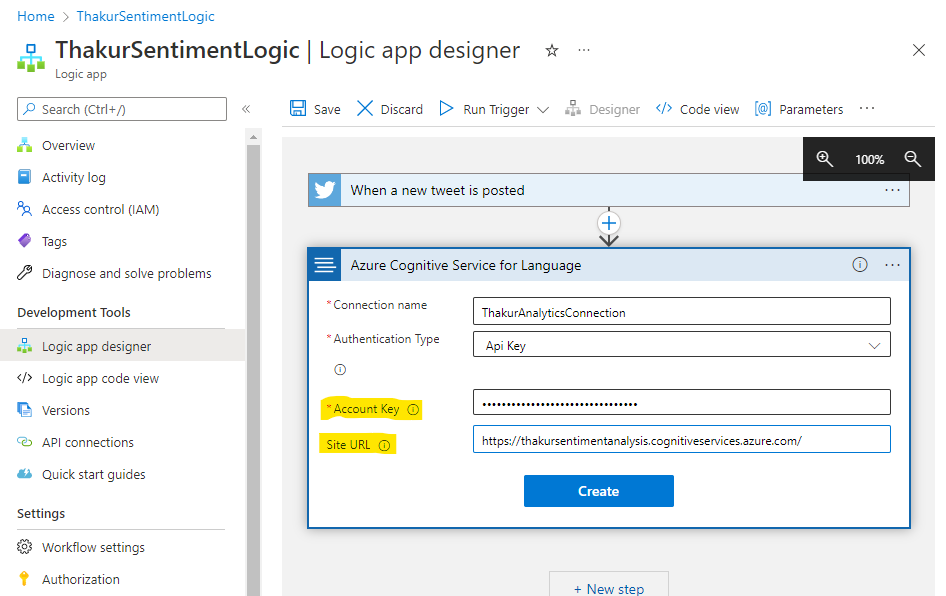


Figure 12: Configuring Text Analytics key and endpoints to Logic App Connector

Once this has been created, we need to enter the values for the documents ID and the documents text. In our case the "Tweet ID" will be considered as the document ID and the "Tweet text" will comprise the documents text in this scenario. These are dynamic values that get fetched from the Twitter API connection. To configure this, we get an option to search for dynamic text whenever we are querying the text boxes for detect language and we need to select the appropriate values as shown in the figure below.

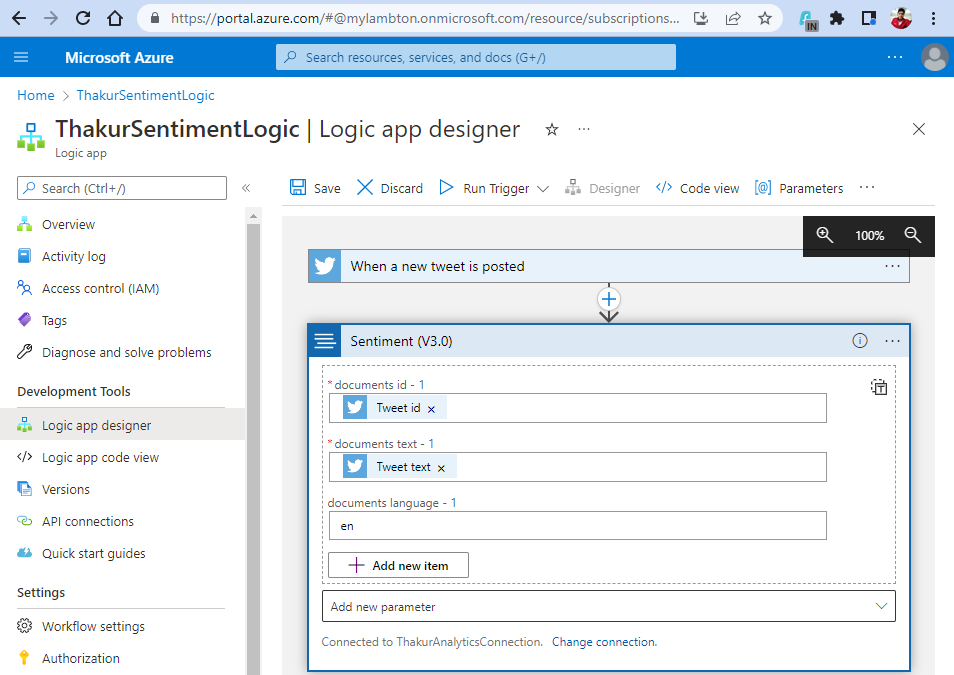


Figure 13: Feeding Twitter API connection values to the Cognitive Service.

