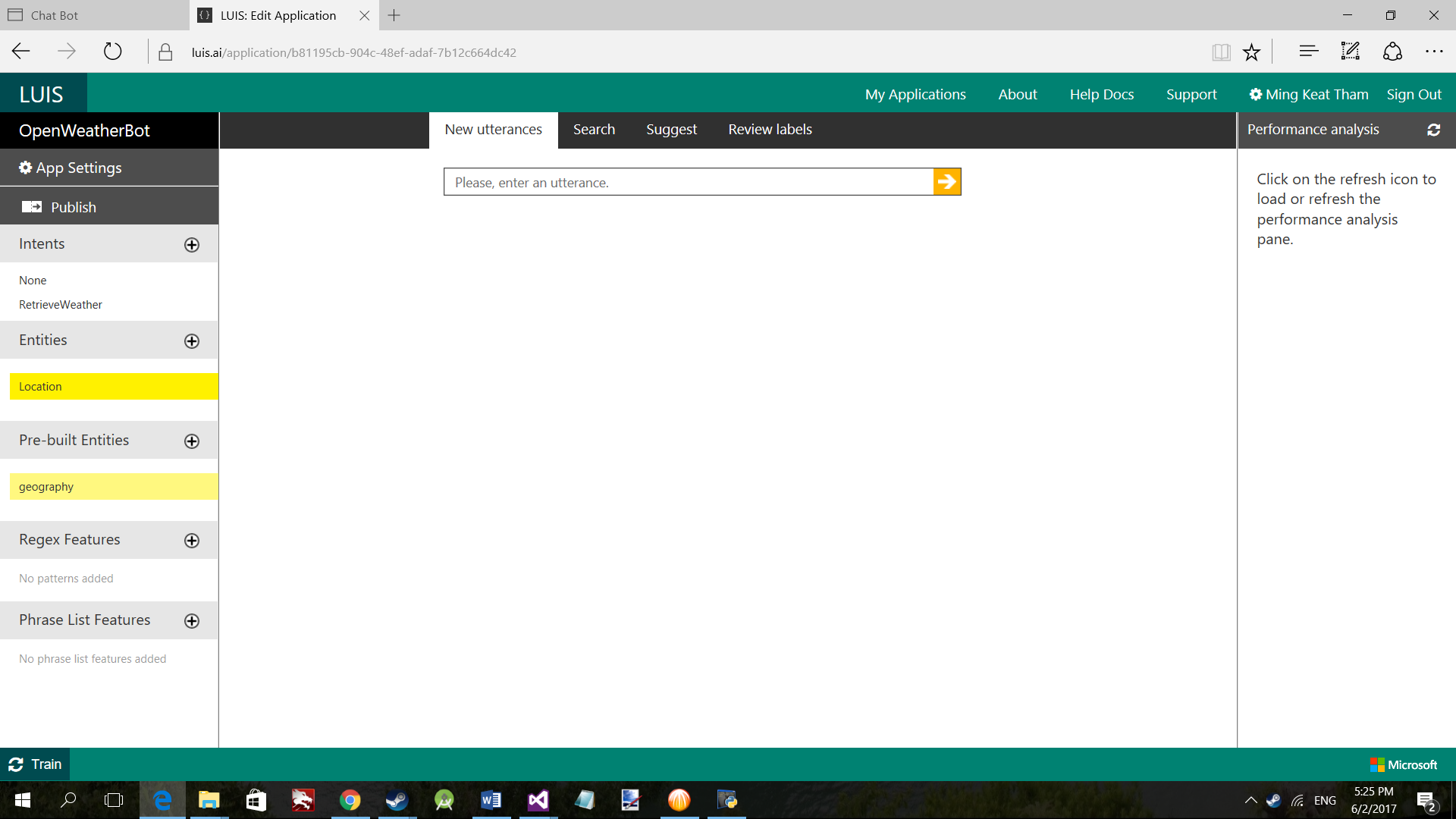
**Introduction**

Microsoft's Language Understanding and Intelligent Services (LUIS) would be integrated with OpenWeather API to build our own Python chatbot. We would communicate with the chatbot through a web chat. The reason for this project is to introduce Microsoft's Cognitive Services and in this case it would be LUIS with Python. There are plenty of other cognitive services that are also provided by Microsoft.

