iNeuron

Table of Contents

Preface

This book is intended to help all the data scientists out there. 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 consistent customer experience, can handle several customers at a time, are cost-effective and hence, result in better overall customer experience.

1.1 Uses

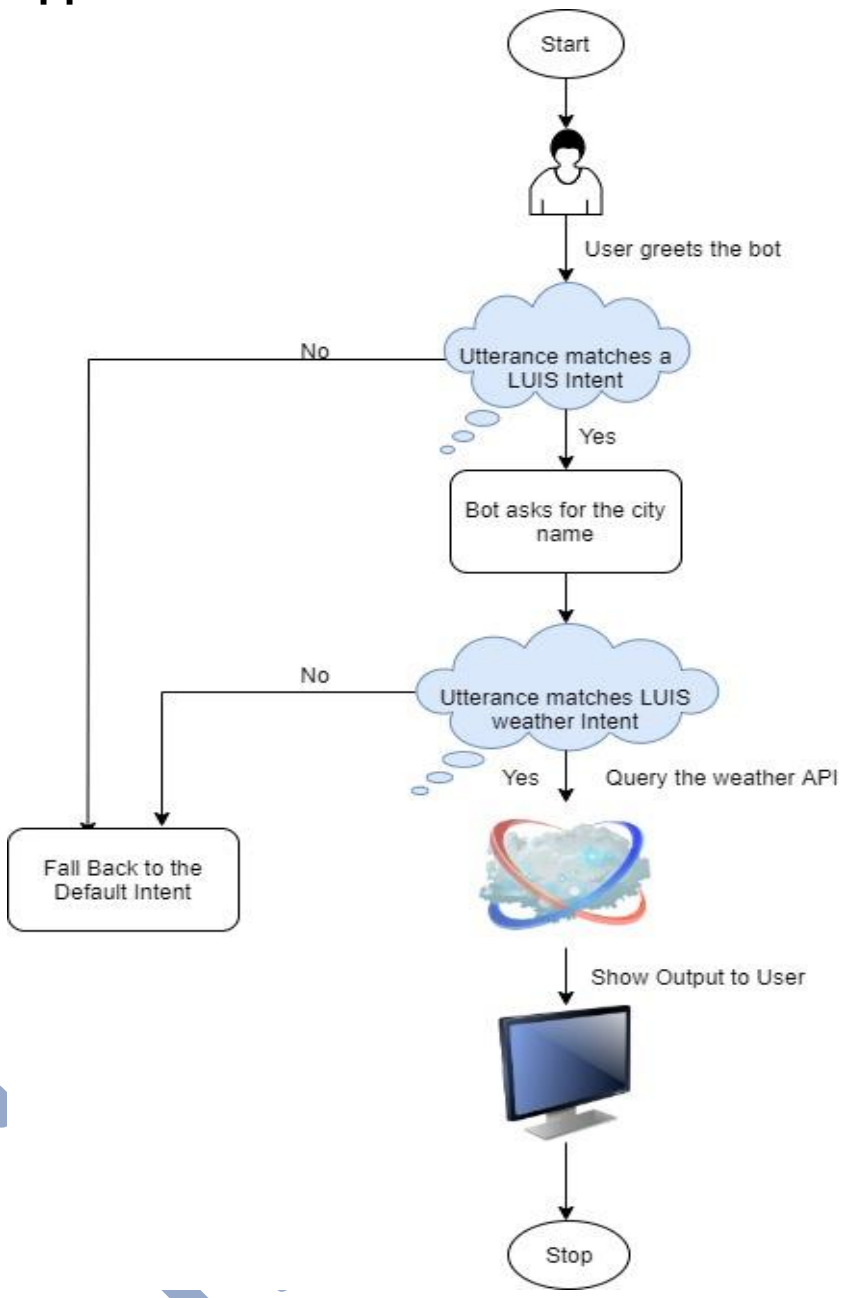
- Customer support
- Addressing Queries
- Frequently Asked Questions
- Addressing Grievances
- Appointment Booking
- Automation of routine tasks

2. Prerequisites

The prerequisites for developing and understanding a chatbot are:

- A basic understanding of Language model and conversation flows.
- An Azure account.
- A fundamental understanding of node.js.