 We will select the Function App that we had configured earlier with the code.

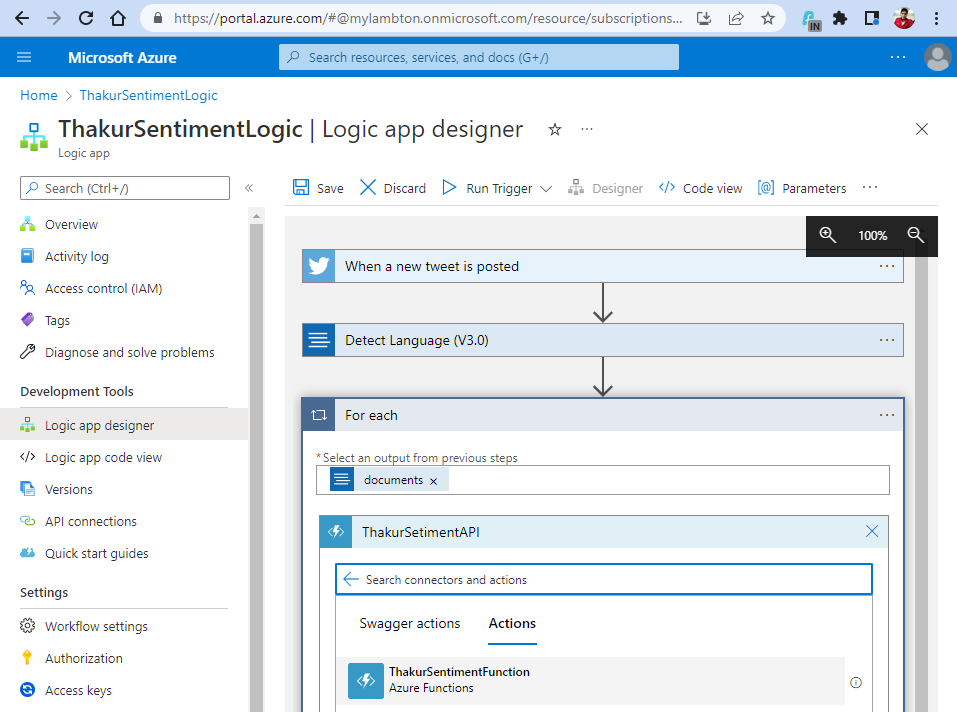


Figure 14: Connecting the Function App in the workflow

Next, we need to pass the Sentiment output from the function into the workflow. For this, the sentiment analysis done by the function app needs to be in the Request body which has been calculated from the Confidence scores of the Cognitive Service. Then the function app will further trigger the next components as we add them below.