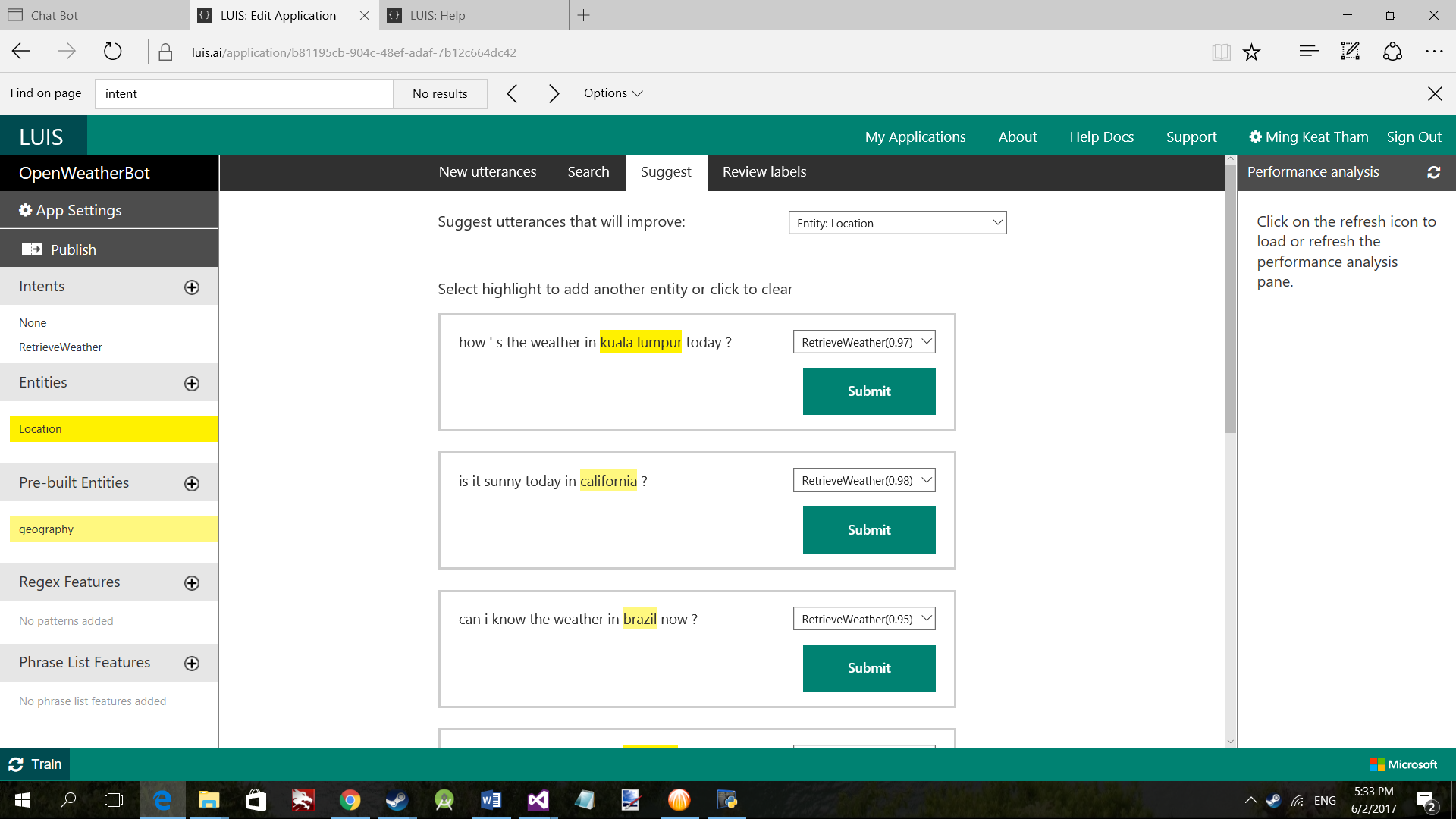
**Pre-requisites**

1. Python Tools for Visual Studio (<https://www.visualstudio.com/vs/python/>)
2. Python 3.0 or above Environment
3. Simple WebSocket Server (<https://github.com/dpallot/simple-websocket-server>)
4. LUIS SDK for Python (<https://github.com/Microsoft?utf8=%E2%9C%93&q=cognitive%20python&type=&language>) \*Already included in the project
5. A subscription in Azure or free trial ([https://portal.azure.com](https://portal.azure.com/))
6. A free account for LUIS which can be registered here (<https://www.luis.ai/>)
7. A free account in OpenWeather to use the APIs (<https://openweathermap.org/>)

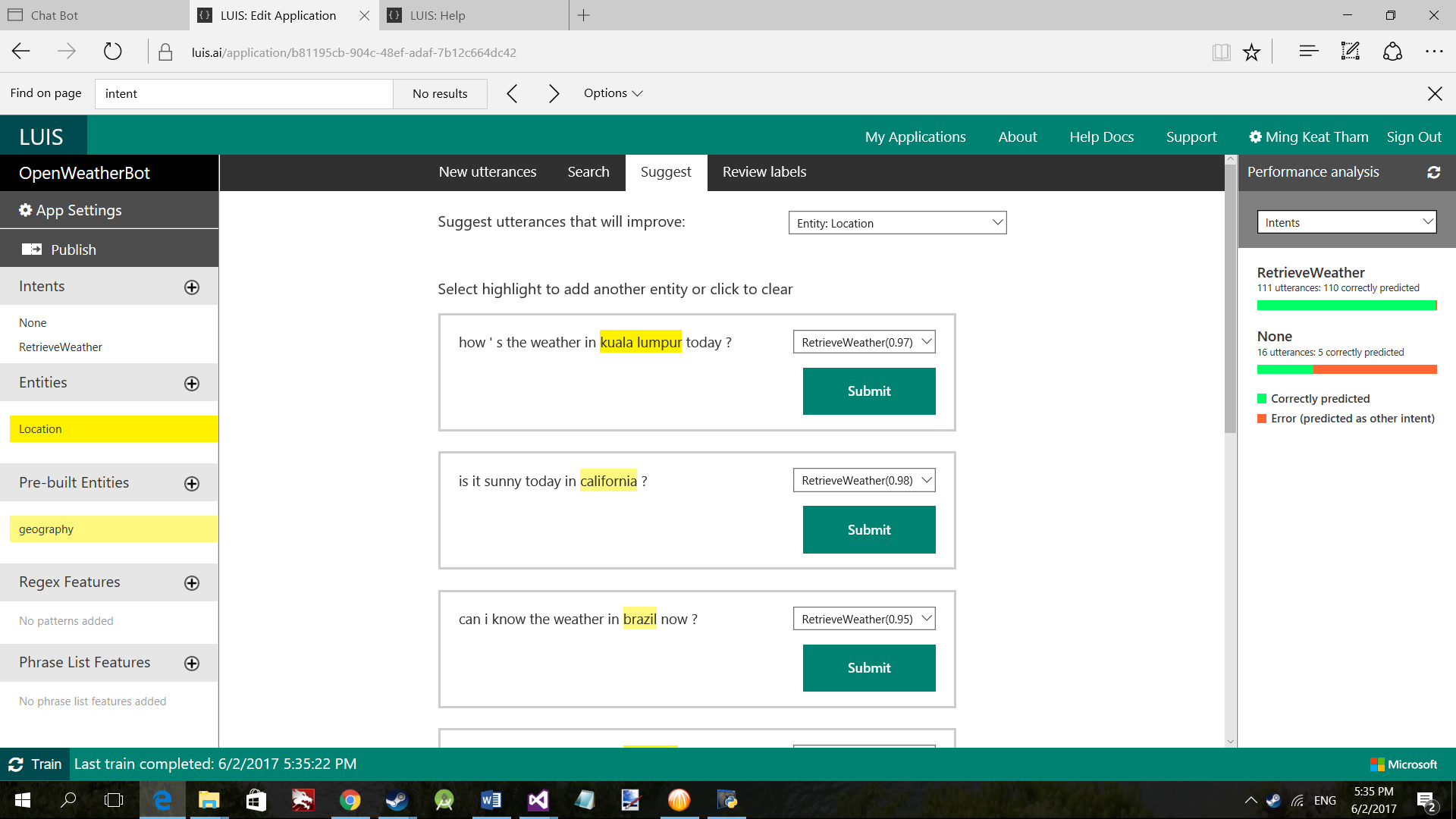
**Setting up LUIS app**

Once you have logged in to LUIS, select create a new app and fill in the form with the necessary details. After that, go in to the application interface for your newly created app and you should have something similar as the screenshot.   
  
  
After that, click on the plus sign (+) next to the intents and add a new intent with a similar name such as RetrieveWeather. Do the same for the entities by adding a new entity called Location.  
  
Intents are the tasks that the user wants to do and entities are the models or data that is required for the intent. In this case, our intent would be to retrieve the weather and the entity would be the location or place that the user wants to find out about the weather there.   
  
Once done, click on the app settings on the top left and add the new key that you will be buying from Azure. Once the key has been added, select the appropriate endpoint key. On how to get the app key for LUIS, refer to the “Creating Subscription Keys Via Azure” in the link provided. (https://www.luis.ai/Help)

At the top of the page there would be new utterances, search and suggest.  
New utterances is where you can enter a phrase or a sentence to train the app. Search is used to search for the previous utterances and suggest is when new utterances has not been labelled. It can come from the HTTP endpoint or from the user adding new utterances. It would look something like the screenshot below.



The drop down list on the right side allows you to select the appropriate intent if it was mislabelled. The entities would be highlighted in yellow and can also manually be changed by highlighting or selecting on the words.