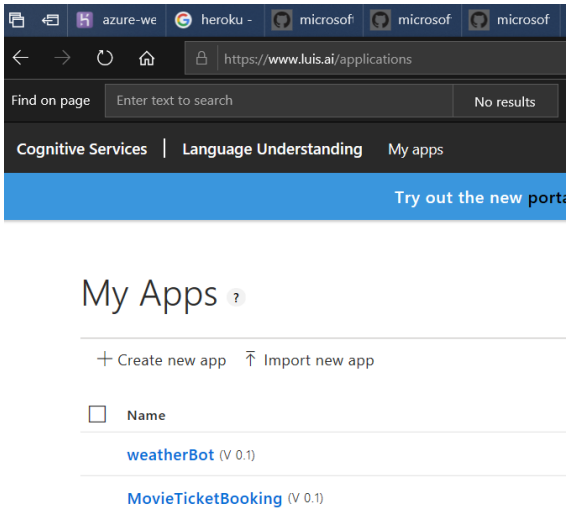
3. Application Architecture



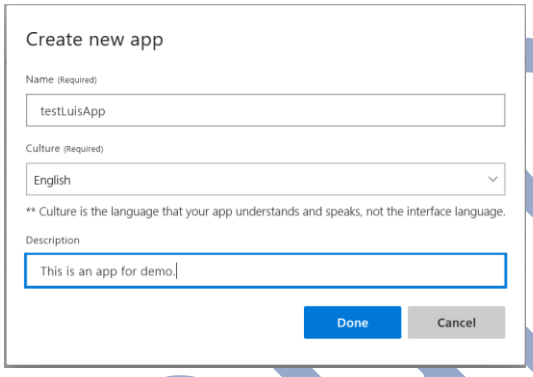
4. Implementation

4.1 Creating a LUIS App

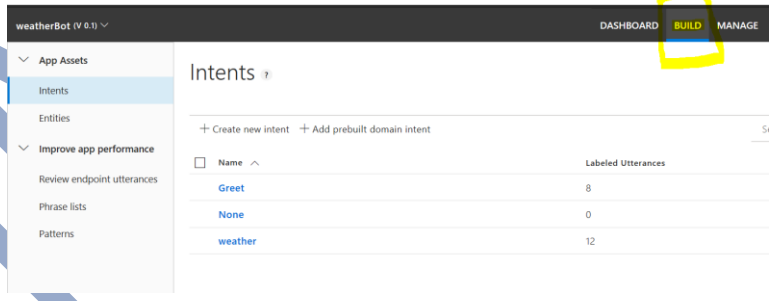
- Go to https://www.luis.ai and create an account if you already don't have one.
- Click on 'create new app' to create a new app by as shown:



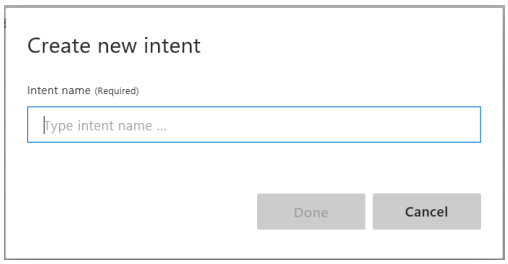
- Provide the following details and click ‘Done.



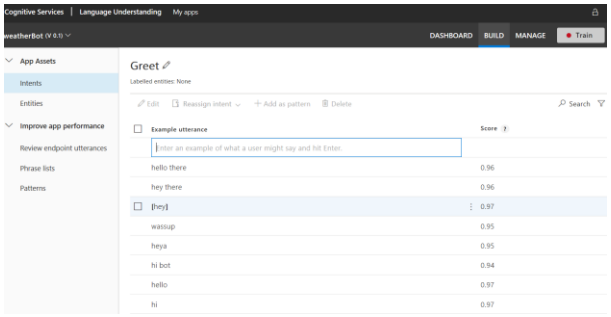
- Once created, open your app, select build, and click ‘Create new intent’ to create a new intent.