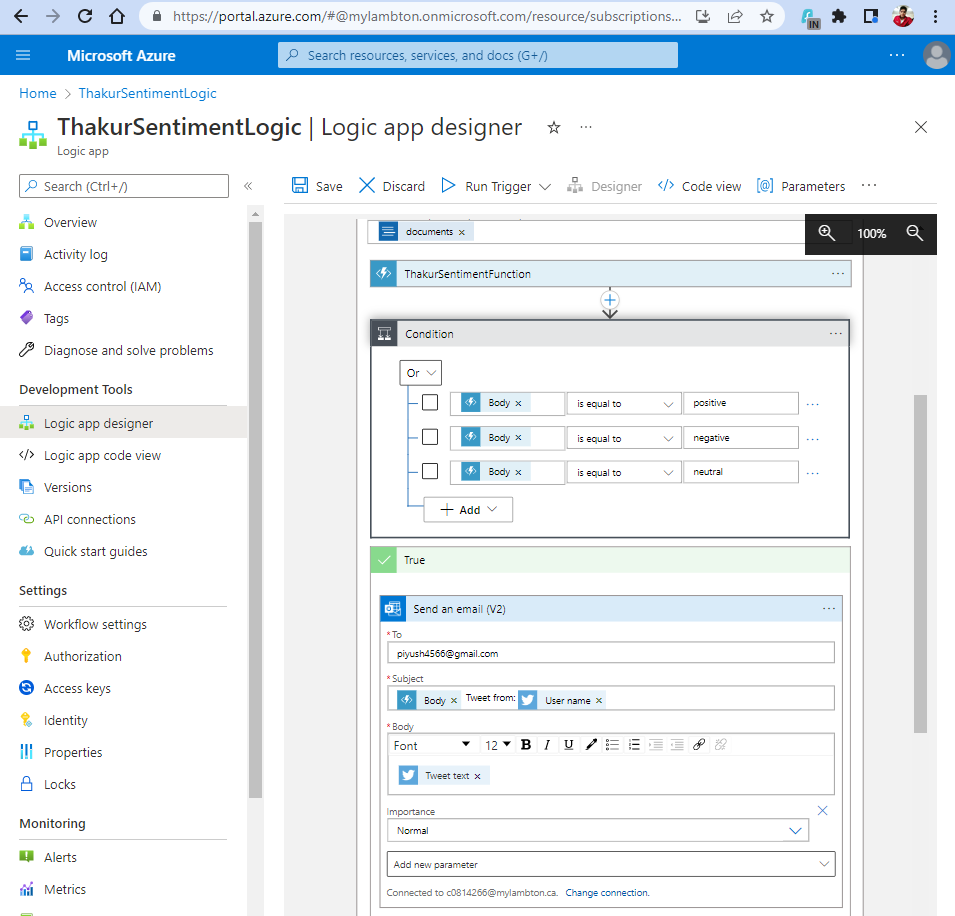


Figure 15: Creating the Conditional statements to trigger the Office 365 API Connection

For the next component we create a Control statement, and we chose to add a Condition Statement. The Condition statement has been configured in such a way where we have created three OR statements. For the Body of the function app output to be either Positive, Negative or

Neutral, an email will be sent out. For the email mechanism to work, we have incorporated Office 365 in our setup as well.

To construct the email and to make it dynamic with individual tweets contents every time, we have used dynamic Values being returned from the Function app and the Twitter API Connection for the Body and Subject of the email. The Username of the author of the tweet is also a part of the subject of the email.

Now that the setup is complete for the environment, the application will automatically trigger according to the frequency that we configure (in our scenario, every 5 hours). When the workflow is activated, we can see all the individual components getting activated and passing the information to the next component. As seen below, all the components execute successfully:

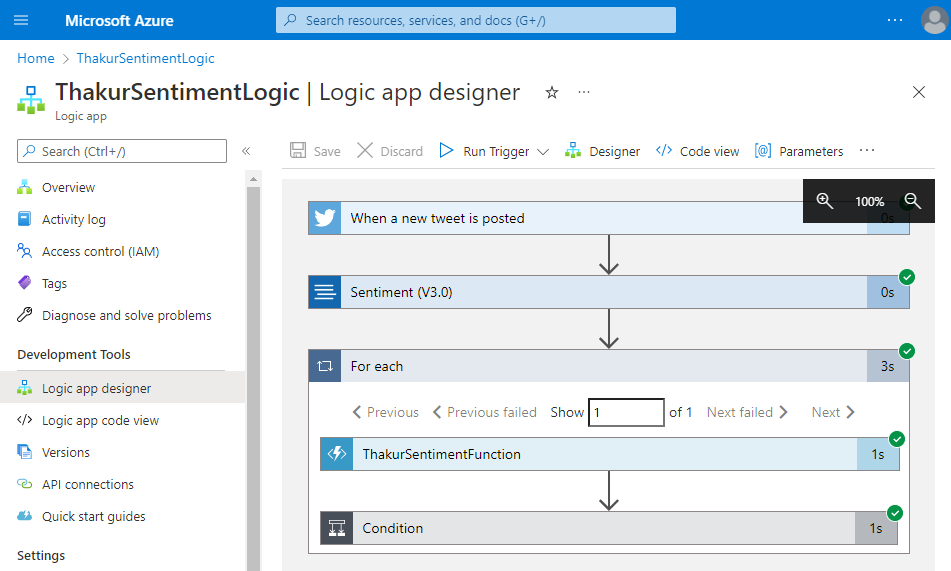


Figure 16: Successful execution of the workflow

**Automation for deploying the workflow:**

After we had successfully completed the entire workflow of the application, we wondered how we can make it accessible for more people to use our solution who are new to Azure and don't know how to work around the Azure portal and all the different resources that need configuration. What if we could also help customers who want to deploy this solution in multiple subscriptions but do not want to spend time in creating and configuring all these resources manually for each subscription.