The performance analysis on the right side would show how many utterances were predicted correctly. The drop down list provides analysis for different intents and entities.  
By clicking on publish, a window will be displayed to allow you to enter a query for testing the app as well as additional properties and also the URL for posting queries.

Once you have finalized everything, click on the update published application to update the app.

**Python Project**

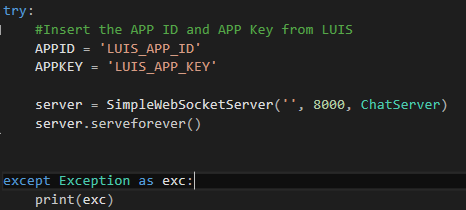
Download the project from the GitHub repository :   
  
<https://github.com/ThamMK/OpenWeatherLUIS>

Overview of the files :

**Index.html** : The web browser chat that was written by MSP Kai Alan Du  
sample\_app.py : A sample app provided in the LUIS SDK on using LUIS in Python  
sample\_app\_async.py : Similar to the sample app but the calls to LUIS would be done asynchronously

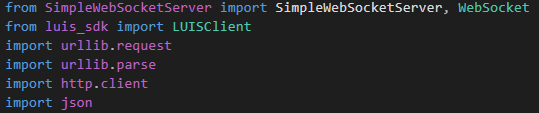
**Server\_incomplete.py** : An incomplete server file that you can use to modify

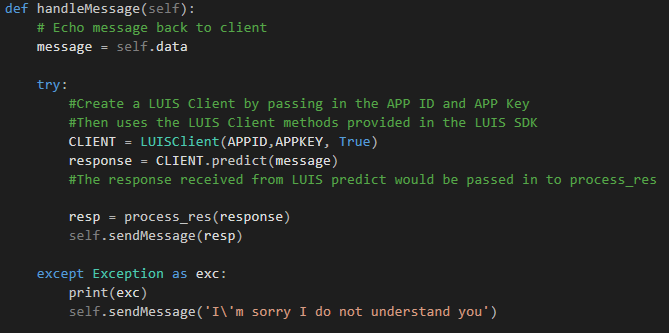
**Server.py** : The server that would be retrieving the user messages from the web browser chat, passing in the message to our created LUIS app, retrieving the JSON response and then passing the required data to the OpenWeather API and replying the user.  
Files in the luis\_sdk can be found here :   
<https://github.com/Microsoft/Cognitive-LUIS-Python>

Before we begin, in the server\_incomplete.py, replace the APP ID and APP KEY from the LUIS app.  
  


The code is wrapped in a try and except to catch any exceptions that may occur. This is the starting point of the code when running this file where a simple web socket server would be created.

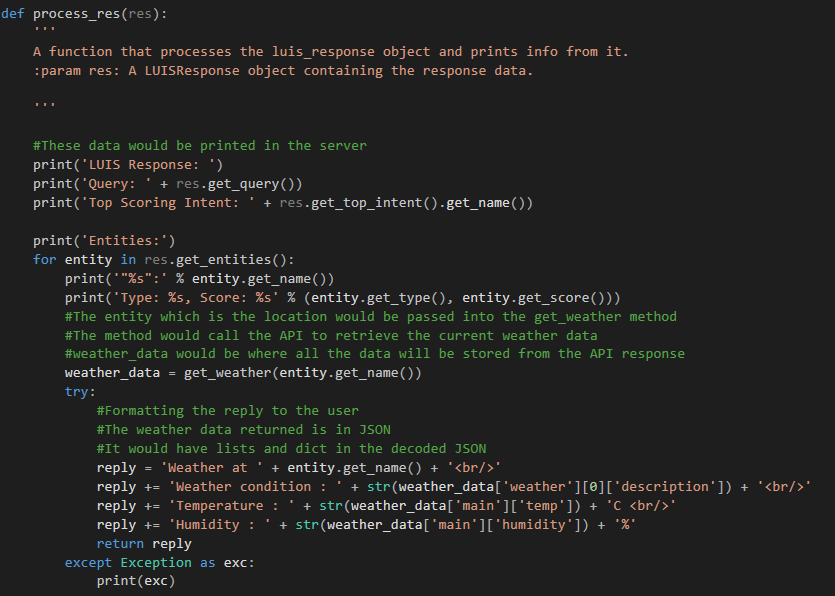
The imports required for this project



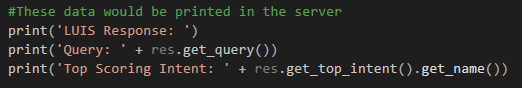


Inside the handleMessage method where it is located inside the ChatServer class, create another try and except. Then, a LUIS Client object would be created which is taken from the LUIS SDK and passing in the APP ID, APP Key and a true value. Inside the LUIS SDK, there would be a luis\_client.py and inside the file would contain more explanation for each of the methods.

After that, the client would be used with a predict method and the message is passed in as a parameter. The predict method would be where the message received from user would then be passed to LUIS for prediction. A LUIS response object would be received back from the method which is then passed into another method called process\_res(). The self.sendMessage(resp) would be where the server sends replies to the user in the web browser chat.

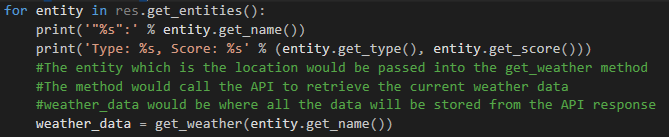


The process\_res() is a function or method that processes the LUIS response object to retrieve the info from it.

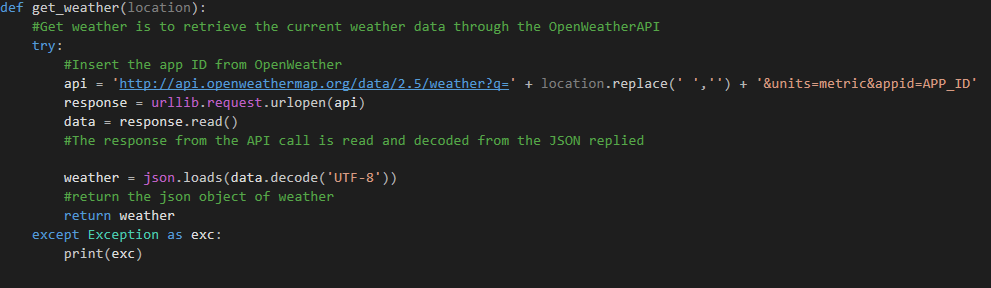


Res.get\_query() would be to retrieve the message that was sent by the user.

Res.get\_top\_intent().get\_name() would be to retrieve the name for the top intent. This is to know what the intent of the user is.

