# **Prerequisites**

* **Azure Subscription**
* **Azure Data Science VM (Windows 2016) –** [**https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/overview**](https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/overview)
* [**Visual Studio Code**](https://www.visualstudio.com/downloads)
  + **Azure App Service extension (instructions below)**
* [**Node.js**](https://nodejs.org/)
* [**Bot Framework Emulator**](https://aka.ms/bot-framework-emulator-readme)
* [**git**](https://git-scm.com/) **(optional)**

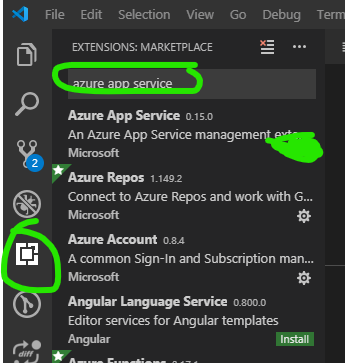
# **Download Workshop code**

* **Check out or download code from the following repository:**
  + [**https://github.com/jakeatmsft/voya\_hack\_bot**](https://github.com/jakeatmsft/voya_hack_bot)

# **Install Azure App Service Extension**

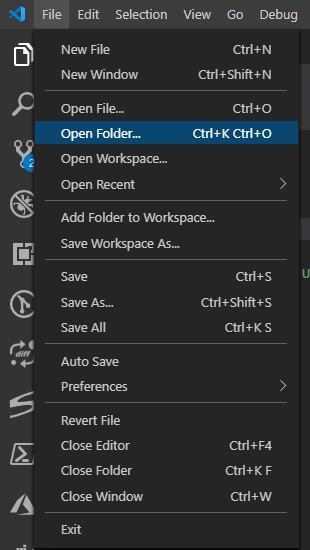
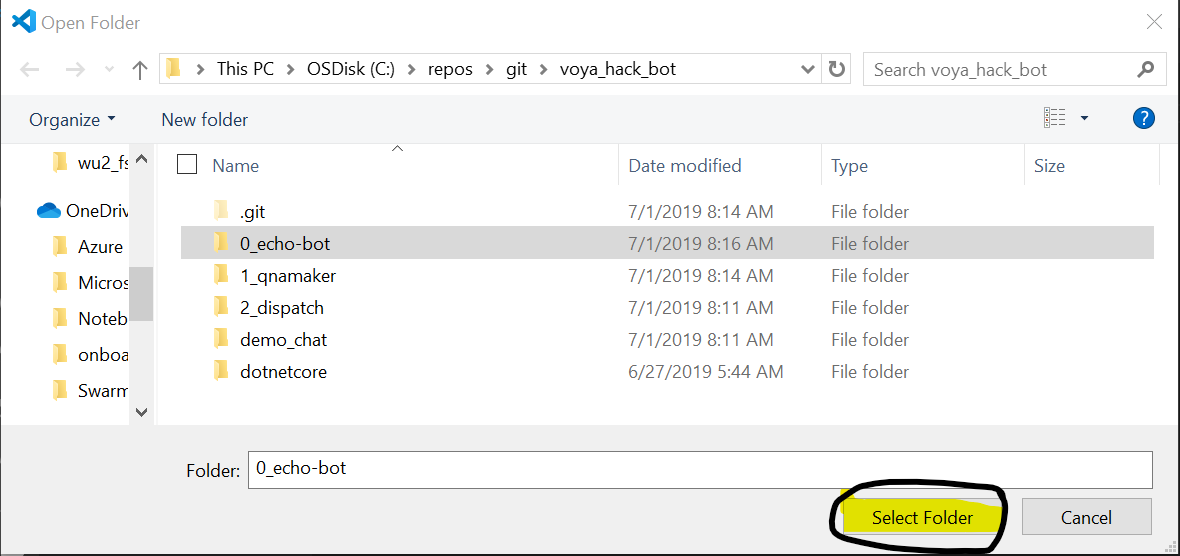
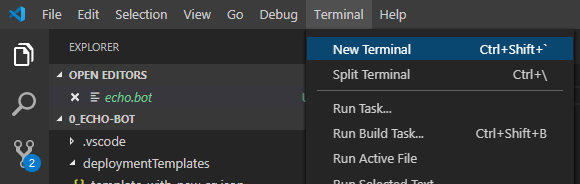
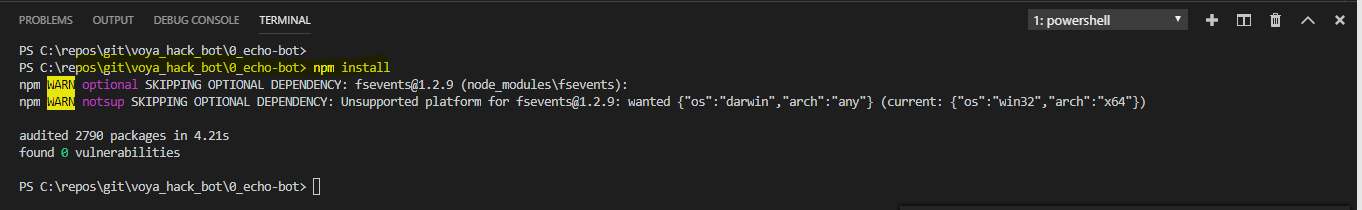
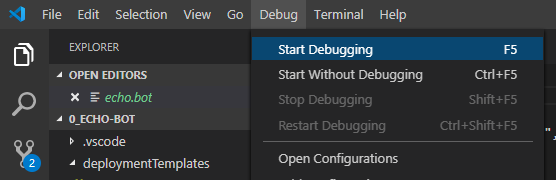
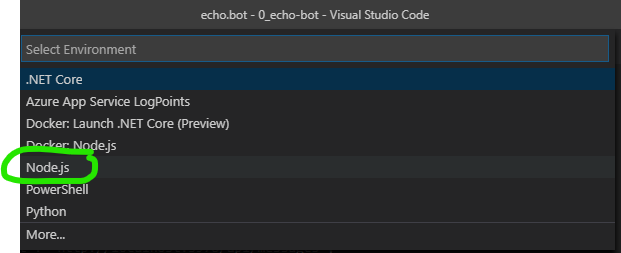
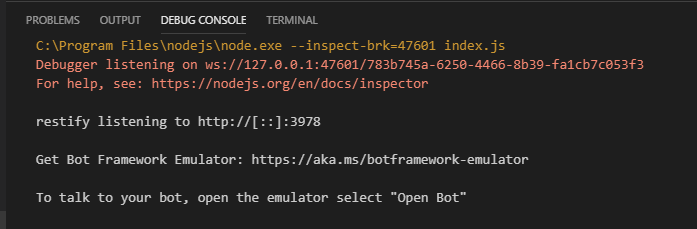
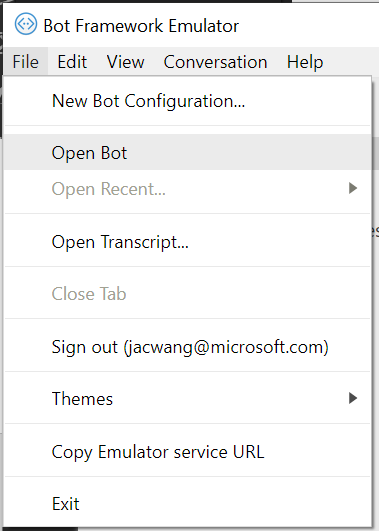
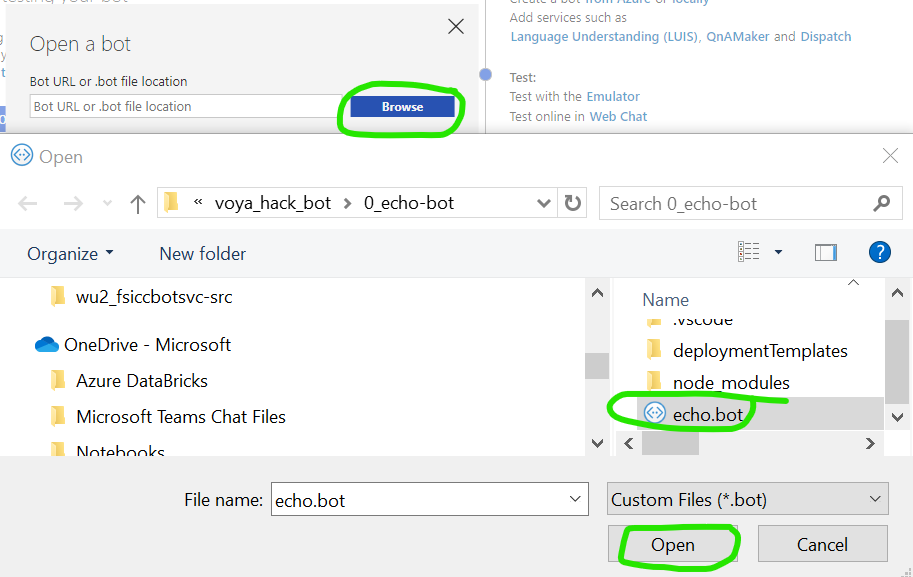
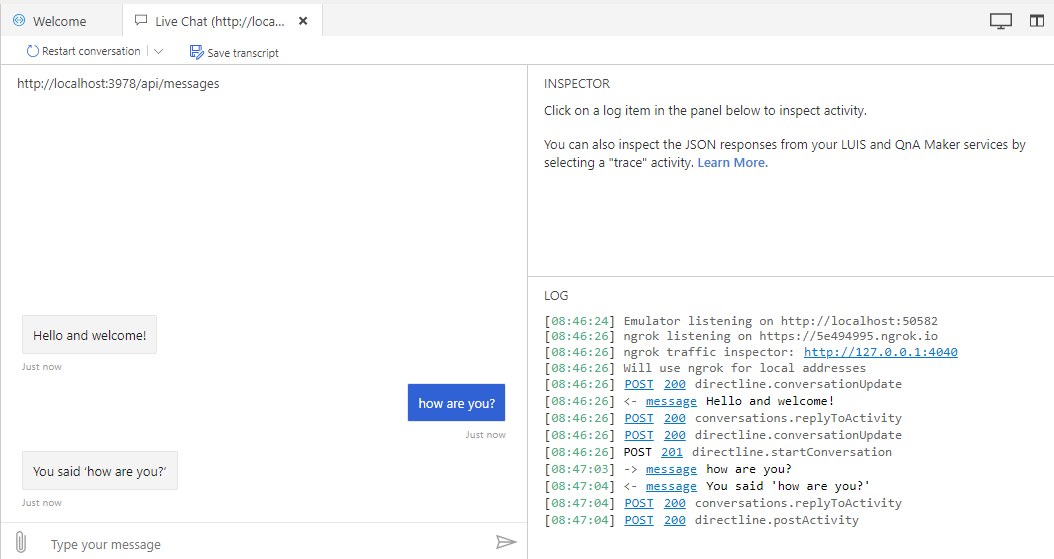
**Open VSCode**