While researching on all the different automation techniques that can be used to create this workflow, apart from creating them on the portal, we found that ARM templates can be used as a great alternative to complete the deployment of all the different resources along with their configuration in a single go. An ARM Template is a JSON file (JavaScript Object Notation) which includes definition of the infrastructure and configurations for your project. In the template, we need to specify the resources along with their properties that we wish to deploy.

This will allow people to use and test this solution without having to understand how to configure all the different components like function apps, logic apps, Azure cognitive services etc. ARM Templates can be considered to be a form of Infrastructure as a Code, which helps us to define the infrastructure that we want to deploy. This concept is also sometimes referred to as declarative syntax. Declarative syntax is when we state what resources we want to create without the need of declaring the sequence of the commands needed to make it. When we use these declarative templates, we're essentially defining the environments final end state.

We have made the ARM template for this Project Available on GitHub, from where anyone can use this and paste it in the Custom Deployment feature on Azure:

<https://github.com/PiyushThakur266/PiyushRepo/blob/main/CapStone%20ARM%20Template>

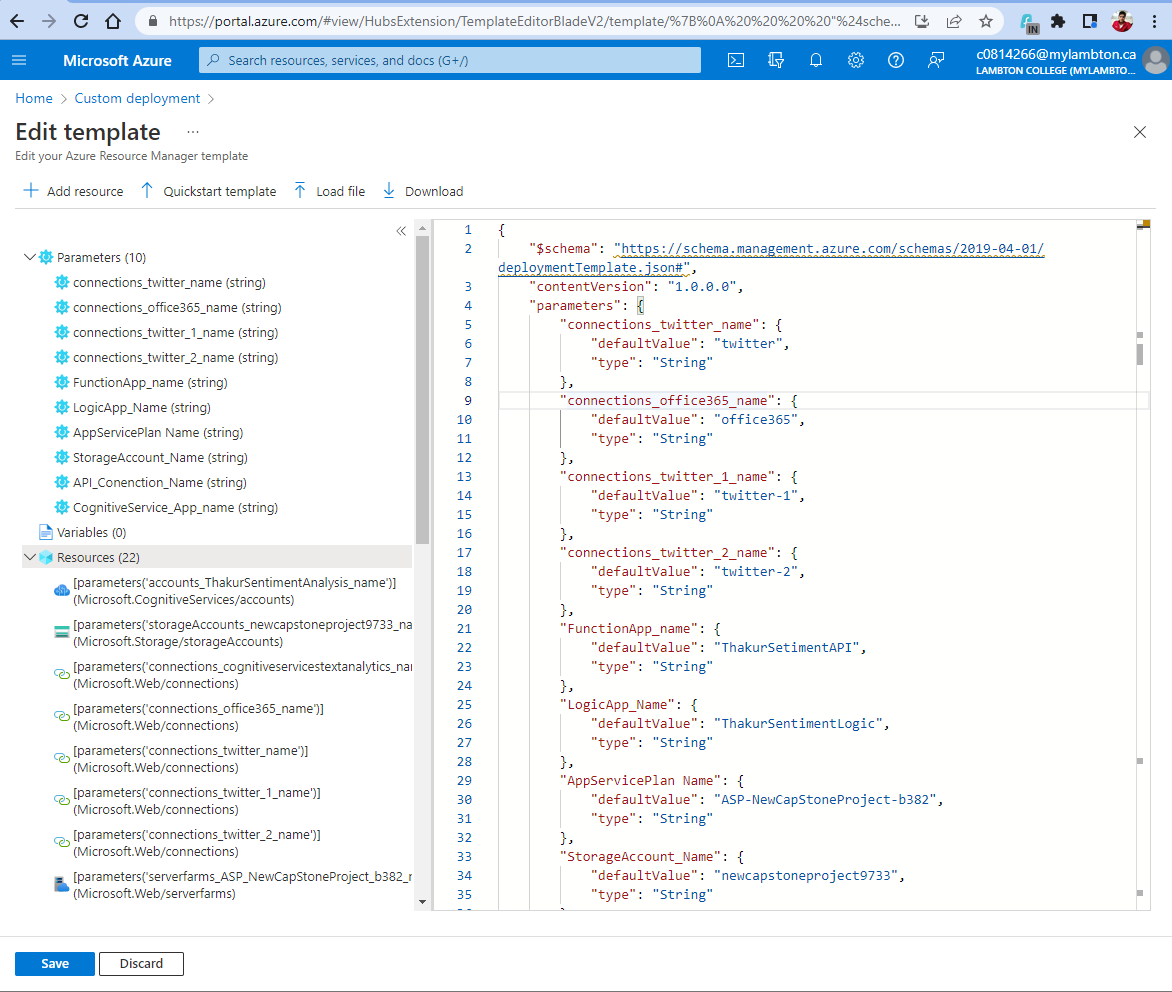


Figure 17: Custom Deployment feature to create the workflow in one go

Once the Template has been pasted and saved in the custom deployment template editor, we get individual text boxes where the user can customize the name of each of the resources. The labels for all these resources have been defined by us properly such that the new user knows what service they are creating and therefore create an appropriate name for it. Apart from that the user needs to choose the resource Group in which they want to perform the deployment along with the region where all the resources should be created. Once this is done the user can click on review and create and begin the deployment process.