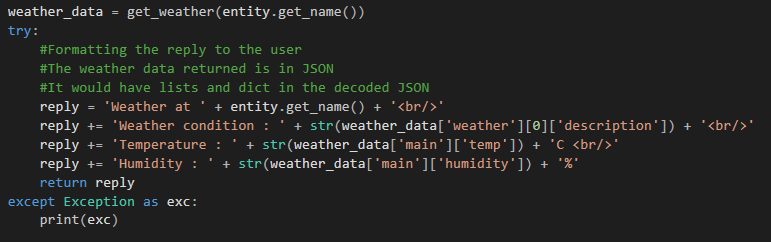


A for loop is used to iterate through each entity that could be detected in the query. Each entity would be printed with their score. The score indicates the accuracy or confidence that it is an entity. The entity, which we have declared earlier in our LUIS app which is a location would then be passed into the get\_weather() method.



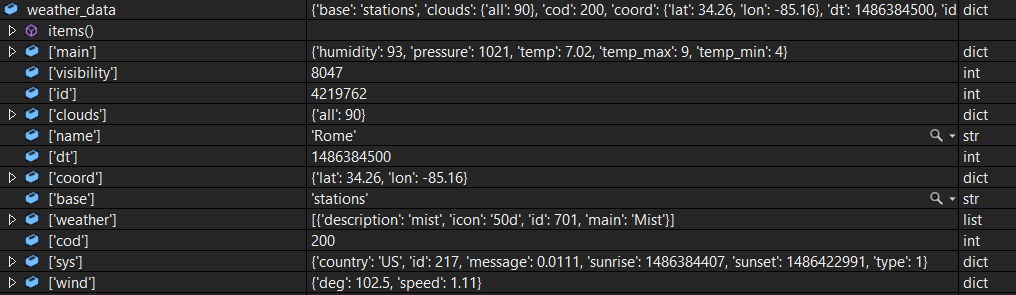
The get\_weather() method is to connect to the OpenWeather API to retrieve the current weather data for the location. The location which is passed in as a string has to remove the empty spaces for locations such as “Kuala Lumpur”. The spaces need to be removed before passing it into the API. An additional parameter has been passed in which is the units, as it can be either metric or imperial. The app ID has to be replaced with your app ID that you have received when creating the account in OpenWeather and has subscribed to the API that you would like.

The response read from OpenWeather API is in the form of JSON which needs to be decoded first.



Once the get\_weather() is completed, it would return back to process the response. Here, we would create a string to reply back to the user. The weather\_data object contains nested list and dicts which requires careful retrieval of the data.

The weather\_data that was decoded from the JSON response is show in the screenshot below.



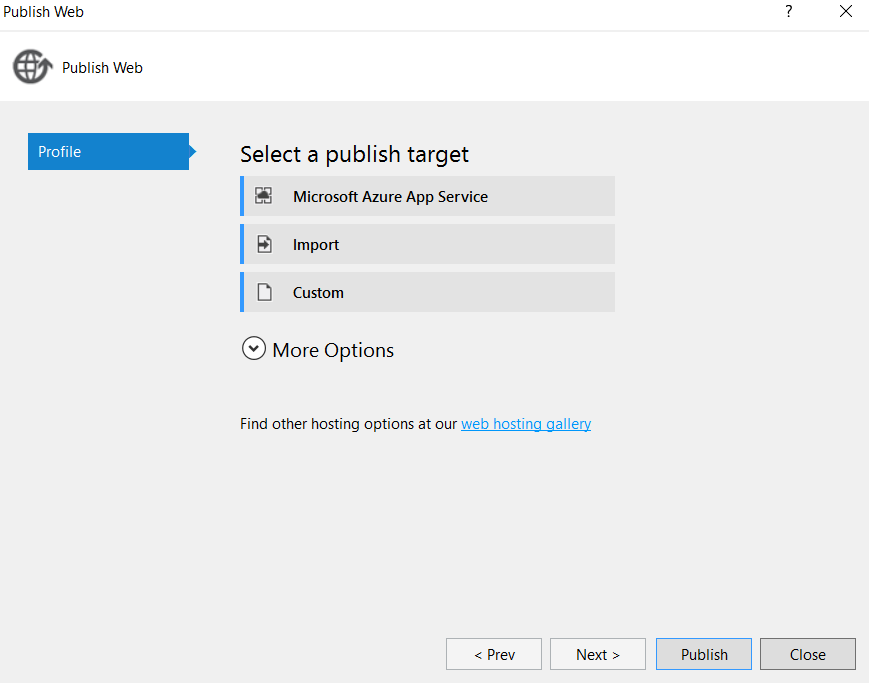
Once everything has been successfully done as shown, run the index.html on a browser then set the server\_incomplete.py or server.py as a start up file in Visual Studio. Start the debug process and you can type messages on the chat. It should look something like this.