**Go to the Extension menu and search form “Azure App Service”, click install in the lower right hand corner to install. If no button appears, the extension is already installed.**



# **Lab 0 – Echo Bot**

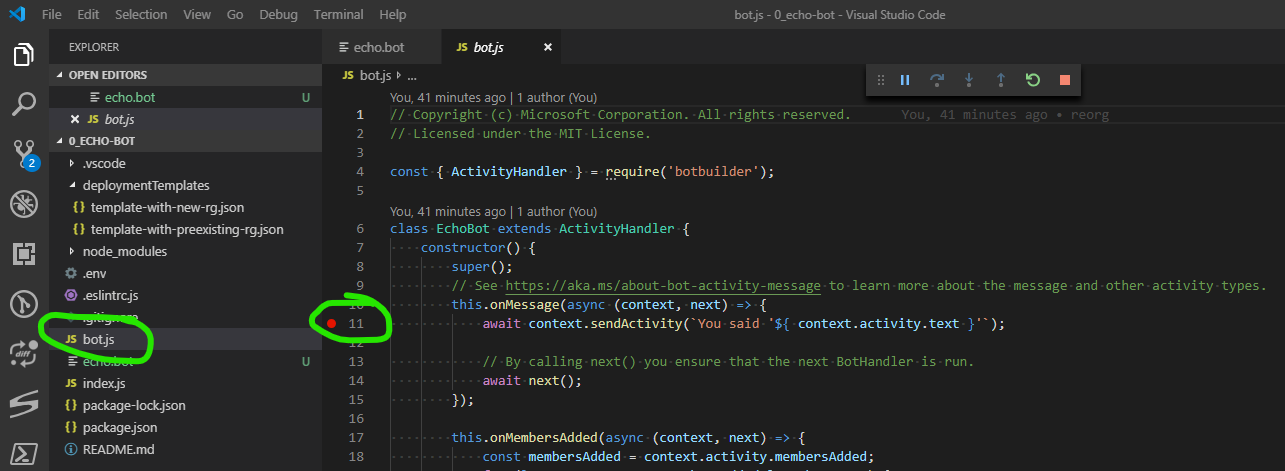
This first example shows a simple bot that will repeat back messages that are sent to it. Let begin:

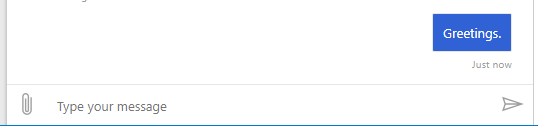
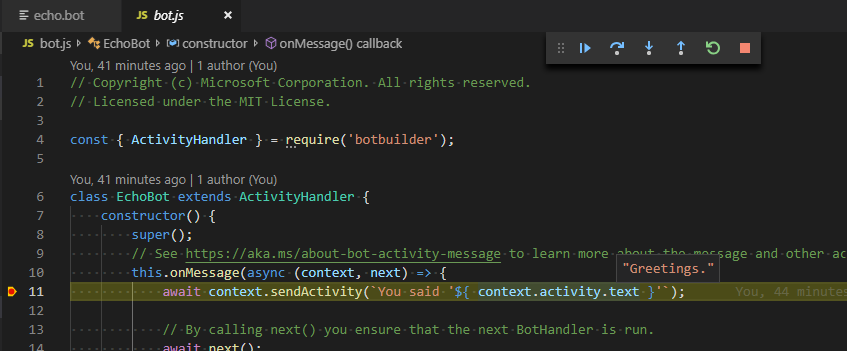
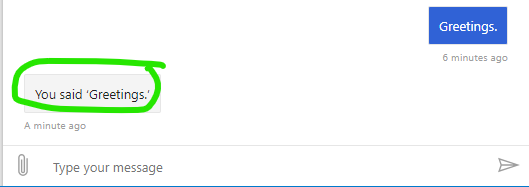
* **Open VSCode.**
* **Open the “0\_echo-bot” folder:**
* ****
* ****
* **Navigate to a new Terminal window**
* ****
* **Run command: npm install**
* ****
* **Now go to Debug menu to Start Debugging**
* ****
* **Select “node.js” environment.**
* ****
* **Verify bot is running in “Debug Console”**
* ****
* **Next open Bot Emulator to connect to bot.**
* **Open the “echo.bot” file to connect to running bot.**
* ****
* ****
* **“Connect”**
* **You should now be connected to your bot and be able to type messages in the chat window.**
* ****

# **Setting Breakpoints**

Now that you are connected to the bot service and are running it locally, we can set breakpoints in the bot and see the code behind the interactions.

* **Set a breakpoint in the bot.js file at line: 11,** you can set the breakpoint by clicking in the margin to the left of the line number. You will see a red circle appear where the breakpoint is set.

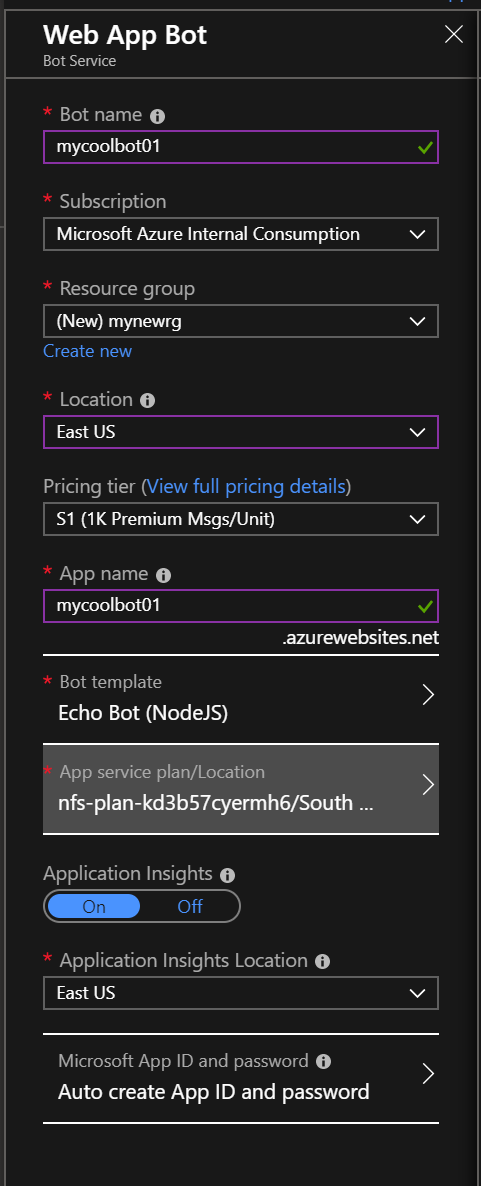
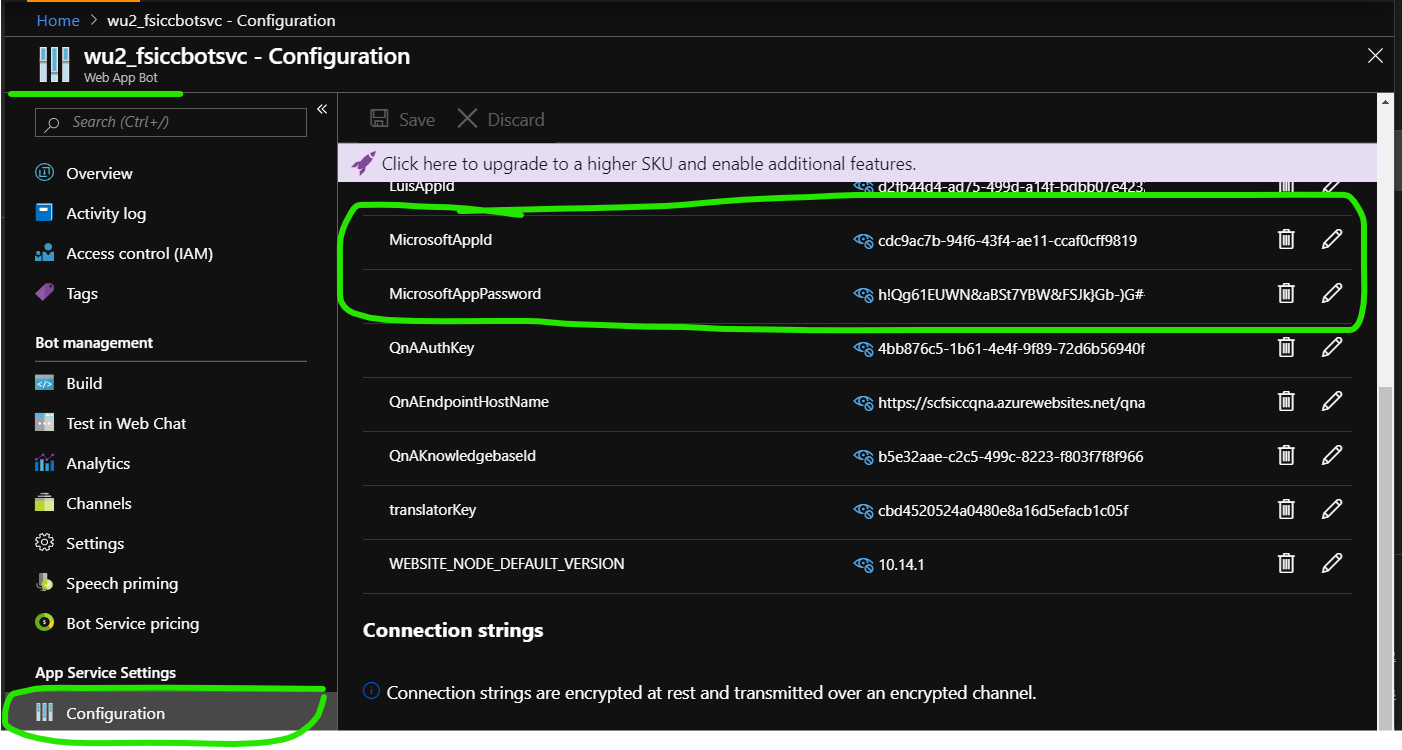
****

