It is a step by step guide for creating a chatbot, in this case, an azure bot right from scratch and then deploying it to the cloud platform. This book takes a simple example of a weather query and tries to explain the concepts simply, extensively, and thoroughly to create a chatbot right from scratch and then its deployment to a cloud environment.

Happy Learning!

**A Chatbot with Microsoft Azure**

1. **Introduction:**

A chatbot is an application that can initiate and continue a conversation using auditory and/or textual methods as a human would do. A chatbot can be either a simple rule-based engine or an intelligent application leveraging Natural Language Understanding. Many organizations today have started using chatbots extensively. Chatbots are becoming famous as they are available 24\*7, provide a consistent customer experience, can handle several customers at a time, are cost-effective and hence, result in a better overall customer experience.

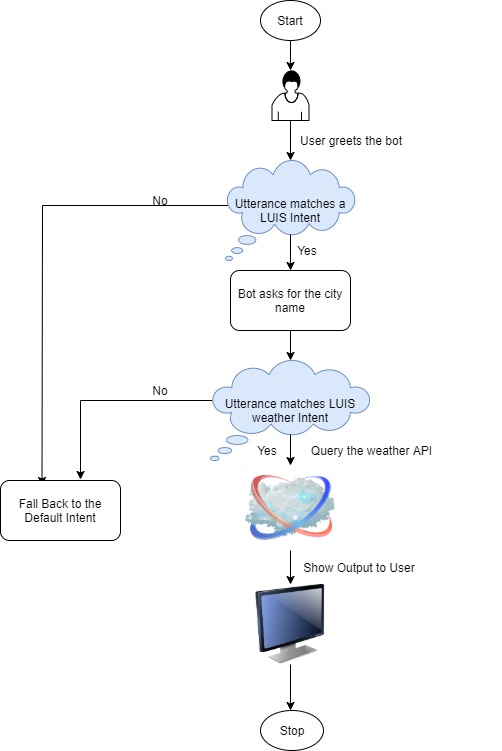
* 1. **Uses**
* Customer support
* Frequently Asked Questions
* Addressing Grievances
* Appointment Booking
* Automation of routine tasks
* Address a query

1. **Prerequisites**

The prerequisites for developing and understanding a chatbot using Microsoft Azure are:

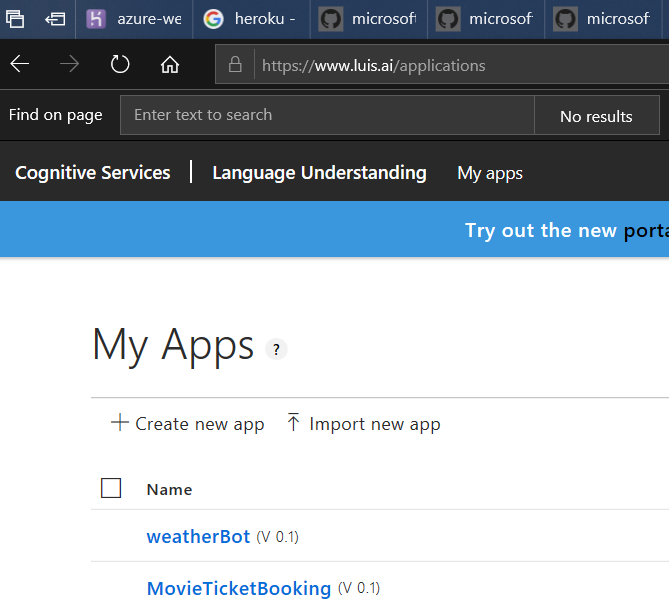
* An Azure account.
* A fundamental understanding of python and flask

1. **Application Architecture**

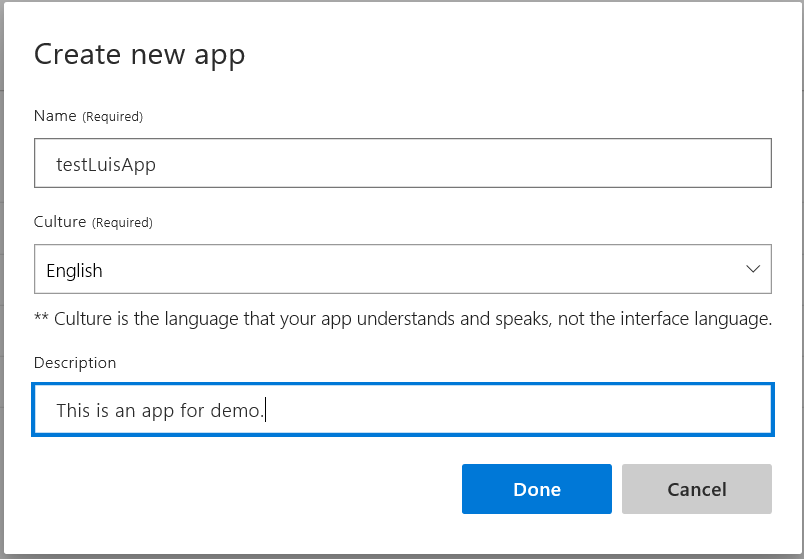
****

1. **Implementation**
   1. **Creating a LUIS App**

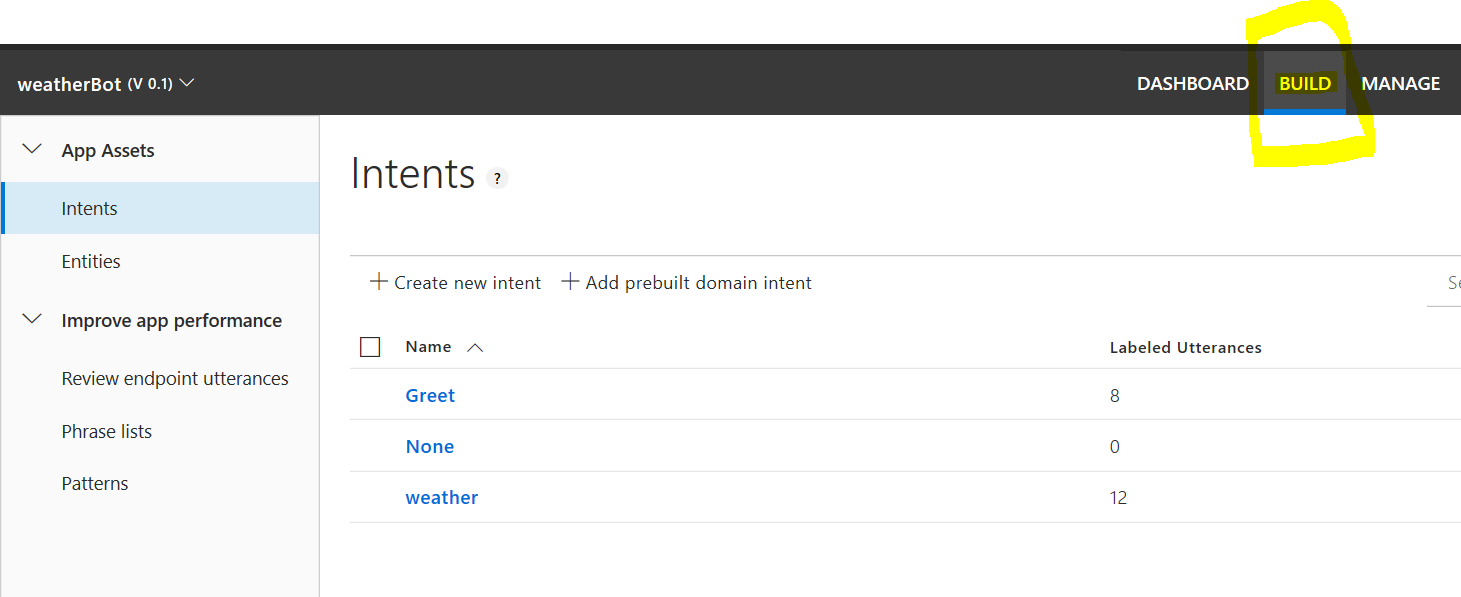
* Go to <https://www.luis.ai> and create an account if you already don’t have one.
* Click on ‘create new app’ to create a new app by as shown:



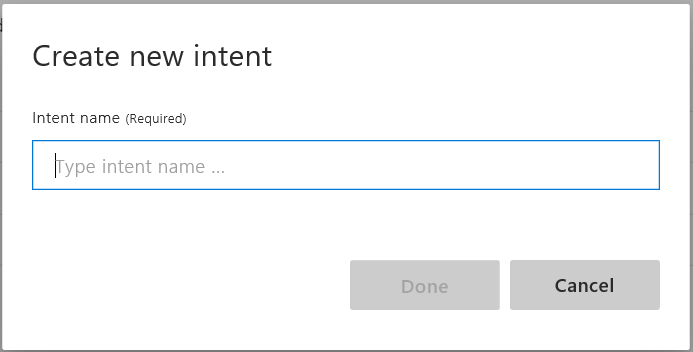
* Provide the following details and click ‘Done.



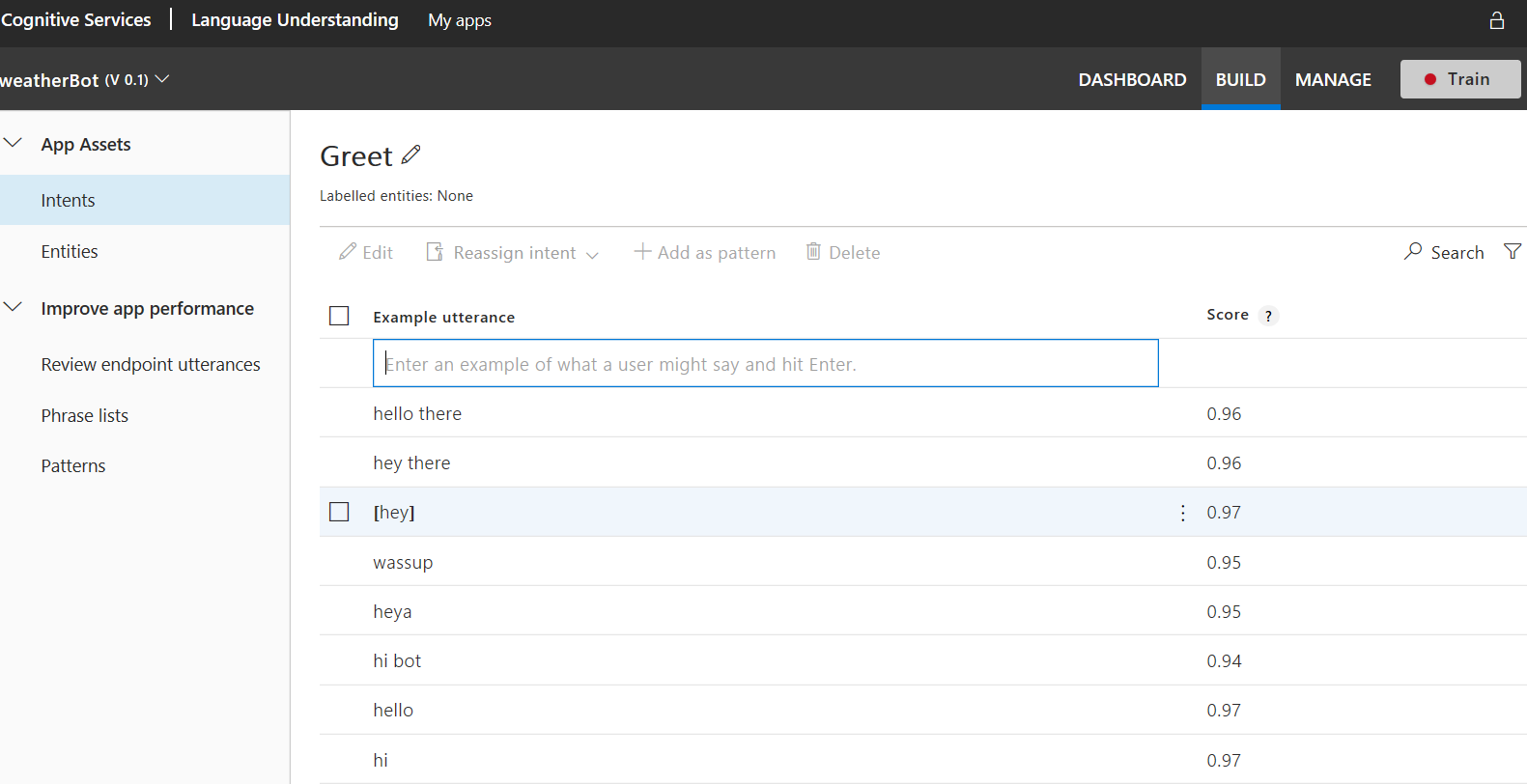
* Once created, open your app, select build, and click ‘Create new intent’ to create a new intent.



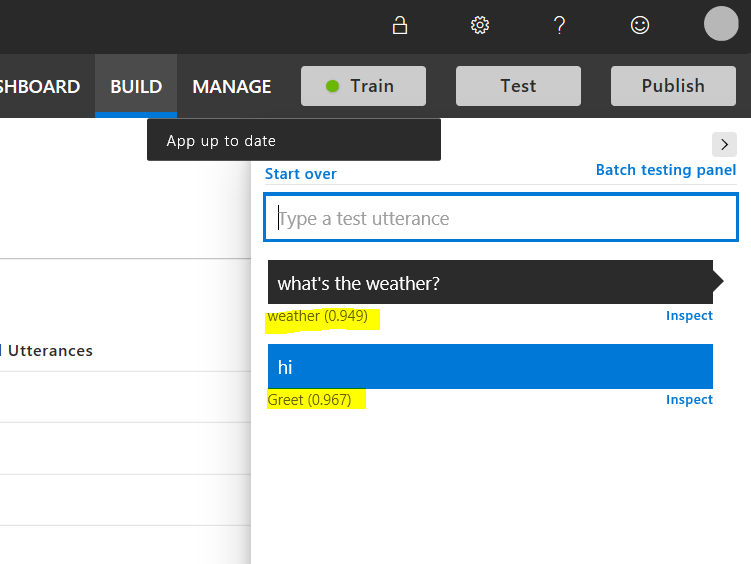
* Enter the name of the intent.