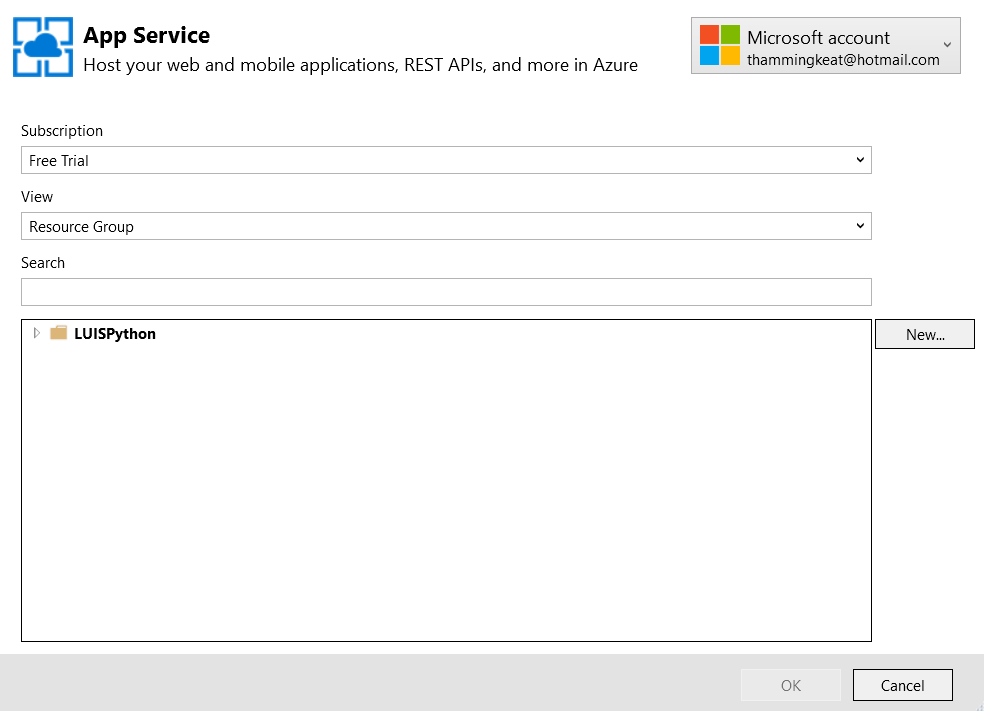


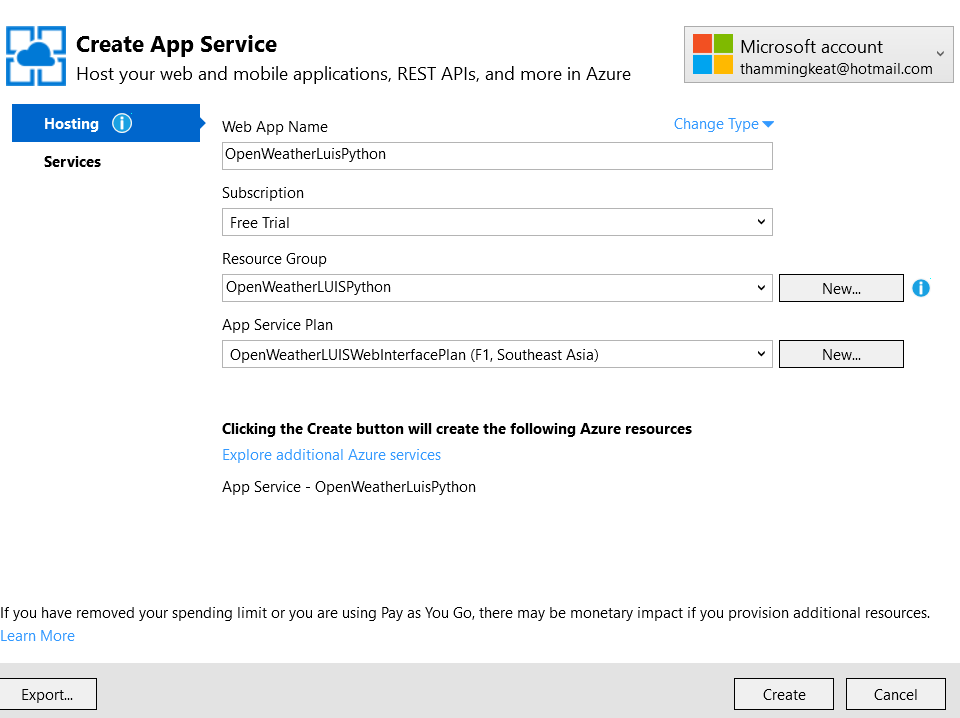
There you have it! You have successfully written a very simple chat bot that allows user to ask for the weather! We barely scratched the surface with LUIS and Python as there are plenty more things to explore. I hope this guide would help to pave the way for you to be more interested and to learn more about LUIS and other cognitive services.

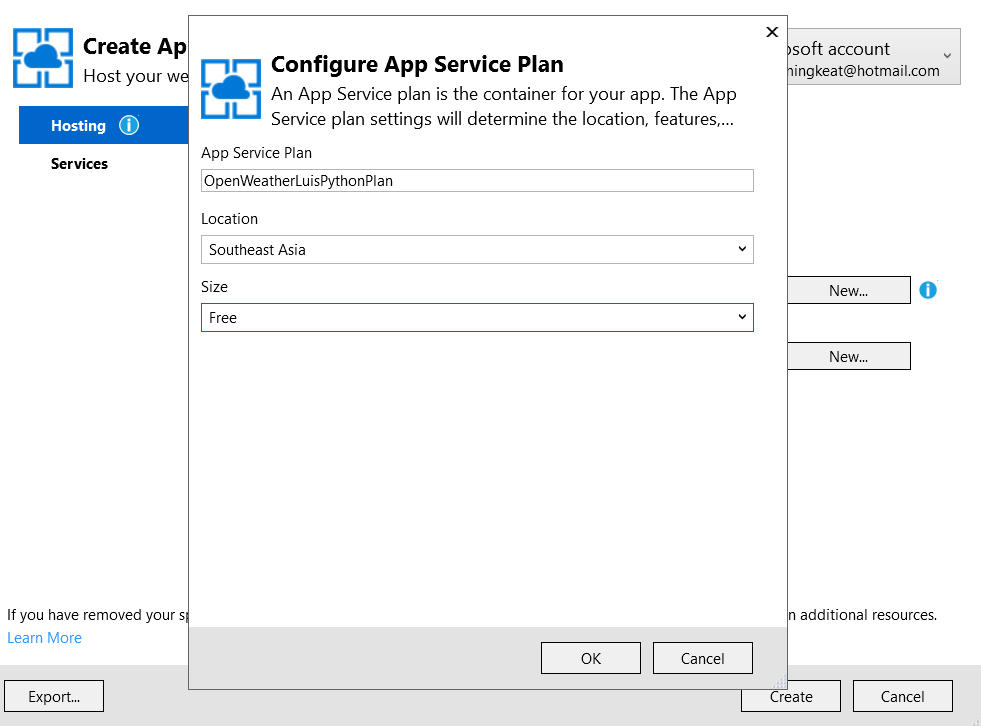
**Hosting the chatbot on Azure**

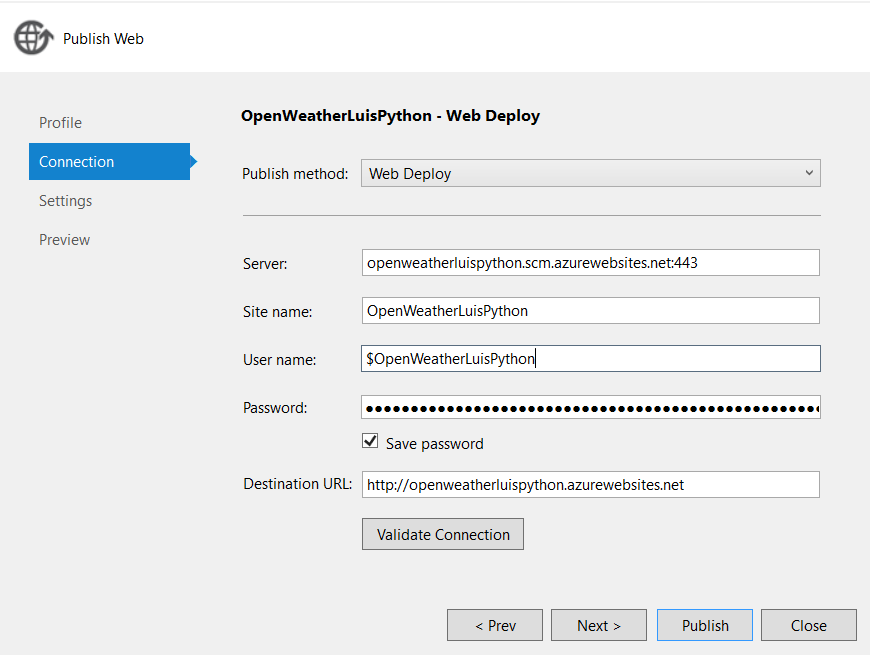
1. Right click the project folder in Visual Studio and select Publish.



1. Select Microsoft Azure App Service and then login to your Azure account. Select a free trial subscription group and then select a resource group or create a new one if there isn’t one.  
   