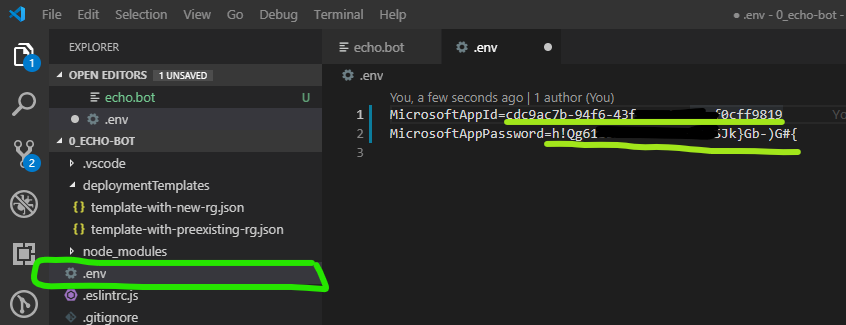
* **Now go back to the Bot Emulator and type a message. You should see the line highlighted once the bot executes that line of code.**
* ****
* **Here you can view the current context by hovering over the code.**
* ****
* **To release control of the breakpoint you can click the “Continue” button in the debug controls.**
* ****
* **Alternatively, you can “Step Into” or “Step Over” lines of code to see the function execution.**
* **The bot will now continue execution and reply within the emulator:**
* ****

# **Bot Deployment**

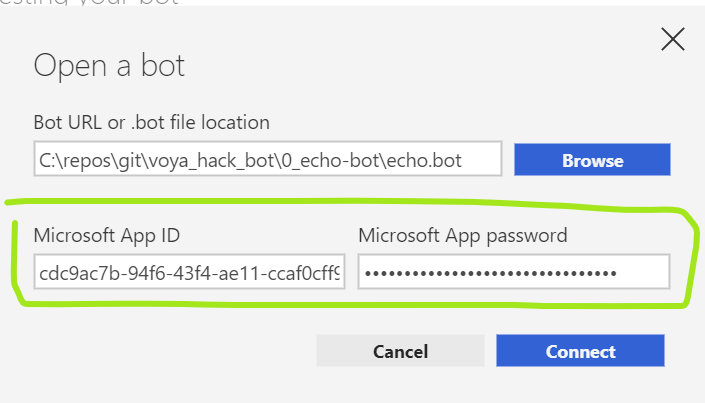
* **To deploy the bot, begin by creating a bot service in the Azure Portal.**
* <https://docs.microsoft.com/en-us/azure/bot-service/bot-service-quickstart?view=azure-bot-service-4.0>
* Use the following settings:

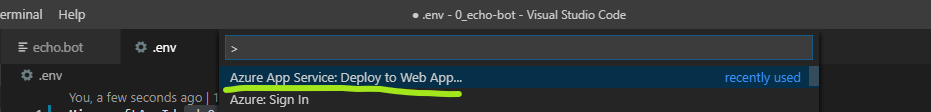
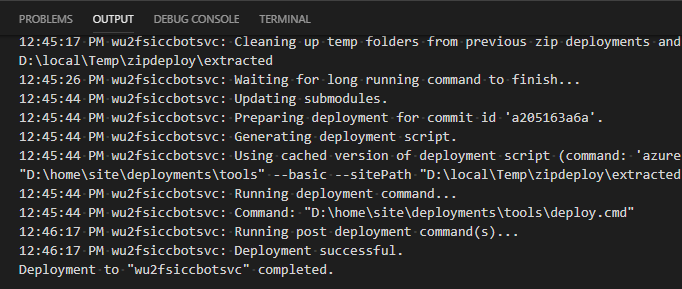
|  |  |  |
| --- | --- | --- |
| **Setting** | **Value** | **Description** |
| **Bot name** | **<display name>** | The display name for the bot that appears in channels and directories. This name can be changed at anytime. |
| **Subscription** | **< your subscription>** | Select the Azure subscription you want to use. |
| **Resource Group** | **<new resource group>** | Create a new resource group |
| **Location** | **East US** | The geographic location for your service. |
| **Pricing tier** | **S1** | The service tier of your application |
| **App name** | **<unique name>** | The unique URL name of the bot. There is a 35 character limit to this field. The App name cannot be changed once the bot is created. |
| **Bot template** | **Echo Bot (NodeJS)** | Node.js Basic Bot |
| **App service plan/Location** | **Your app service plan** | Standard |
| **Application Insights** | **On** | Application diagnostic info |
| **Microsoft App ID and password** | **Auto create App ID and password** | Set to automatically create |

* ****
* **Retrieve, the MicrosoftAppID and MicrosoftAppPassword from the “Configuration” blade:**
* ****
* **Paste the “MicrosoftAppID” and “MicrosoftAppPassword” into the .env file in your bot project:**

****

**(Note: Once you have the “AppID and Password” defined in your Bot solution if you want to connect through the emulator you will need to reconnect with the AppID and Password in the “Open” prompt.)**

****

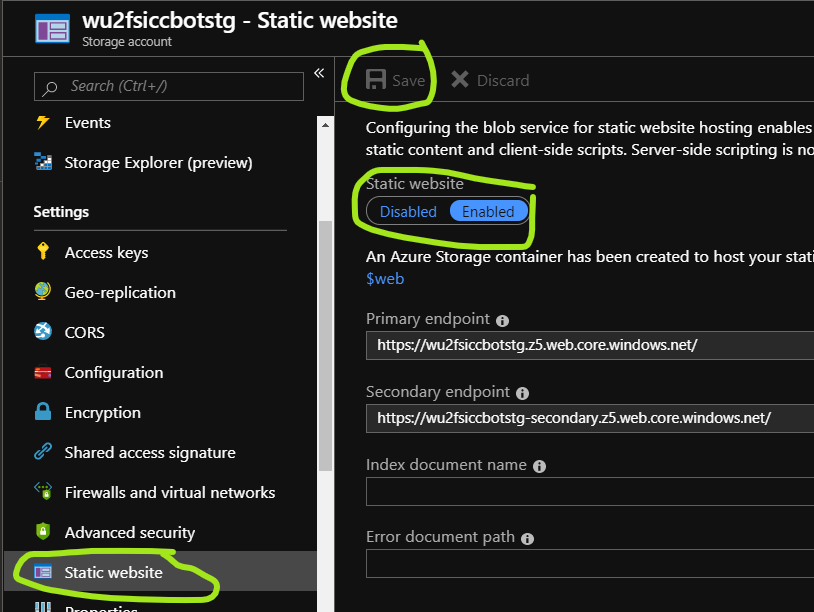
* **Once the “.env” file is updated you can deploy using the App Service Extension, by opening the command menu (Control – Shift – P) and using the “Azure App Service: Deploy to Web App” command.**
* ****
* **Select the current folder.**
* **Select the target bot App Service (created in step 1).**
* **Verify the deployment. Wait for “Deployment Completed” message.**
* ****