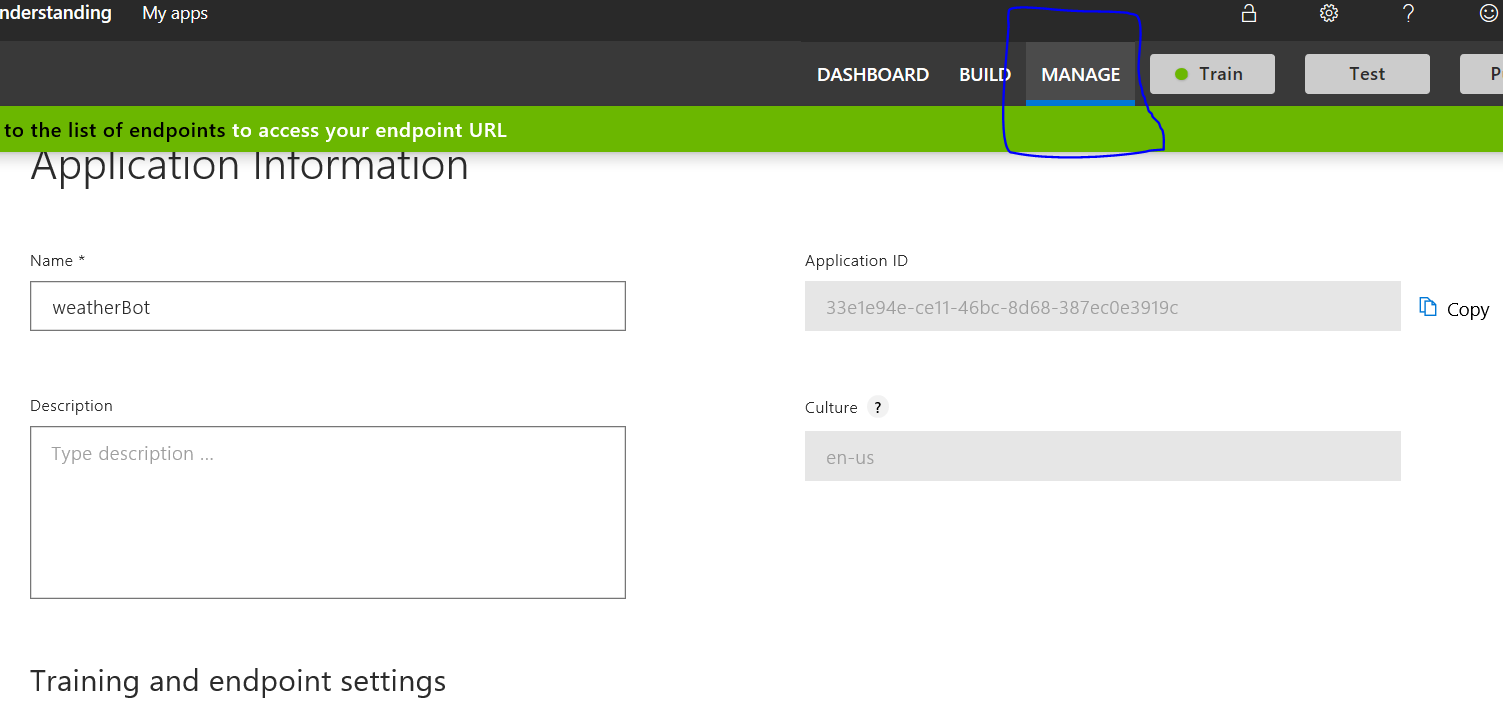
* Enter the user utterances and then click ‘train’ to train the LUIS app.



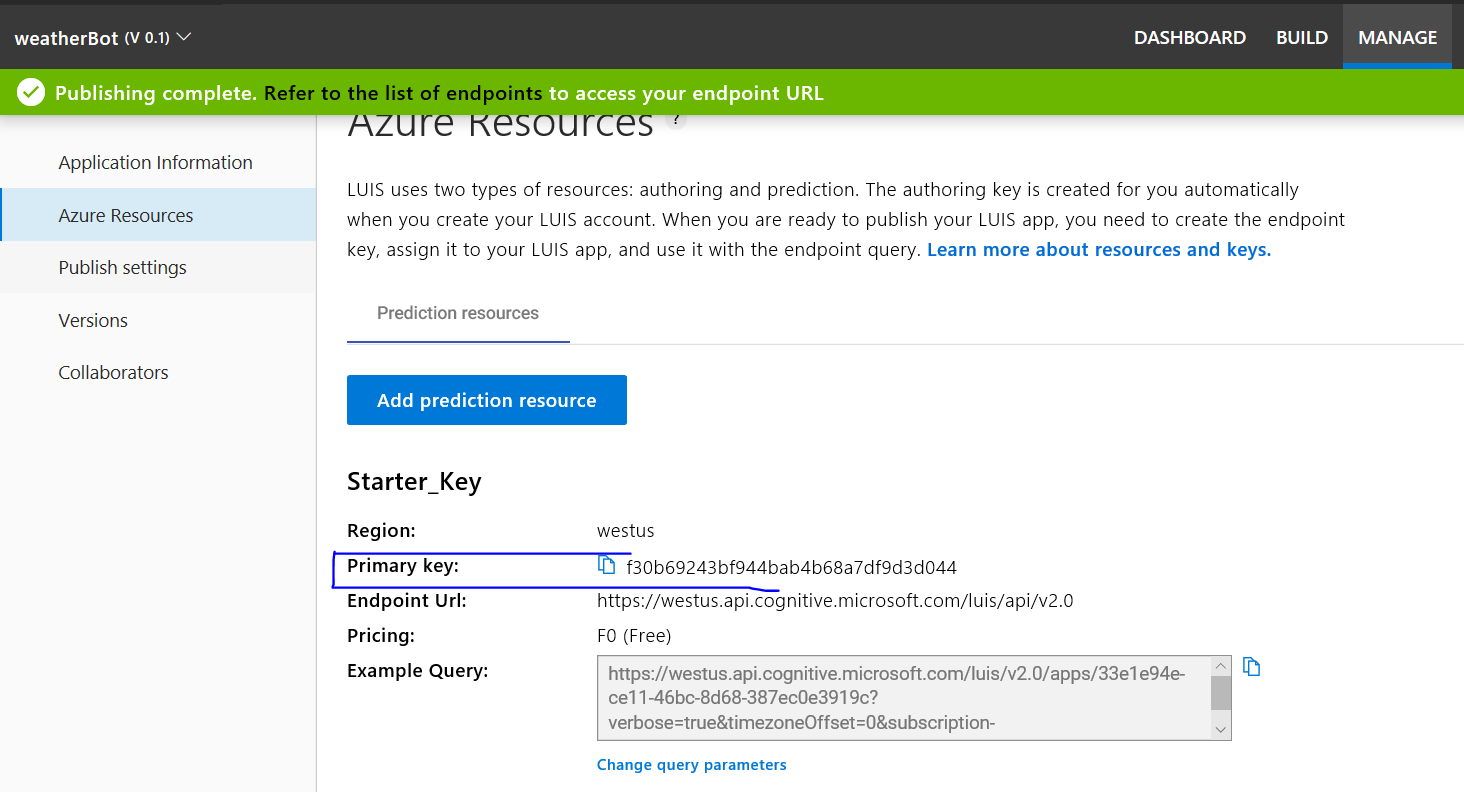
* Click ‘Test’ to test the intent and see the confidence of the app for various utterances.