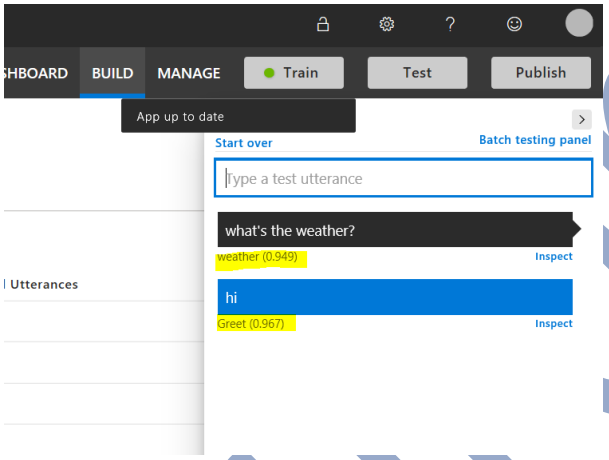
- Enter the name of the intent.



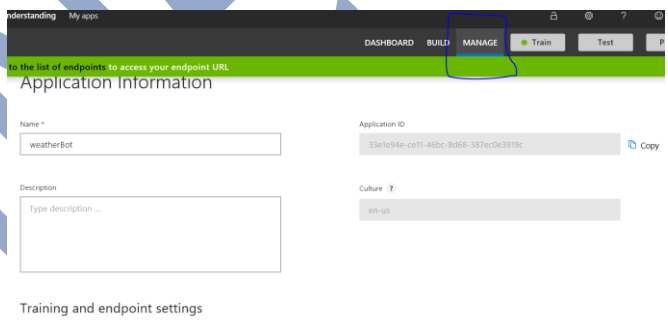
- Enter the user utterances and then click ‘train’ to train the LUIS app.



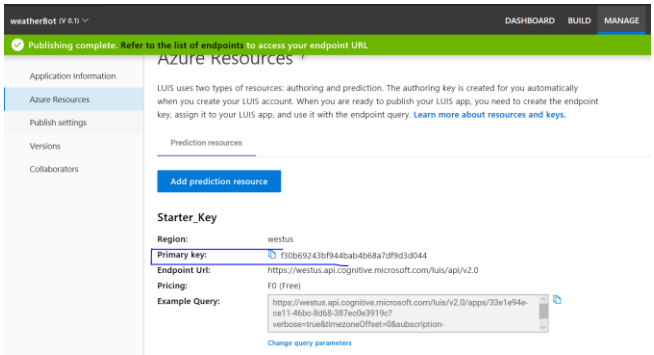
- Click 'Test' to test the intent and see the confidence of the app for various utterances.



- 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.

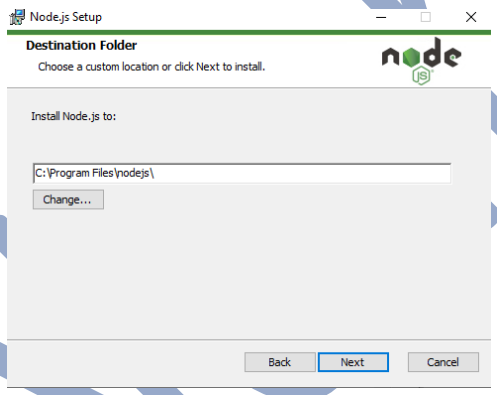


4.2 Create a node.js app

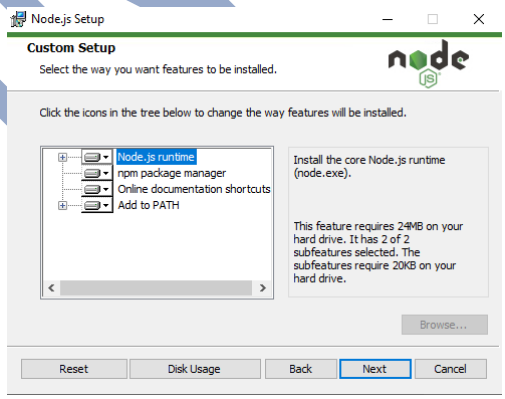
4.2.1 Node Installation

- Go to https://nodejs.org/en/download/ and download the installer package based on your operating system.
- Double click the installation file, and the installation shall start.
- Click next and accept the terms.