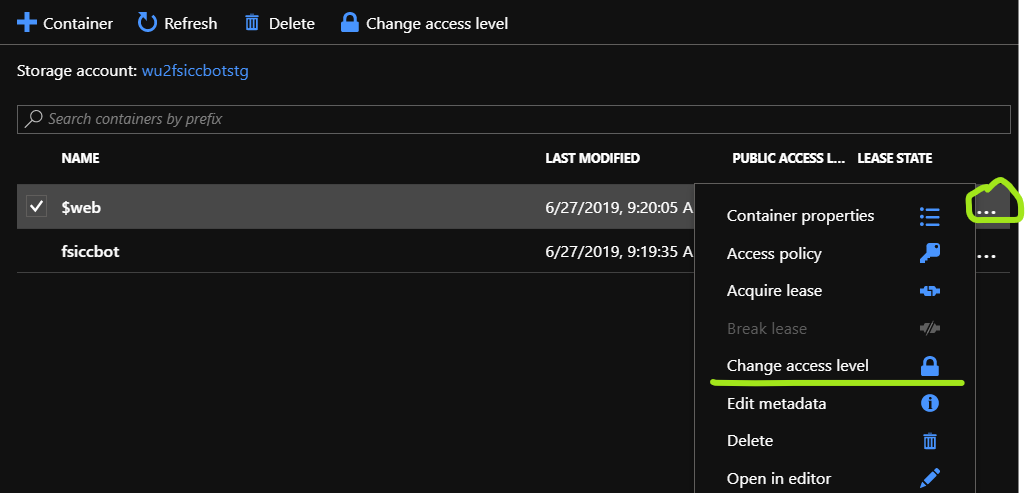
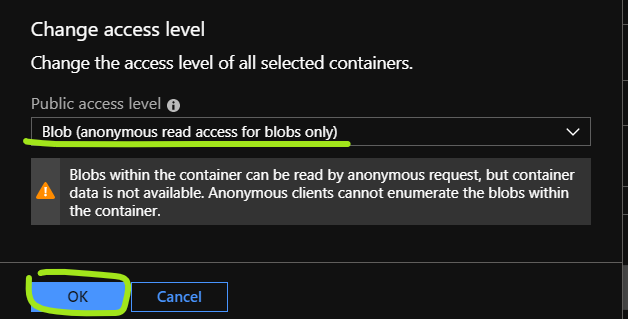
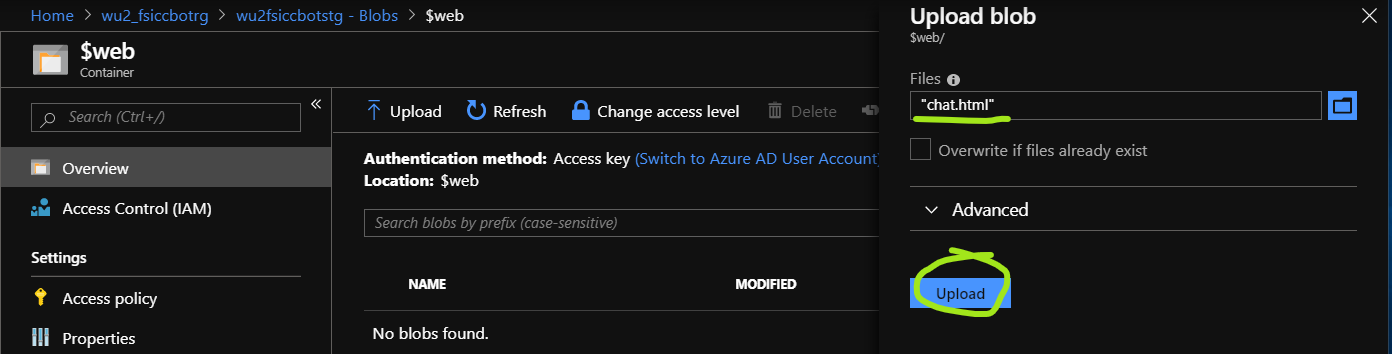
# Deploy Webchat Page

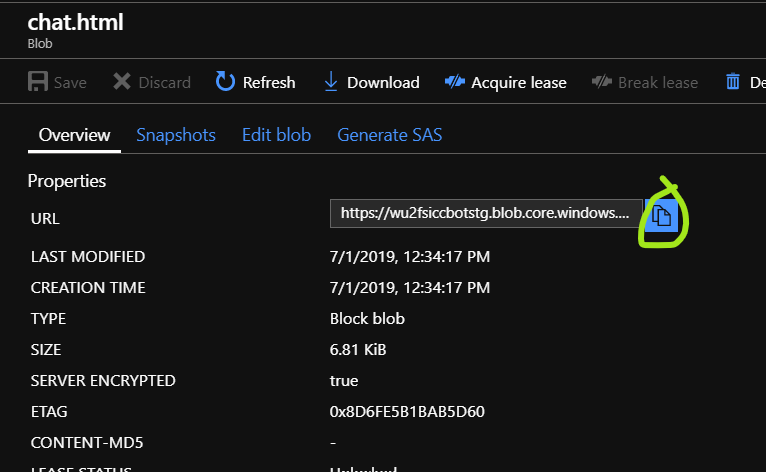
* Create Azure Storage Account –
* <https://docs.microsoft.com/en-us/azure/storage/common/storage-quickstart-create-account?tabs=azure-portal>
* Use default settings:

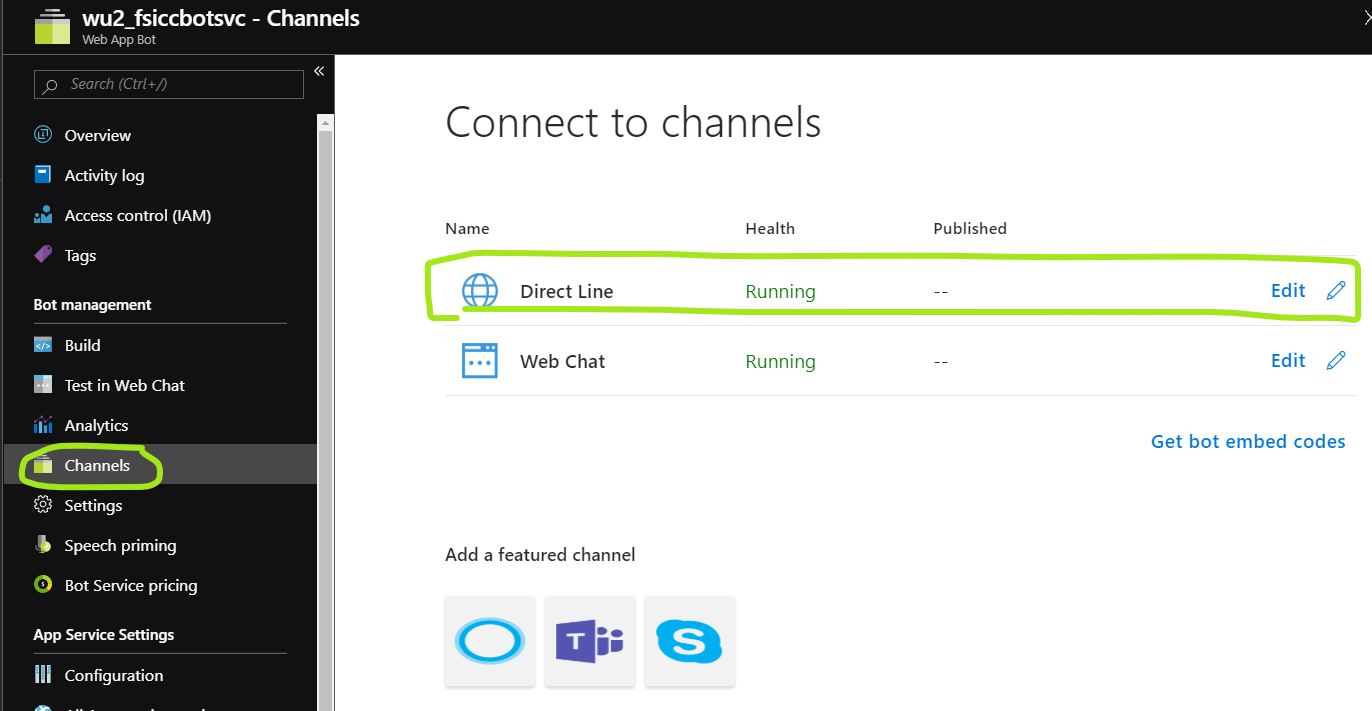
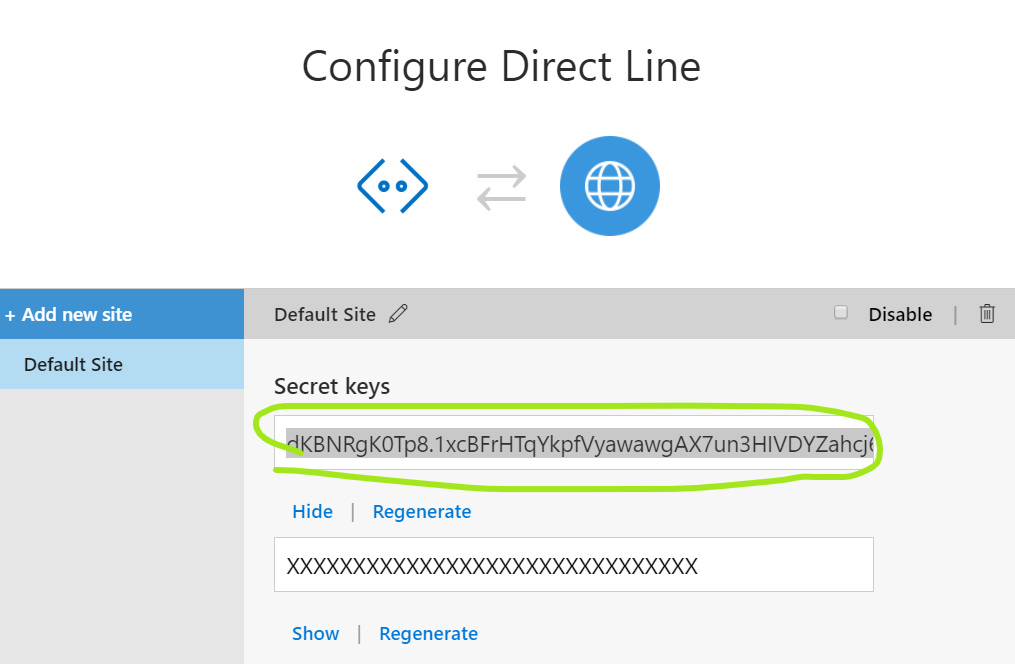
| **Field** | **Value** |
| --- | --- |
| Deployment model | Resource Manager |
| Performance | Standard |
| Account kind | StorageV2 (general-purpose v2) |
| Replication | Read-access geo-redundant storage (RA-GRS) |
| Access tier | Hot |

* Open Storage Account in Azure Portal and Select “Static Website” blade.
* Enable Static Website. Save.

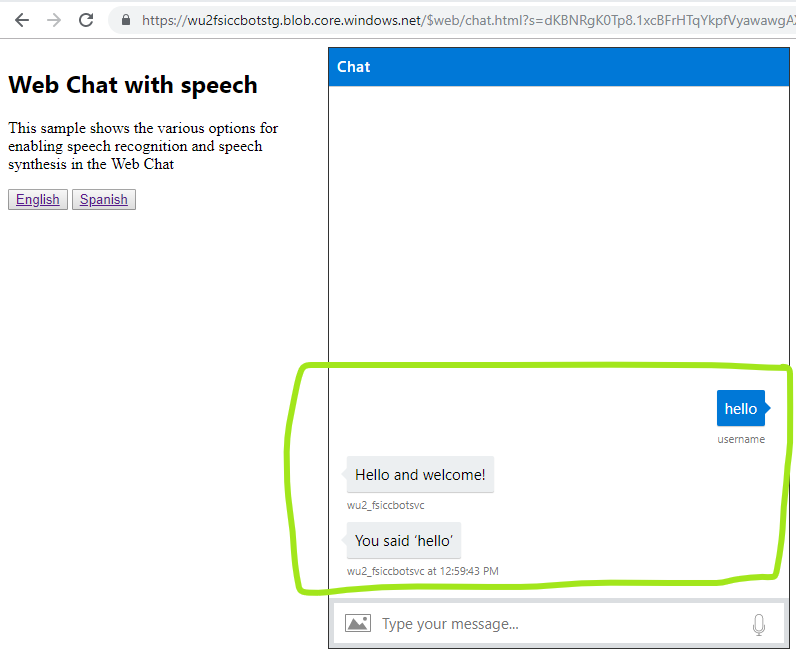


* Select Blobs -> $web Container.
* Change access level to “Blob”
* 
* 
* Upload “chat.html” to $web container.
* 
* **Open chat.html blob and Copy URL to new browser window to view webpage.**

****

* **To connect the chat window with your deployed service you need to retrieve service keys to your Bot Service.**
* **Navigate to your deployed Bot Service.**
* **Open Channels**
* ****
* **If not enabled – Add Direct Line Channel to Bot Service.**
* **Click edit and retrieve “Secret Key” for Bot.**
* ****
* **Copy url for chat.html and fill in with your own settings:**
  + **https://<your storage name>.azurewebsites.net/$web/chat.html?s=<direct line secret key>**
* **Example:** [**https://wu2fsiccbotstg.blob.core.windows.net/$web/chat.html?s=dKBNRgK0Tp8.1xcBFrHTqYkpfVyawawgAX7un3HIVDYZahcj6beEK-E**](https://wu2fsiccbotstg.blob.core.windows.net/$web/chat.html?s=dKBNRgK0Tp8.1xcBFrHTqYkpfVyawawgAX7un3HIVDYZahcj6beEK-E)