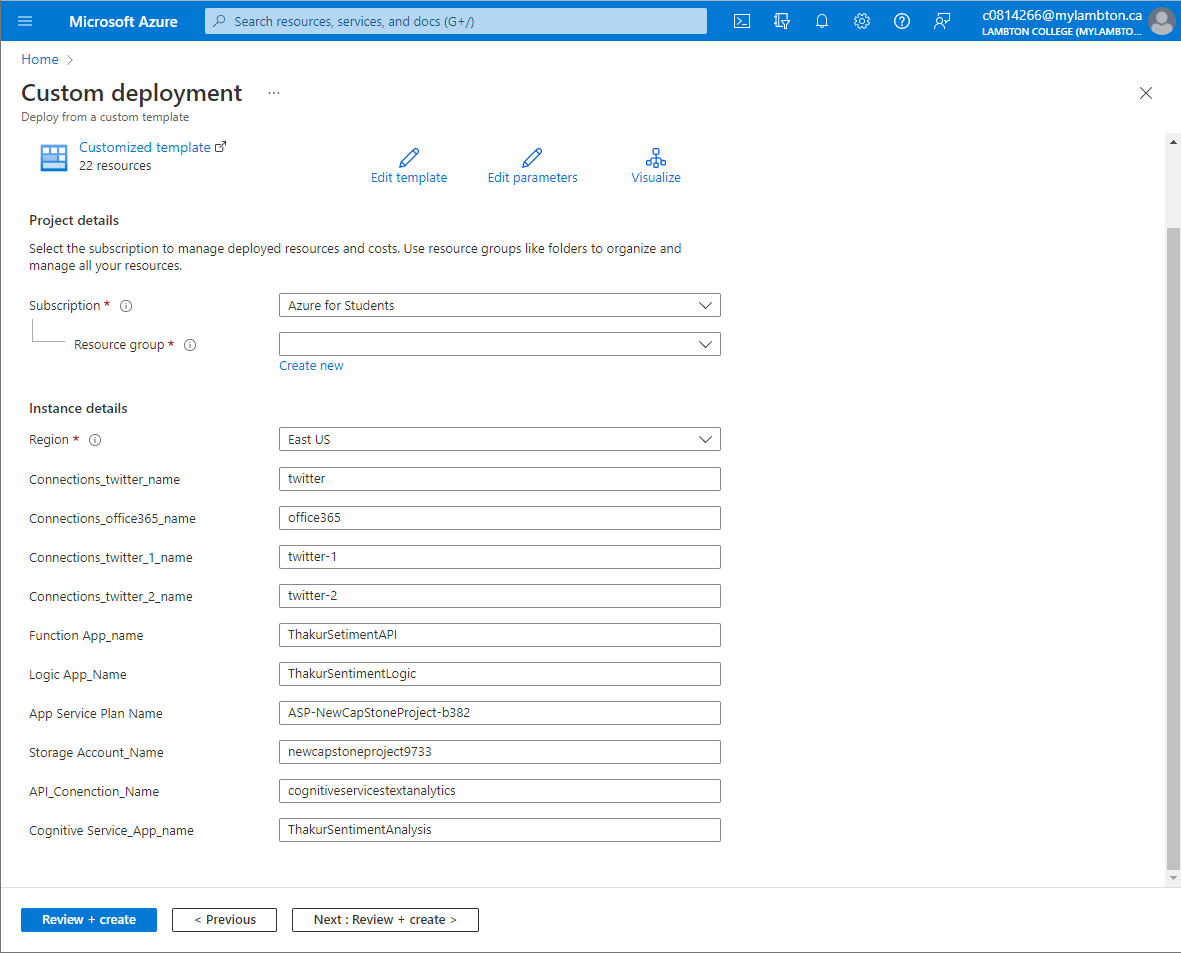


Figure 18: Declaring the resource names before applying the ARM template.

# **Results**

To show the results of the application workflow, we have chosen the hashtag #Yphone14 simulating a scenario where our “customer” has announced their new device YPhone 14, which will be hitting the markets next month. They have launched a public campaign where they are asking potential customer to tweet using this hashtag to let them know what they like the most about the new device. Using our application, they will get notified of tweets that the customers are making and if they find tweets which have a negative sentiment, the customer can use the feedback to make changes to the device specifications and features, its pricing, marketing strategies to make the product a success at launch. Here is a look at some of the tweets that customers have made using the hashtag:

**\*\*\*\*\*\*\*\*\*\*\*\*<Image of Tweets from twitter, we will take tomorrow**>\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Simultaneously, we can also see the Customer getting notified of the tweets along with their sentiments to absorb feedback to improve their devices via email. into our Inbox where we have been notified for recent tweets for #mufc along with their Sentiment Analysis, the Author of the tweet and the Body of the tweet as well.\

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*<We will replace this with a better screenshot of the email mechanism once the tweets are posted>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***



# **Conclusions and Future Work**

To conclude, our project provides users a very efficient, easy and affordable way of experiencing the power of Natural Language Processing in the form of Sentiment Analysis. The workflow has been created keeping in mind that it does not generate any excessive cost for the user when there are no application executions happening. This has been achieved due to the serverless architecture approach which has been implemented both for the function app as well as the logic app. Moreover, with the power of ARM templates, the automation solution that we have provided to deploy the entire workflow in one click, makes this an accessible approach if the user requires to perform the deployment of the entire project in multiple Azure subscriptions.