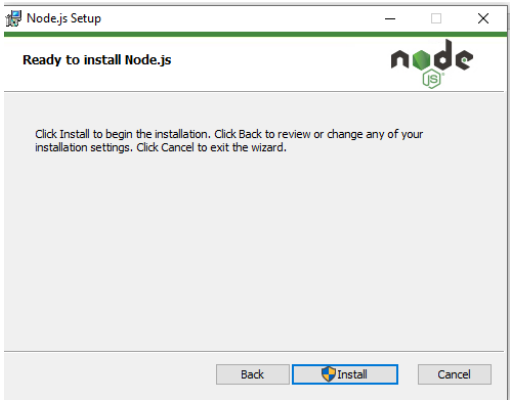
- Select the installation directory and then click next.



- Select the features to install and click next.



- Click Install and then click Finish to complete the installation.



4.2.2 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*.

4.2.3 App creation and Integration with LUIS

- Create a folder for your chatbot called azure_weather_bot.
- Open a command prompt window and navigate to your project directory(azure_weather_bot).
- Run the command 'npm init' to initialize the node context. It will create the necessary files for proceeding with our node application.
- Run the commands npm install botbuilder --save, npm install restify --save and npm install request --save to install the required packages and add them to the dependencies in *package.json* file.
- Once done, open the project folder using any text editor.
- Open the app.js file(create if not present) and enter the following code snippet.

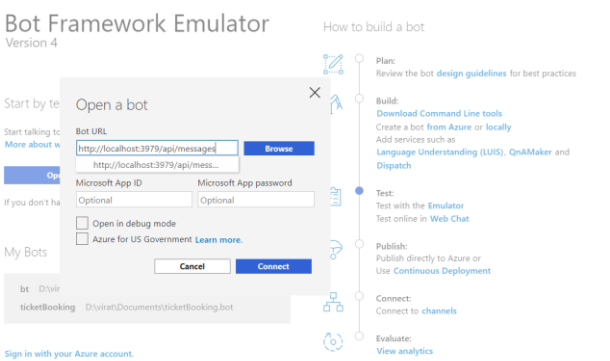
```
var builder = require ('botbuilder');
var restify = require('restify');
const request = require('request');

var server =restify.createServer();
server.listen( process.env.PORT || 3979, function () {
    console.log('%s listening to %s', server.name, server.url);
});


var inMemoryStorage = new builder.MemoryBotStorage();
var connector = new builder.ChatConnector();
server.post('/api/messages',connector.listen());
var bot= new
builder.UniversalBot(connector).set('storage',inMemoryStorage);

var luisapApId="<your LUIS APP ID>";
var luisApiKey="<your LUIS API Key>";
var luisApiHostname="westus.api.cognitive.microsoft.com";
var weather_api_Key = '<your API Key>';
const
luisModelUrl='https://'+luisApiHostname+'/luis/v2.0/apps/'+luisapApId+'?s
ubscription-key='+luisApiKey;
```

```
var recognizer= new builder.LuisRecognizer(luisModelUrl);
var intents= new builder.IntentDialog({
    recognizers : [recognizer]
});

bot.dialog('/',intents);
intents.matches('Greet',(session,args,nest)=>{
    session.send("Hello! This is a weather bot. I can help you know the
weather of any city.");
});
intents.matches('weather',[(session,args,nest)=> {
    var city = args.entities.filter(e => e.type == 'city');
    if (city.length > 0) {
        session.userData.city = city[0].entity;

        //session.userData.city ='portland';
        let url =
`http://api.openweathermap.org/data/2.5/weather?q=${session.userData.city
}&units=metric&appid=${weather_api_Key}`
        request(url, function (err, response, body) {
            if (err) {
                console.log('error:', error);
            } else {
                let weather = JSON.parse(body);
                console.log('weather is ',weather);
                try {
                    let message = `The weather details for
${weather.name} is-- Maximum Temperature(in Degree Celsius):
${weather.main.temp_max}, Minimum Temperature(in Degree Celsius):
${weather.main.temp_min}, Humidity : ${weather.main.humidity}%, Forecast:
${weather.weather[0].description}, Wind Speed: ${weather.wind.speed}
Km/h`;
                    session.send(message);
                }
                catch (e) {
                  console.log('output for weather is ',weather);
                  console.log('The exception message is: ',e);
                  let message = `Sorry, we could not process your
request`;
                session.send(message);
                }
            }
        });
    } else {
        delete session.userData.city;
    }
},
    (session, args, next) => {
    if (!session.userData.city) {
        session.beginDialog('askNameOfCity');
    } else {
        next();
    }
}]);
bot.dialog('askNameOfCity', [(session, args, next) => {
    builder.Prompts.text(session, 'Which city weather are you interested
in?')
}, (session, results) => {
    session.userData.city = results.response.entity;
    console.log('this is after city: ',session.userData.city);
    onsole.log('this is after centity ',results);
    session.endDialogWithResult(results);
}]);
```