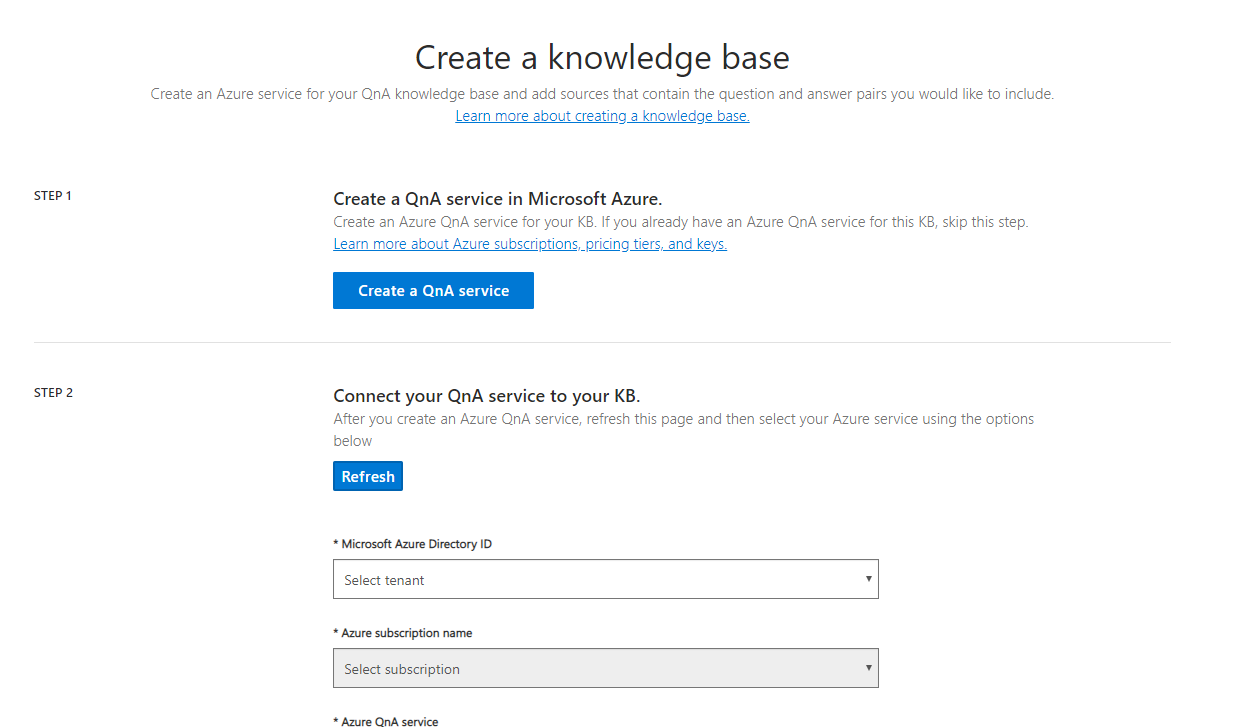
You can now interact with your bot in the chat window:

****

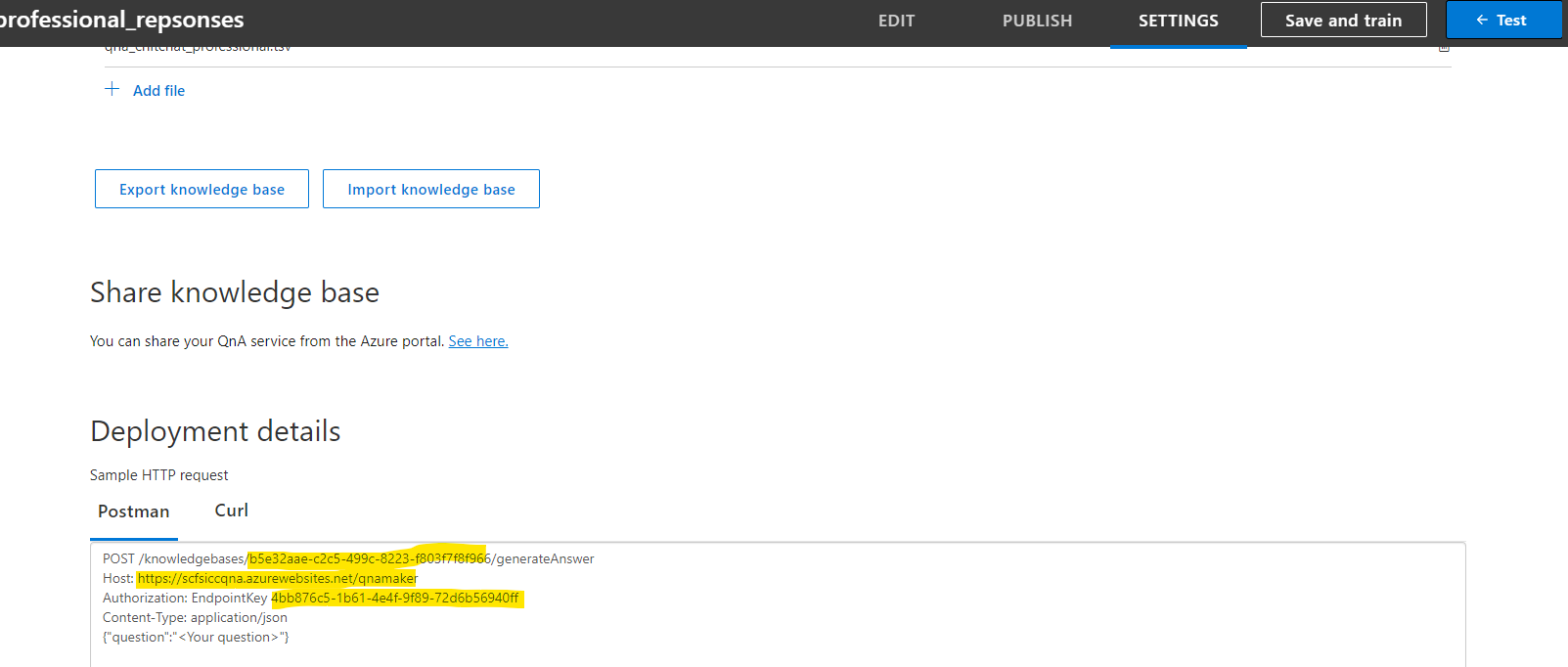
# Lab 1 - QnA Bot

In this lab we will create a Question and Answer (QnA) bot that can reply to simple questions using answers based off of FAQs, documents, manuals, or editorial content.

## Create QnA Knowledge Base (KB).

* To begin, go to [www.qnamaker.ai](http://www.qnamaker.ai).
* Login using Azure account credentials.
* Create knowledge case.
* 
* Once KB has been created, “Save and train” to build KB.