* If satisfied with the test, click ‘publish’ and select the environment to make the LUIS app ready for consumption.
* Go to the *Manage* section of the published app and copy the *Application ID*. It will act as the ‘LUIS App ID.’



* Go to the *Azure Resources* section and copy the *Primary key*. It will serve as the LUIS API KEY.



* 1. **Create a python app**
     1. **Subscribing to the weather API**
* Go to <https://home.openweathermap.org/>, sign in/signup, and create an API Key for calling the *current weather data* API.
* This will act as the *weather\_api\_key.*
  + 1. **App creation and Integration with LUIS**
* Create a folder for your chatbot called azurePythonBot.
* Open the folder through pycharm.
* Create a file called app.py and put the following code.

from flask import Flask, request, Response  
from botbuilder.core import BotFrameworkAdapter, BotFrameworkAdapterSettings, ConversationState,MemoryStorage  
from botbuilder.schema import Activity  
import asyncio  
from luis.luisApp import LuisConnect  
import os  
from logger.logger import Log  
  
app = Flask(\_\_name\_\_)  
loop = asyncio.get\_event\_loop()  
  
bot\_settings = BotFrameworkAdapterSettings("", "")  
bot\_adapter = BotFrameworkAdapter(bot\_settings)  
  
#CON\_MEMORY = ConversationState(MemoryStorage())  
luis\_bot\_dialog = LuisConnect()  
  
  
@app.route("/api/messages", methods=["POST"])  
def messages():  
 if "application/json" in request.headers["content-type"]:  
 log=Log()  
 request\_body = request.json  
 user\_says = Activity().deserialize(request\_body)  
 log.write\_log(sessionID='session1',log\_message="user says: "+str(user\_says))  
 authorization\_header = (request.headers["Authorization"] if "Authorization" in request.headers else "")

async def call\_user\_fun(turncontext):  
 await luis\_bot\_dialog.on\_turn(turncontext)  
  
 task = loop.create\_task(  
 bot\_adapter.process\_activity(user\_says, authorization\_header, call\_user\_fun)  
 )  
 loop.run\_until\_complete(task)  
 return ""  
 else:  
 return Response(status=406) # status for Not Acceptable  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 #app.run(port= 3978)  
 app.run()

* Create a folder logger and create logger.py inside as shown below:

from datetime import datetime  
class Log:  
 def \_\_init\_\_(self):  
 pass  
  
 def write\_log(self, sessionID, log\_message):  
 self.file\_object = open("conversationLogs/"+sessionID+".txt", 'a+')  
 self.now = datetime.now()  
 self.date = self.now.date()  
 self.current\_time = self.now.strftime("%H:%M:%S")  
 self.file\_object.write(  
 str(self.date) + "/" + str(self.current\_time) + "\t\t" + log\_message + "\n")  
 self.file\_object.close()

* Create a folder luis and create luisApp.py inside as shown below:

from botbuilder.core import TurnContext,ActivityHandler  
from botbuilder.ai.luis import LuisApplication,LuisPredictionOptions,LuisRecognizer  
import json  
from weather.weatherApp import WeatherInformation  
from config.config\_reader import ConfigReader  
from logger.logger import Log  
class LuisConnect(ActivityHandler):  
 def \_\_init\_\_(self):  
 self.config\_reader = ConfigReader()  
 self.configuration = self.config\_reader.read\_config()  
 self.luis\_app\_id=self.configuration['LUIS\_APP\_ID']  
 self.luis\_endpoint\_key = self.configuration['LUIS\_ENDPOINT\_KEY']  
 self.luis\_endpoint = self.configuration['LUIS\_ENDPOINT']  
 self.luis\_app = LuisApplication(self.luis\_app\_id,self.luis\_endpoint\_key,self.luis\_endpoint)  
 self.luis\_options = LuisPredictionOptions(include\_all\_intents=True,include\_instance\_data=True)  
 self.luis\_recognizer = LuisRecognizer(application=self.luis\_app,prediction\_options=self.luis\_options,include\_api\_results=True)  
 self.log=Log()  
   
  
 async def on\_message\_activity(self,turn\_context:TurnContext):  
 weather\_info=WeatherInformation()  
 luis\_result = await self.luis\_recognizer.recognize(turn\_context)  
 result = luis\_result.properties["luisResult"]  
 json\_str = json.loads((str(result.entities[0])).replace("'", "\""))  
 weather=weather\_info.get\_weather\_info(json\_str.get('entity'))  
 self.log.write\_log(sessionID='session1',log\_message="Bot Says: "+str(weather))  
 await turn\_context.send\_activity(f"{weather}")

* Create a folder weather and create weatherApp.py inside it as shown below:

import pyowm  
from config.config\_reader import ConfigReader  
  
class WeatherInformation():  
 def \_\_init\_\_(self):  
 self.config\_reader = ConfigReader()  
 self.configuration = self.config\_reader.read\_config()  
 self.owmapikey = self.configuration['WEATHER\_API\_KEY']  
 self.owm = pyowm.OWM(self.owmapikey)  
  
 def get\_weather\_info(self,city):  
 self.city=city  
  
 observation = self.owm.weather\_at\_place(city)  
 w = observation.get\_weather()  
 latlon\_res = observation.get\_location()  
 lat = str(latlon\_res.get\_lat())  
 lon = str(latlon\_res.get\_lon())  
  
 wind\_res = w.get\_wind()  
 wind\_speed = str(wind\_res.get('speed'))  
  
 humidity = str(w.get\_humidity())  
  
 celsius\_result = w.get\_temperature('celsius')  
 temp\_min\_celsius = str(celsius\_result.get('temp\_min'))  
 temp\_max\_celsius = str(celsius\_result.get('temp\_max'))  
  
 fahrenheit\_result = w.get\_temperature('fahrenheit')  
 temp\_min\_fahrenheit = str(fahrenheit\_result.get('temp\_min'))  
 temp\_max\_fahrenheit = str(fahrenheit\_result.get('temp\_max'))  
 self.bot\_says = "Today the weather in " + city +".\n Maximum Temperature :"+temp\_max\_celsius+ " Degree Celsius"+".\n Minimum Temperature :"+temp\_min\_celsius+ " Degree Celsius" +": \n" + "Humidity :" + humidity + "%"  
 return self.bot\_says