2. Fill in the necessary details and select the appropriate resource group and app service plan if there is. Create a new one if there isn’t one.

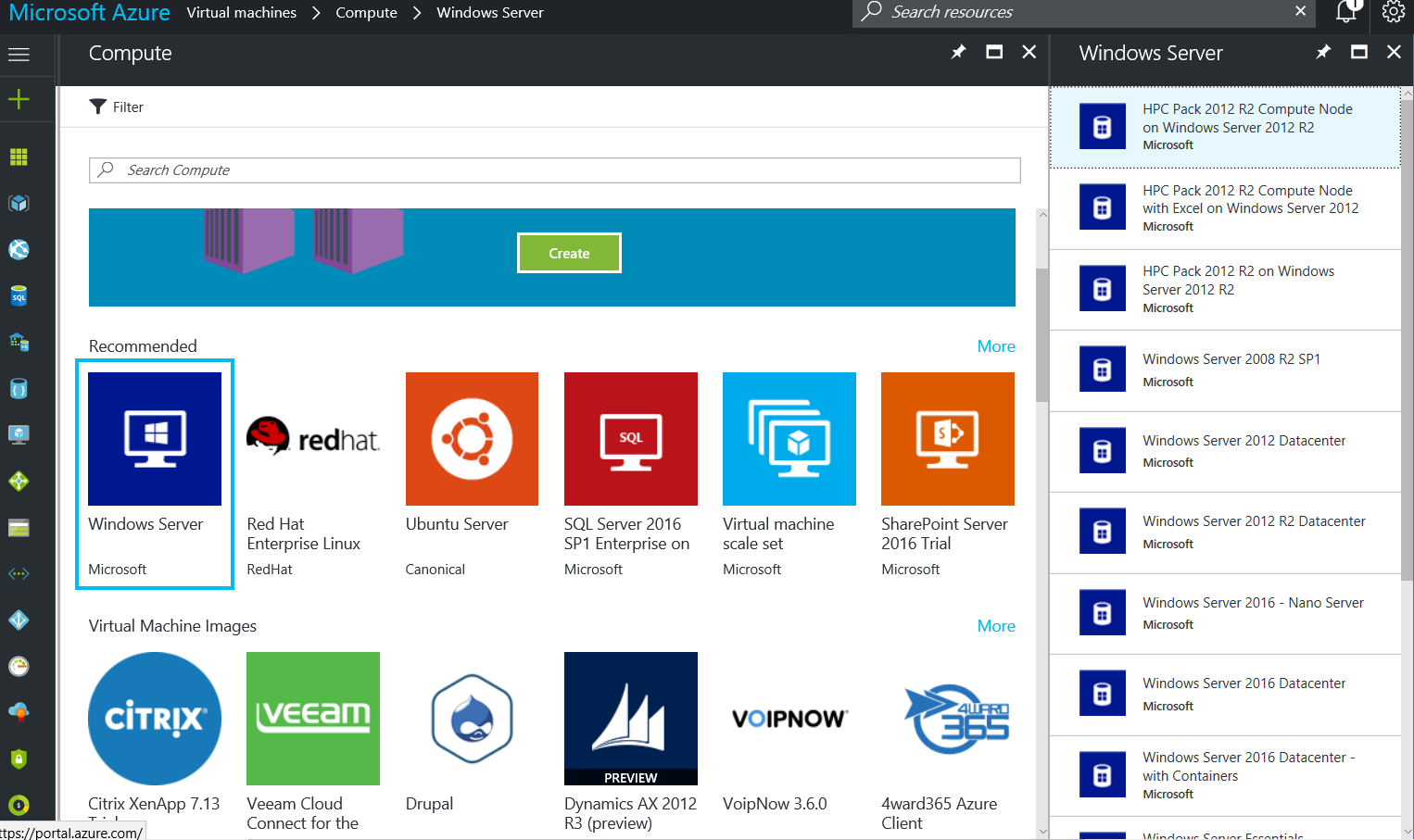


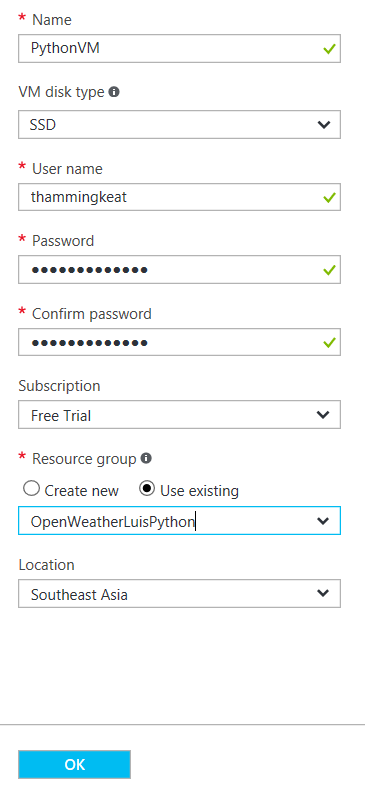


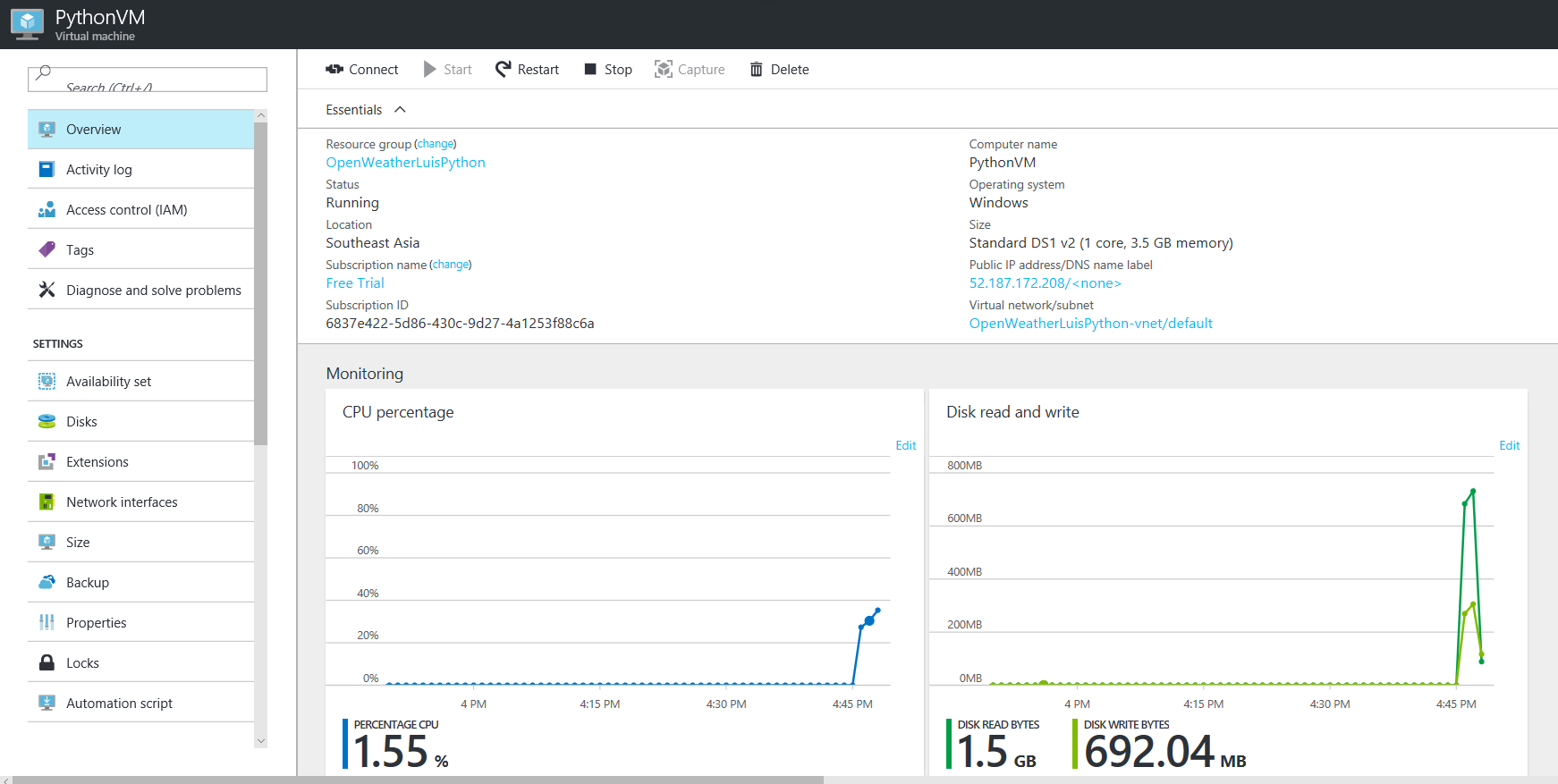
1. Once everything has been filled in, click create and deployment would be started for the app service plan and resource group.
2. You will then be brought to the connection part and most of the details would be already filled in.   
   
3. Select publish and Visual Studio will publish the web app onto Azure automatically. Once it has been successful, your default web browser should open up the web app for you.

However, we aren’t complete yet! That’s because previously server.py was running locally on your computer. Now we want to be able to communicate with the server from other computers! We would then be required to create a virtual machine on Azure and then have it run server.py. After that, we would then modify index.html for it to connect to the virtual machine!