- Run this using node app .js in a command prompt, and if it runs, the code is ready to be tested.

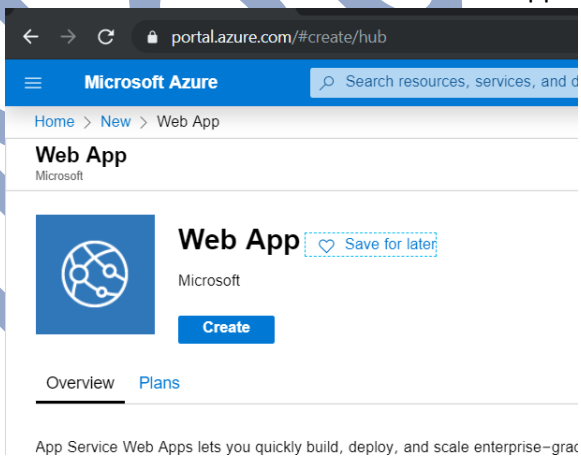
4.3 Install Bot Emulator and test

- 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 Javascript file(app.js) as shown:

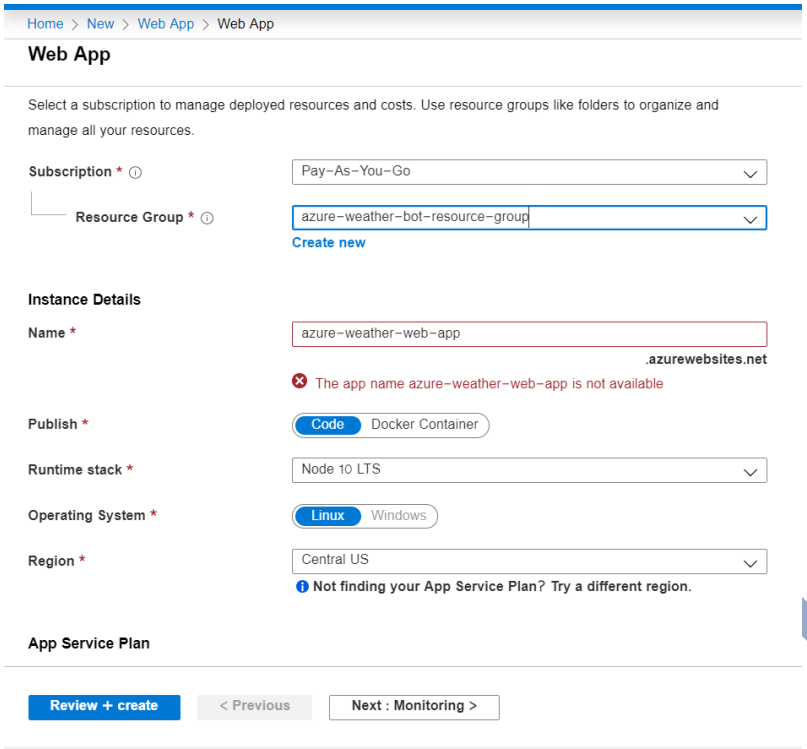


4.4 Deploy to azure

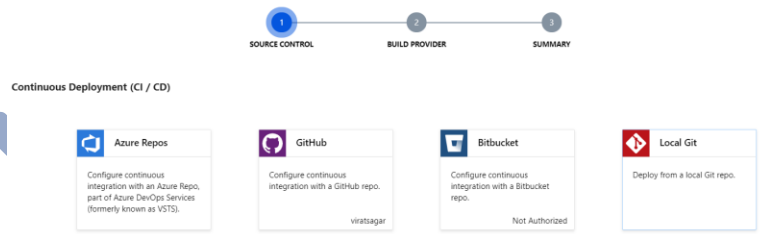
- Go to the Azure account and create a web app.



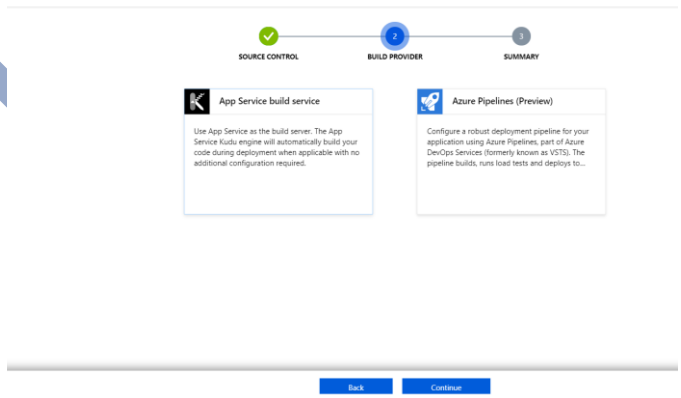
- Provide the app name, resource group(create new if necessary), runtime stack(node 10 LTS), region, select the 1 GB size, which is free to use. Click *Review+create* to create the web app.



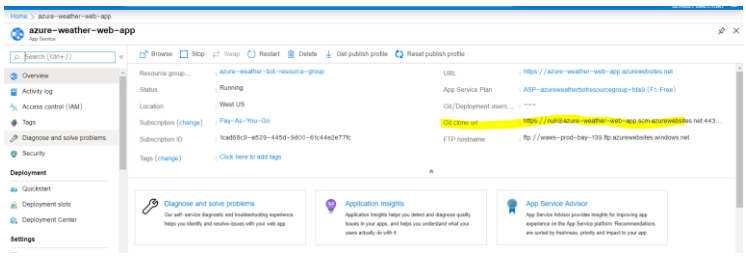
- Once the deployment is completed, open the app and go to the 'Deployment Center' option. Select 'local git' for source control and click continue.