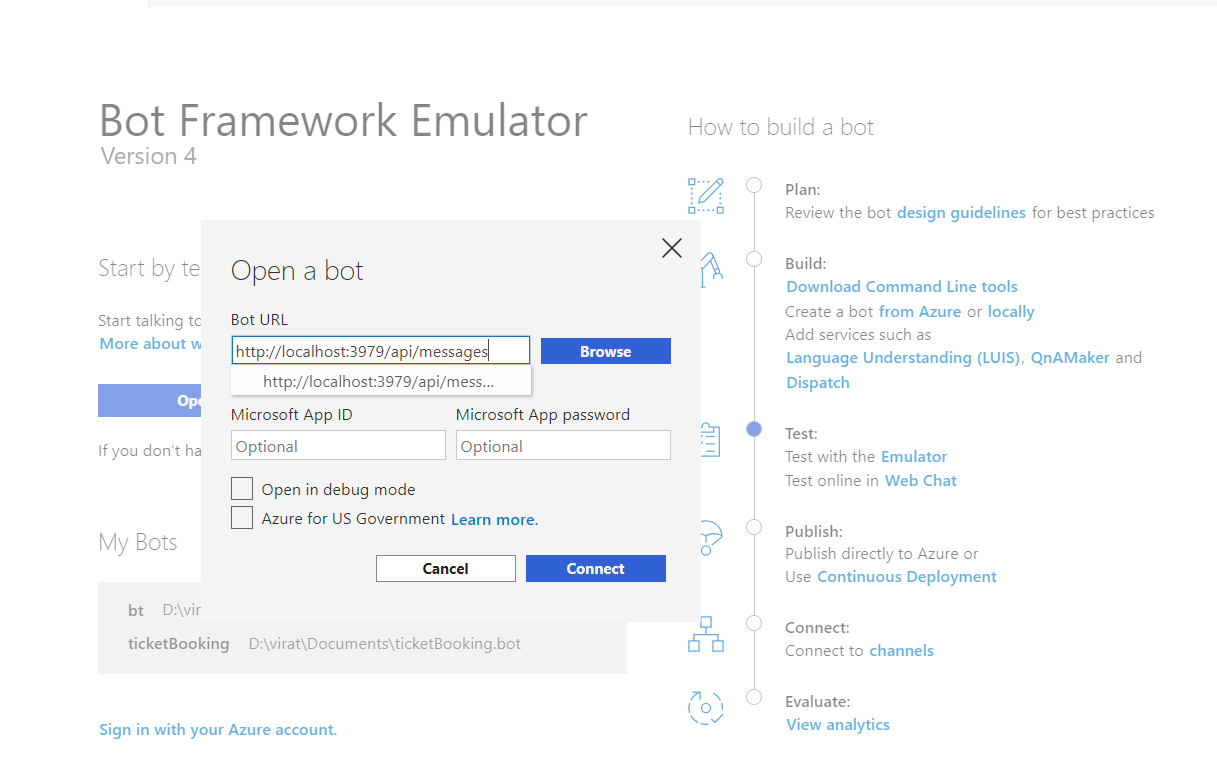
* Create a file config.ini and put the following details:

[DEFAULT]  
WEATHER\_API\_KEY=your weather api key  
LUIS\_APP\_ID=your LUIS app id  
LUIS\_ENDPOINT\_KEY=your LUIS endpoint key   
LUIS\_ENDPOINT=your LUIS endpoint

* Create a folder config and and create config\_reader.py inside as shown below:

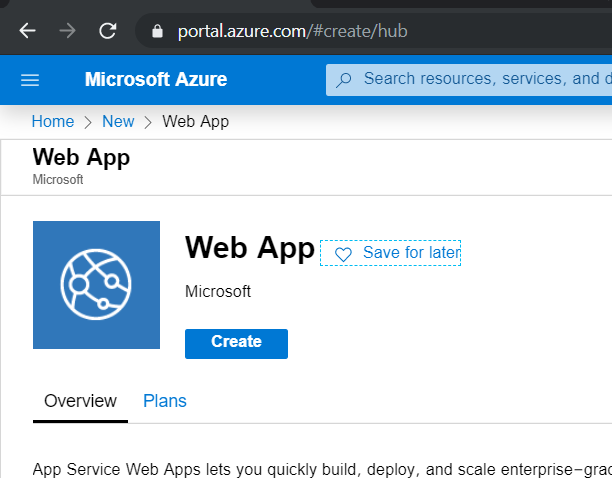
import configparser  
  
class ConfigReader:  
 def \_\_init\_\_(self):  
 self.filename = 'config.ini'  
 def read\_config(self):  
 self.config = configparser.ConfigParser()  
 self.config.read(self.filename)  
 self.configuration=self.config['DEFAULT']  
 return self.configuration

* 1. **Install Bot Emulator and test**
* Go to <https://github.com/Microsoft/BotFramework-Emulator/releases> and download the Bot Emulator setup file based on your computer.
* Once the download is completed, double click the installation file and it’ll automatically install the Bot Emulator.
* Run the bot emulator and connect to the already running bot file(app.py) as shown:

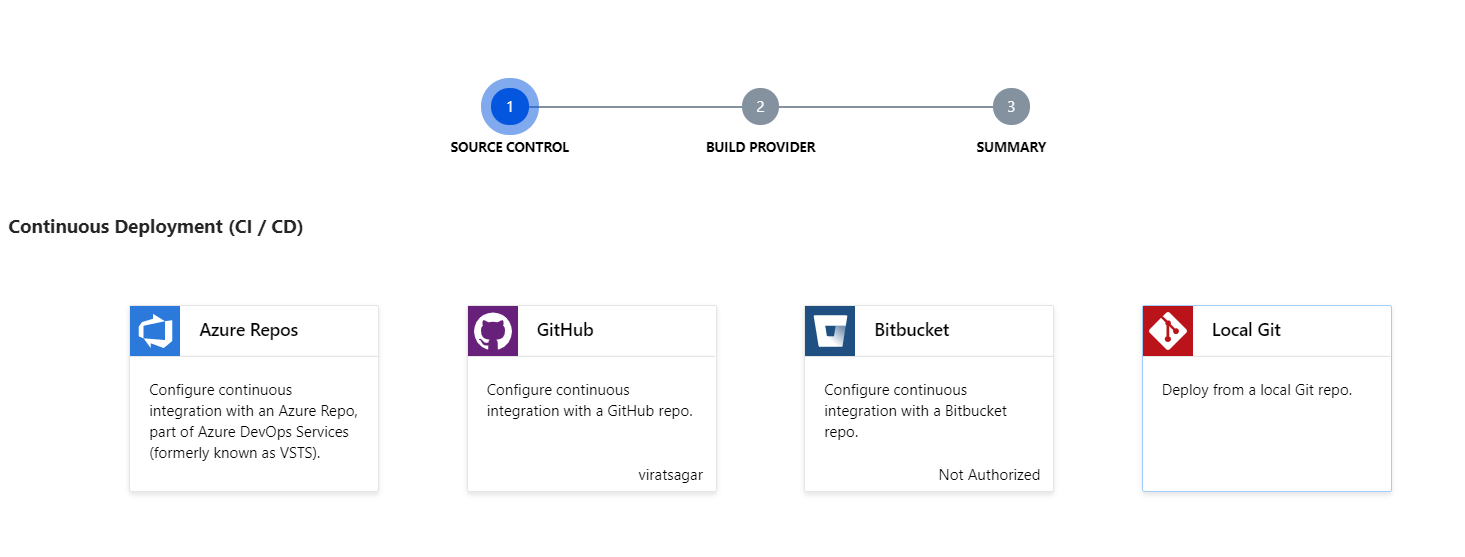


Give the port number on which your app is running.

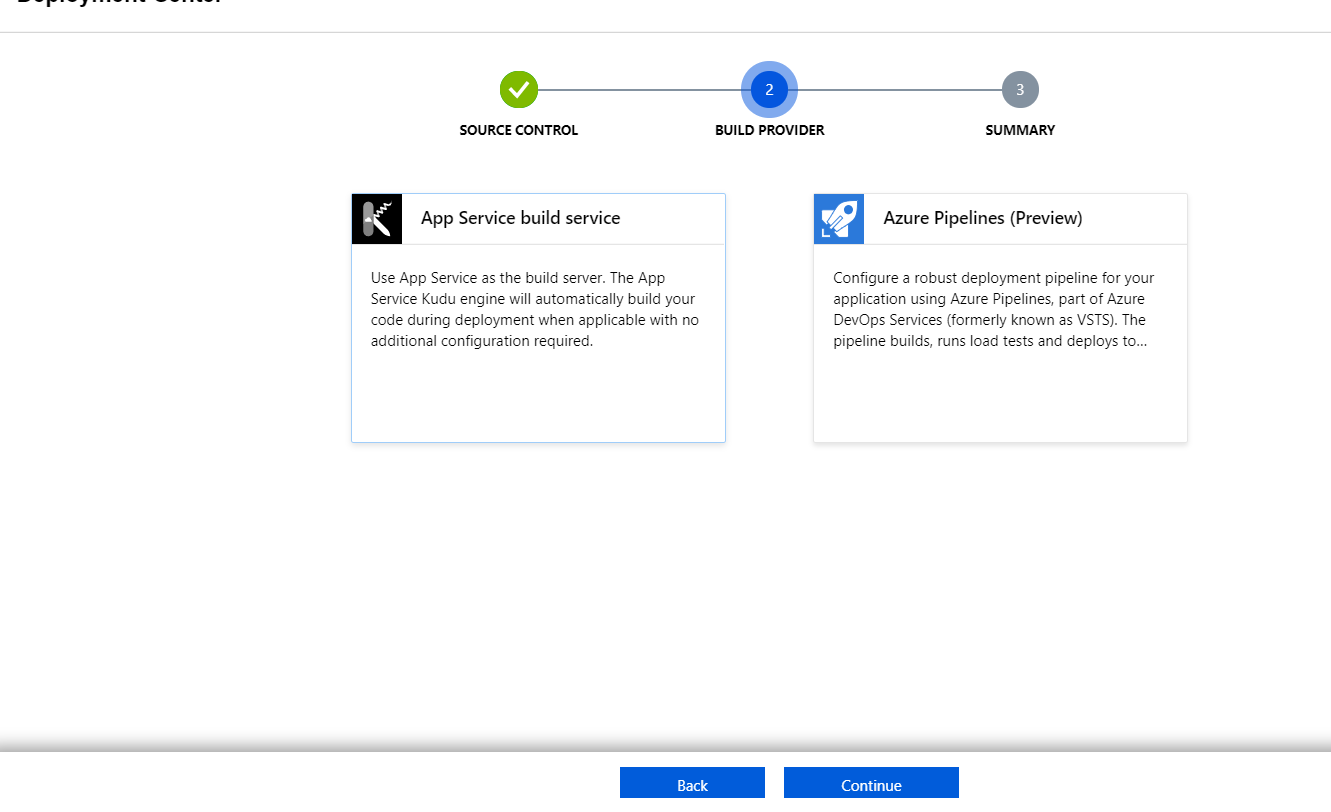
* 1. **Deploy to azure**
* Go to the Azure account and create a web app.



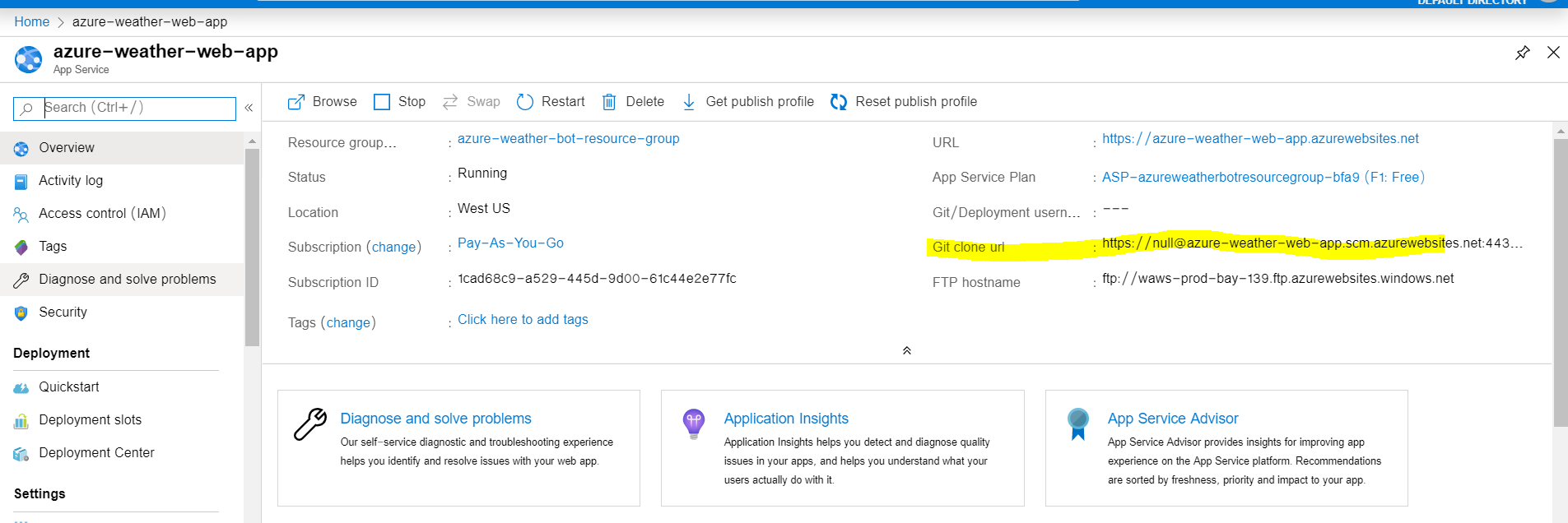
* Provide the app name, resource group(create new if necessary), runtime stack(Python <version>), region, select the 1 GB size, which is free to use. Click *Review+create* to create the web app.
* Once the deployment is completed, open the app and go to the ‘Deployment Center’ option. Select ‘local git’ for source control and click continue.