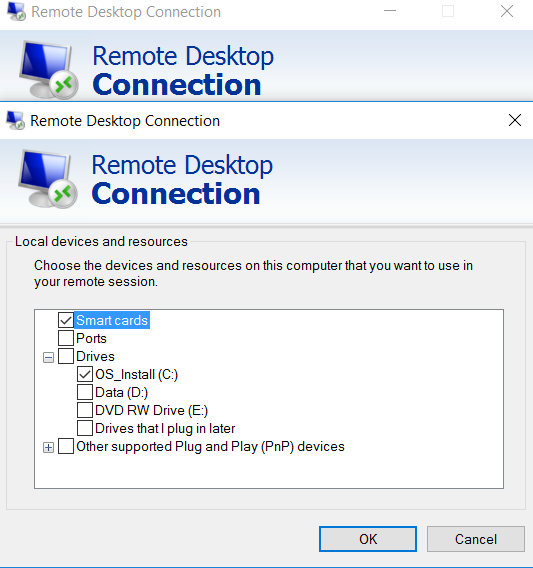
1. Login to <http://portal.azure.com> and select virtual machine. Click add then select Windows Server and in this guide we would be using Windows Server 2016 Datacenter. Alternatively, you could directly search for Windows Server on the market place in Azure.



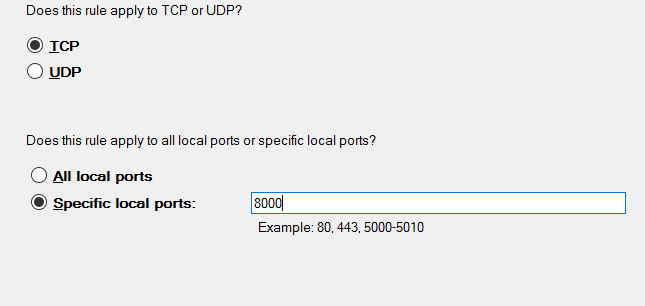
1. Click create and then fill in the details as follow. You should also use the existing resource group that we have previously created in Visual Studio. Click OK then select the appropriate server size that you would like.   
   
2. Click ok for the settings if you do not have any additional settings and then proceed to the summary and click ok as well. After that, the virtual machine will be deployed and might require a few minutes.