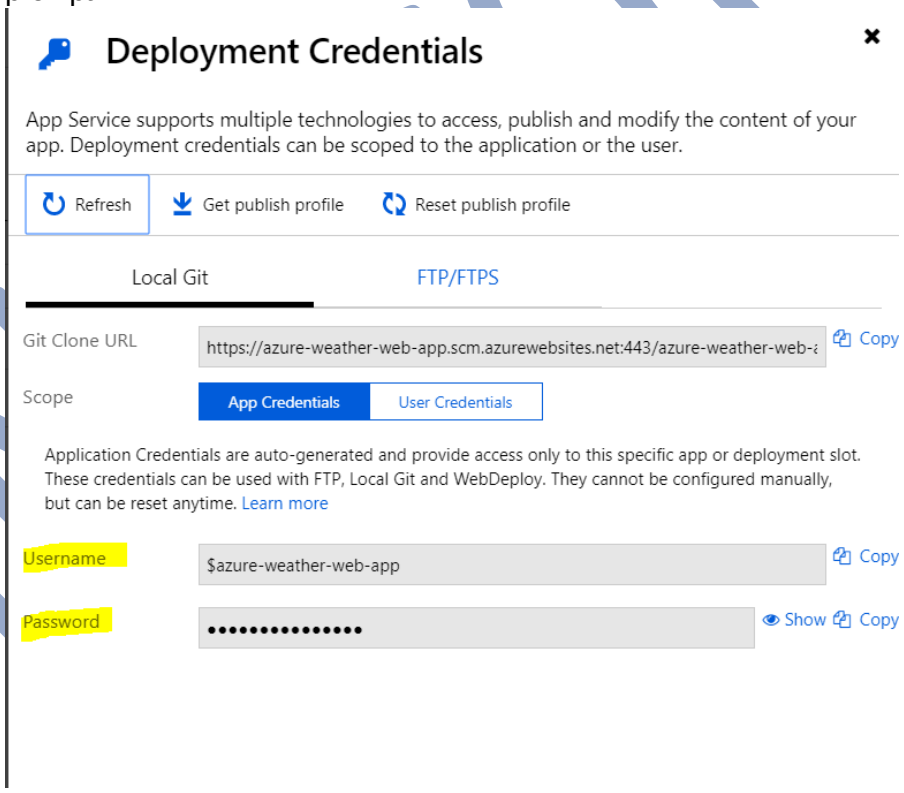
- Select the kudo 'App service build provider' as the build provider and click continue.



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



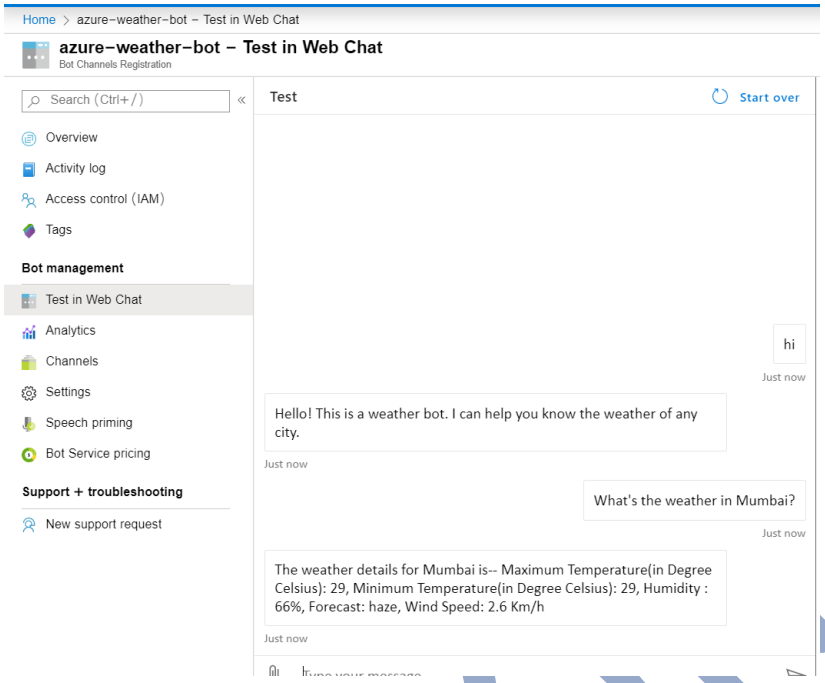
- Open a command prompt and navigate to your project folder.
- Run git init to initialize an empty git repository