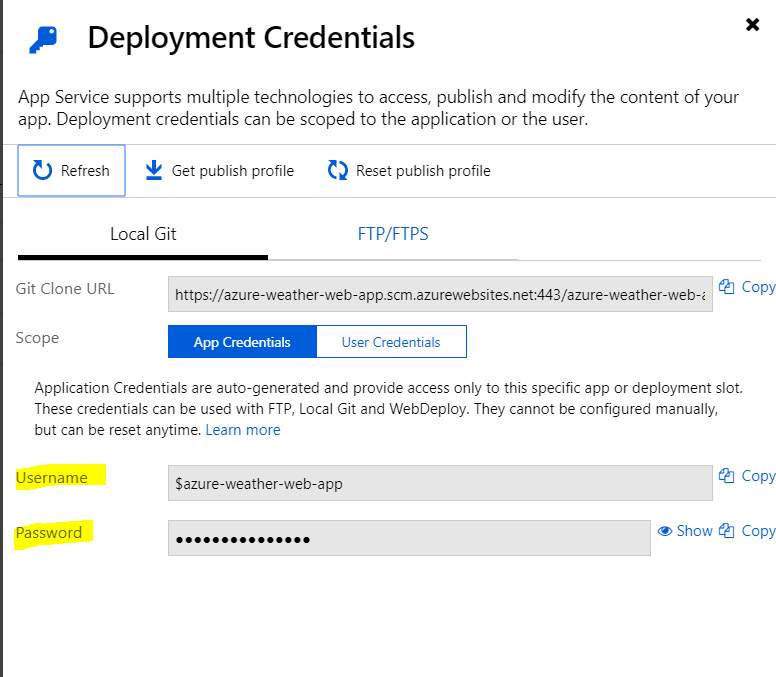
* Select the kudo ‘App service build provider’ as the build provider and click continue.



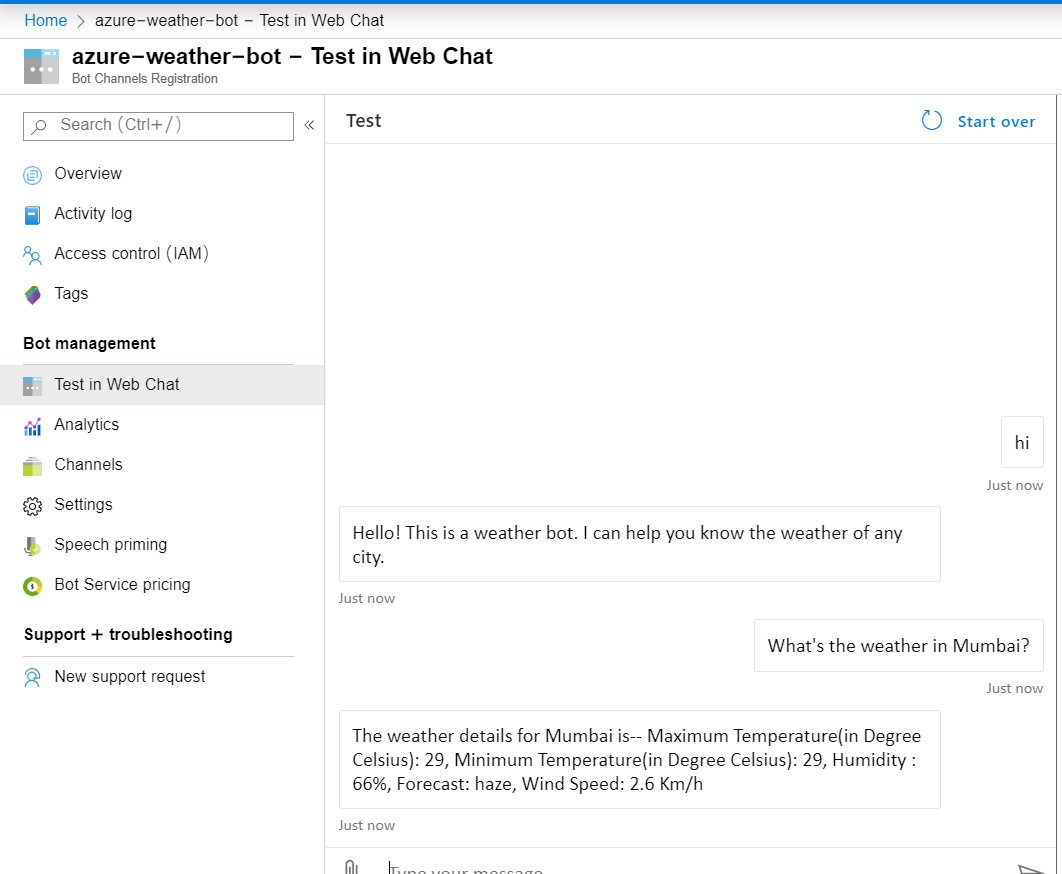
* Click ‘Finish’ to complete the setup.
* Go to the overview section of the app, and the Git link now will be visible.