During our research we had also decided to provide an automation option using PowerShell. This would have allowed the user to deploy the entire workflow on Azure without even needing to login to the Azure Portal. The user can login to the Azure account on PowerShell using the *Connect-AzAccount* command and will have access to all their subscriptions. The user would have been able to select a subscription and execute the PowerShell script to deploy the entire workflow on the subscription. To redeploy it in a different subscription, all the user would need to do is change his subscription ID and run the PowerShell script again and deploy the workflow in as many subscriptions as needed.

However, to deploy an ARM template using PowerShell, we would need to create a different file for all the parameters as well as needed to connect our PowerShell session to a storage account to be successfully able to do so. In the interest of completing the Capstone project before the deadline, we were unable to complete our research on how to achieve this and had to drop the idea. We would like to invest more time on this in the future ad incorporating this feature in our Project as well.

# **References**

<https://docs.microsoft.com/en-us/azure/cognitive-services/what-are-cognitive-services>

<https://docs.microsoft.com/en-us/azure/cognitive-services/language-service/overview>

<https://www.serverless360.com/azure-functions#:~:text=Azure%20Functions%20are%20the%20individual,by%20providing%20some%20raw%20inputs>.

<https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview>

<https://docs.microsoft.com/en-us/azure/azure-functions/>

<https://brand24.com/blog/sentiment-analysis/#:~:text=Sentiment%20analysis%20is%20a%20technique,customers%20feel%20about%20their%20company>.

<https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview>

<https://www.skylinesacademy.com/blog/2019/8/15/arm-templates-part-1>