- Create a new remote git alias using the command: git remote add <alias> <git clone url>
- Use git add . to add all the files to the local git repository.
- Use git commit –m “First Commit” 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.

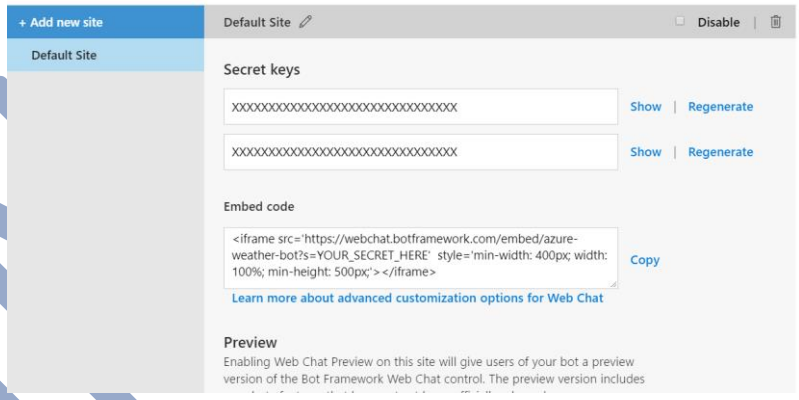
4.5 Create bot channel Registration

- 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.