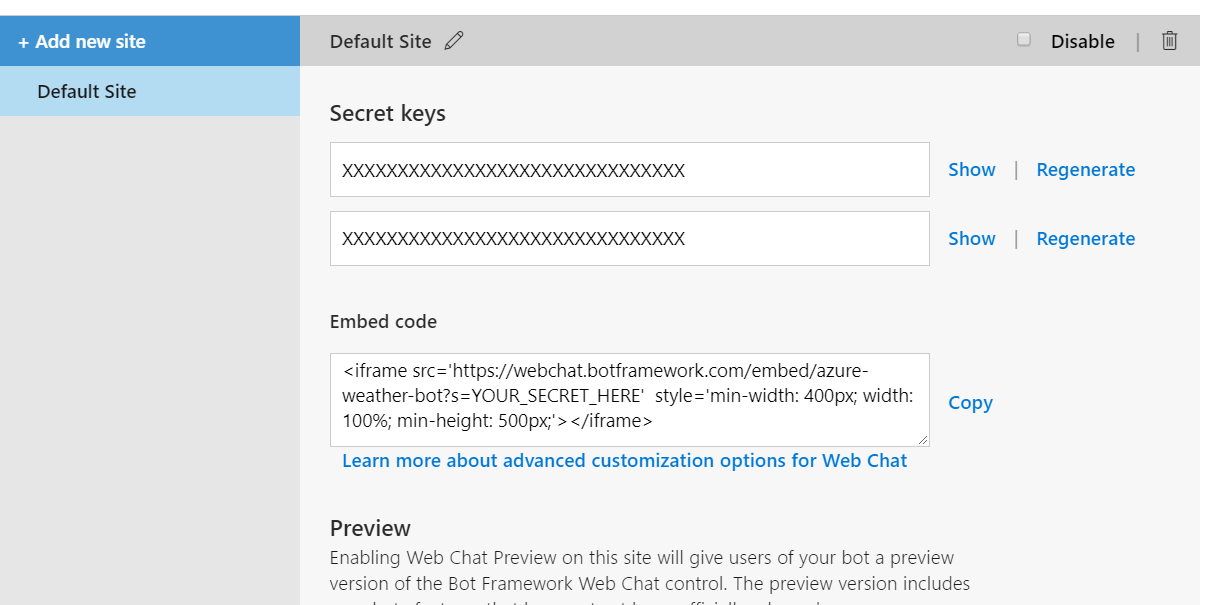
* Open a command prompt and navigate to your project folder.
* Run git init to initialize an empty git repository
* Create a new remote git alias using the command: git remote add <alias> <git clone url>
* Use git add . to add all the files to the local git repository.
* Use git commit –m “First Commit” to commit the code to the git repo.
* Push the code to the remote repo using git push <alias> master –f
* This prompts for a username and password. Go to the ‘Deployment Credentials’ section and copy the username and password to enter in the prompt.



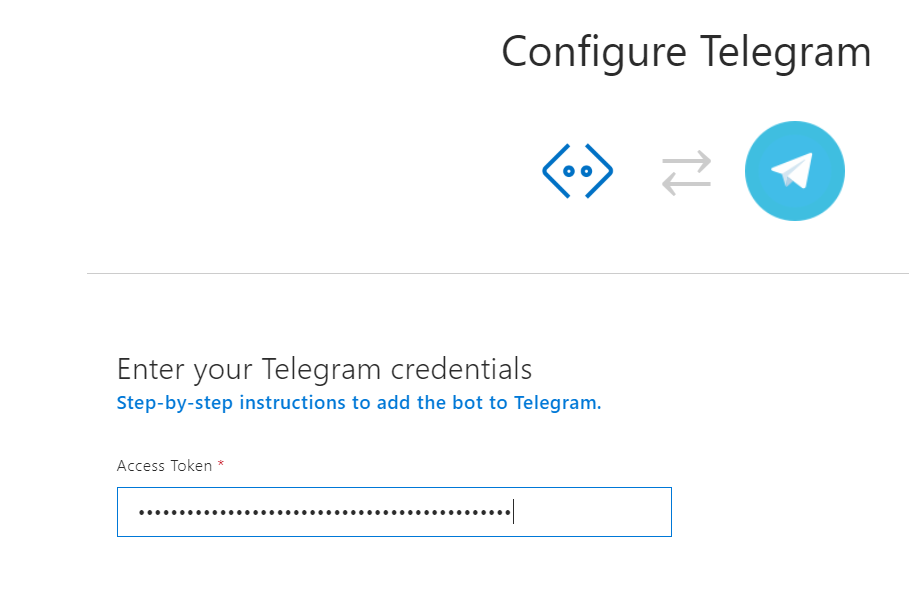
* Once the credentials are correctly entered, the app deployment to azure is completed.
  1. **Create bot channel Registration**
* Now in the Azure portal, create a bot channel registration.
* Provide the bot handle, resource group and other fields.
* For Message endpoint, provide: <URL from the web app created above>api/messages. Click Create to create the bot.
* Once your web channel registration gets done, open the bot and then click ‘test in web chat.’ If the chat works fine, our deployment is a success.



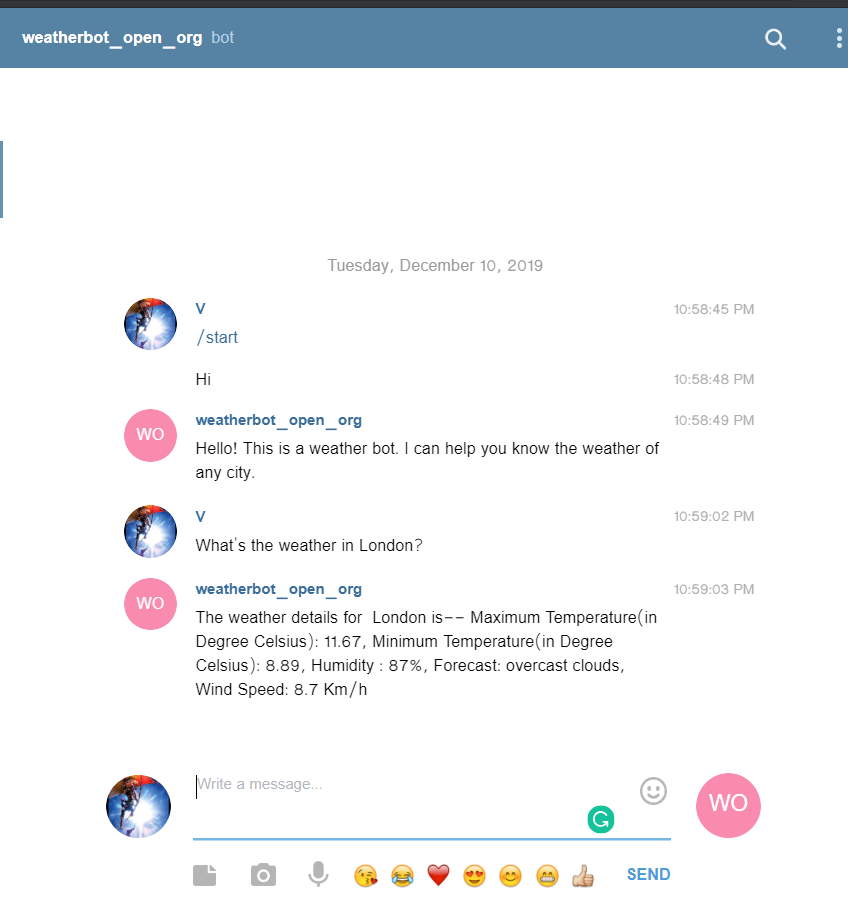
* 1. **Deployment**
* Go to the channels section of your bot.
* The bot can be deployed as an embedding to an existing HTML page by selecting the get bot embedded code option



* + - 1. **Telegram Deployment**
* Open the telegram app and search for botfather(it is an inbuilt bot used to create other bots)
* Start a conversation with botfather and enter /newbot to create a newbot.
* Give a name to your bot
* Give a username to your bot, which must end in \_bot.
* This generates an access token. Enter that access token after clicking the telegram channel in your bot app and click save.



* Now, search the username of the bot in telegram and start conversation with your bot.



Thank You!