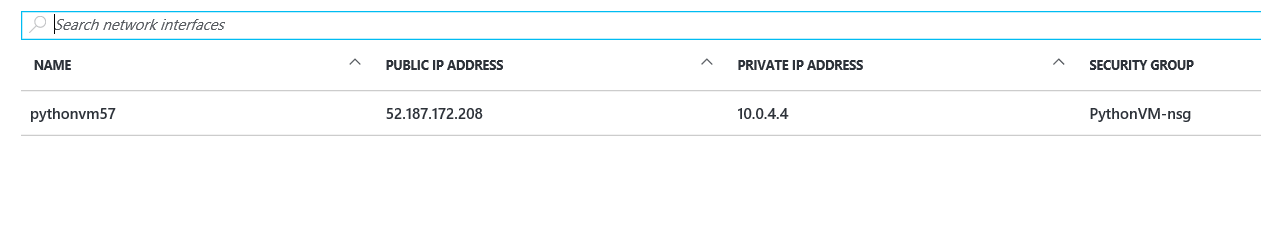


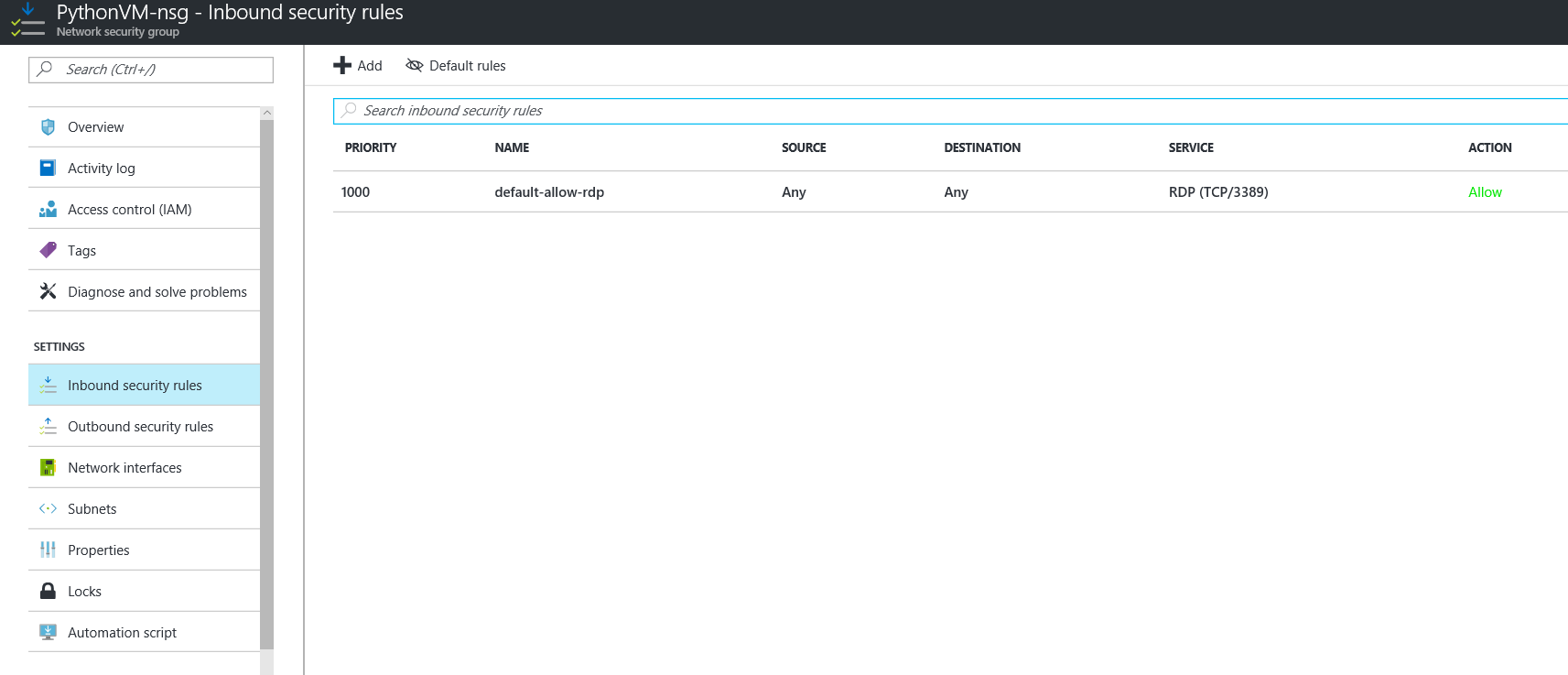
1. After that, the first thing we would need to do is to install Python on the server and run server.py. We would also need to install the necessary libraries as well as the LUIS SDK. To transfer the files from our desktop to the server, start remote desktop connection which can be found on your computer.
2. Select show options, local resources, and under local devices and resources select more. Ensure that the drive where your project is saved is enabled. This is to allow transfer of files between our desktop and the server easily.



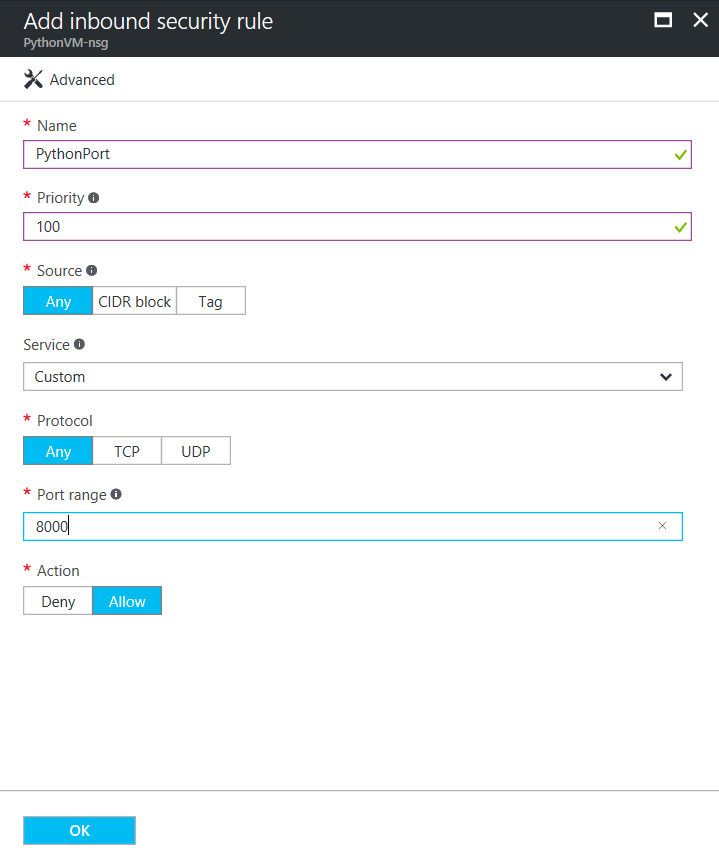
1. Click connect, then use the account that you have created on Azure when creating the virtual machine. The interface would be similar to a desktop. Then, install Python 3.4 on the server similar to how installation would take place on a desktop. (<https://www.python.org/download/releases/3.4.0/>)
2. In the mean time, transfer the server.py, Luis SDK folder and SimpleWebSocketServer folder from your desktop into the server and save it in an easily accessible location. This allows easy installation of the library that we require for the Python script to run. Install the files by using command prompt and navigating to the respective folders. The simple web socket server can be downloaded from this link. (<https://github.com/dpallot/simple-websocket-server>)
3. After installing the required files, open Windows Firewall with Advanced Security, select inbound rules and then click new rule.
4. Select port, click next and then enter a specific port. In this case we would be using 8000. Click next, ensuring that allow connection has been selected then click next again and then apply the rule to domain, private and public. Click next and then fill in the details. This is to allow the port to be accessible.



1. Run server.py by using command prompt and ensure that the port numbers are correct and identical.
2. Back to Azure, proceed back to the virtual machine that we created earlier and then select network interface on the list located on the left.
3. Select the network interface then select network security group on the left. There should be a network security group that can be selected. On the list on the left, select inbound security rules.



1. Click add, then fill in the required details for adding the new inbound rule.



1. Click ok, then the rule will be created. Ensure that the port number is the same as the one that you have added in the virtual machine. The web socket used on both the server script and the index.html also needs to be the same.
2. That’s it! Open up the web app that we have published earlier and it should be able to communicate with the server that we successfully hosted on a virtual machine. Remember to disable or shut down the virtual machine whenever you aren’t using it to ensure that no additional charges would occur to your Azure account.

**Resources**

1. Tutorial on chat bot with sentiments and text analysis cognitive services (<https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/11/28/creating-my-first-chatbot-using-microsoft-cognitive-services-and-python/>)
2. Github repository for cognitive services SDK with Python (<https://github.com/Microsoft?utf8=%E2%9C%93&q=cognitive%20python&type=&language>=)
3. Github repository for this tutorial (<https://github.com/ThamMK/OpenWeatherLUIS>)
4. Github repository for simple web socket server

(<https://github.com/dpallot/simple-websocket-server>)