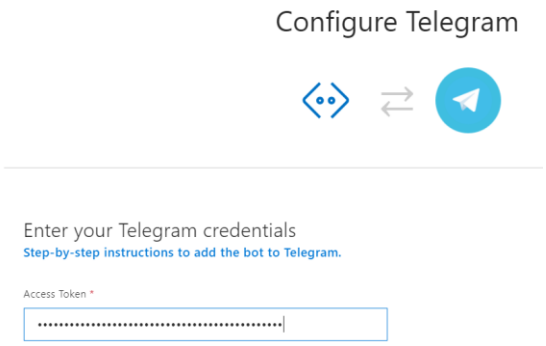
4.6 Deployment

- 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

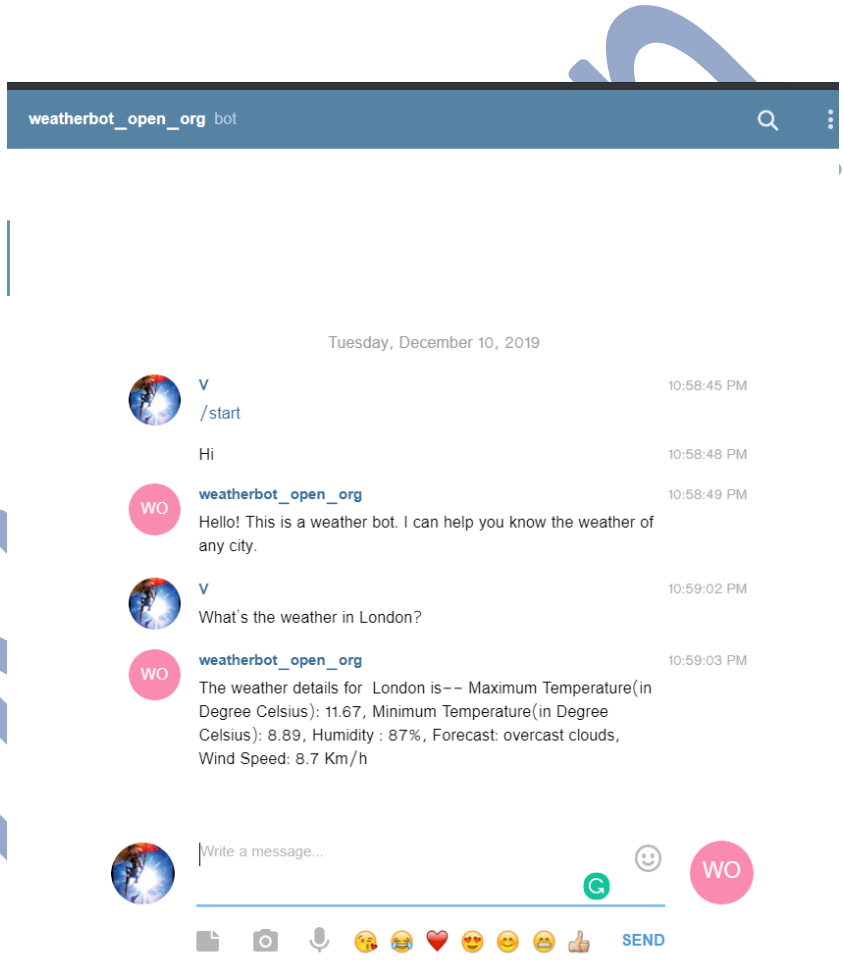


4.6.1.1 Telegram Deployment

- 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!