## Save Settings.

* Navigate to “Settings” and go to “Deployment details” section.
* 

Example:

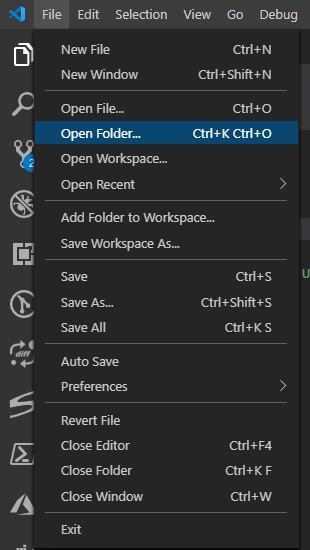
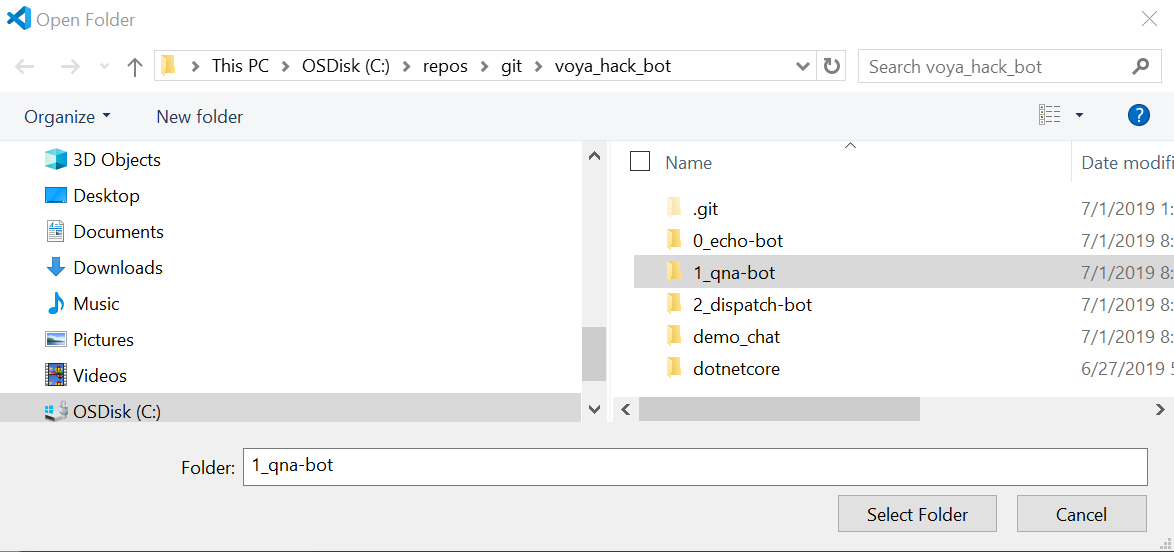
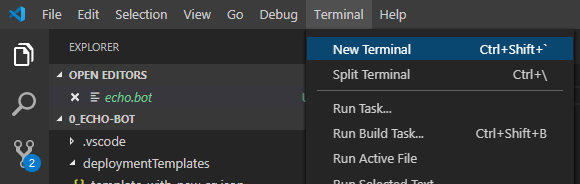
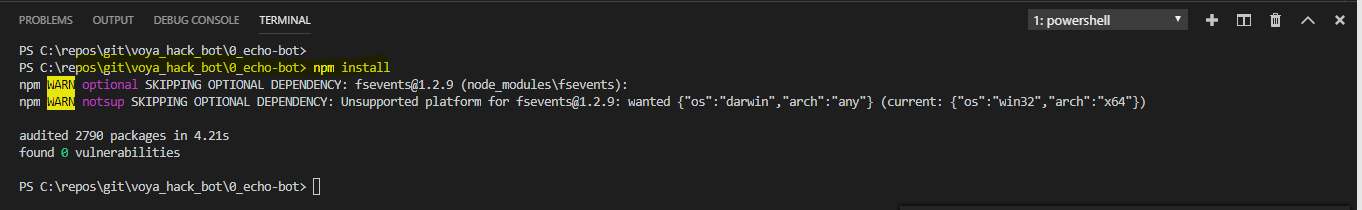
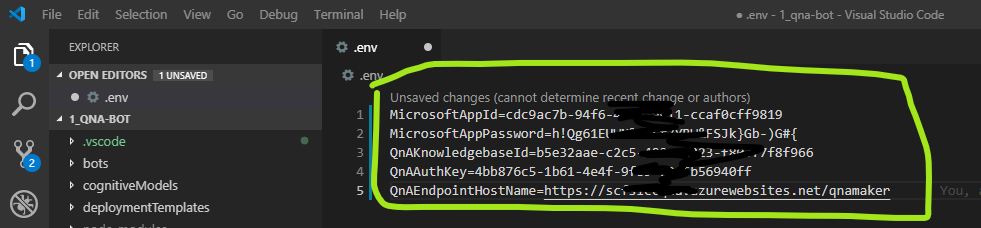
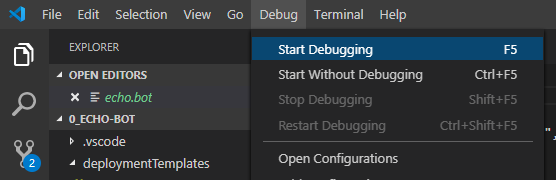
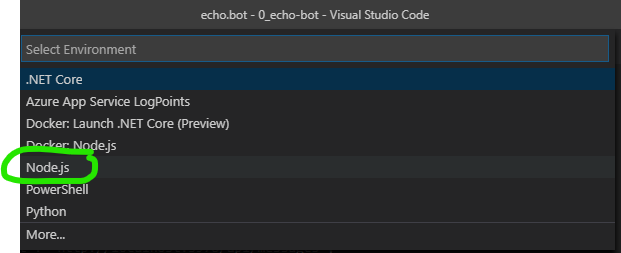
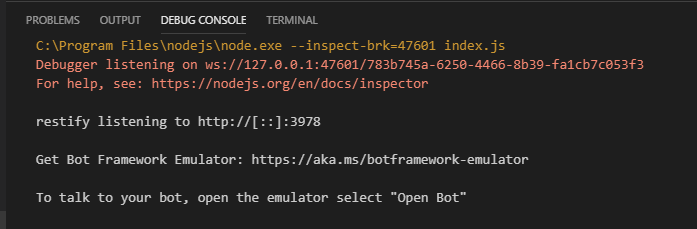
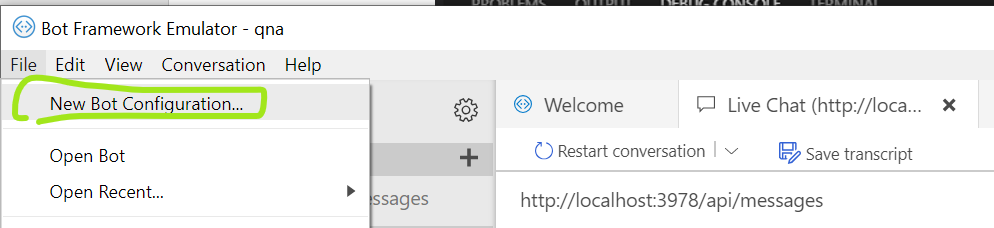
POST /knowledge bases/<QnAKnowledgebaseId>/generateAnswer

Host: <QnAEndpointHostName>

Authorization: EndpointKey <QnAAuthKey>

Save the configuration keys to text editor as they will be required by bot.

# Open QnA-Bot

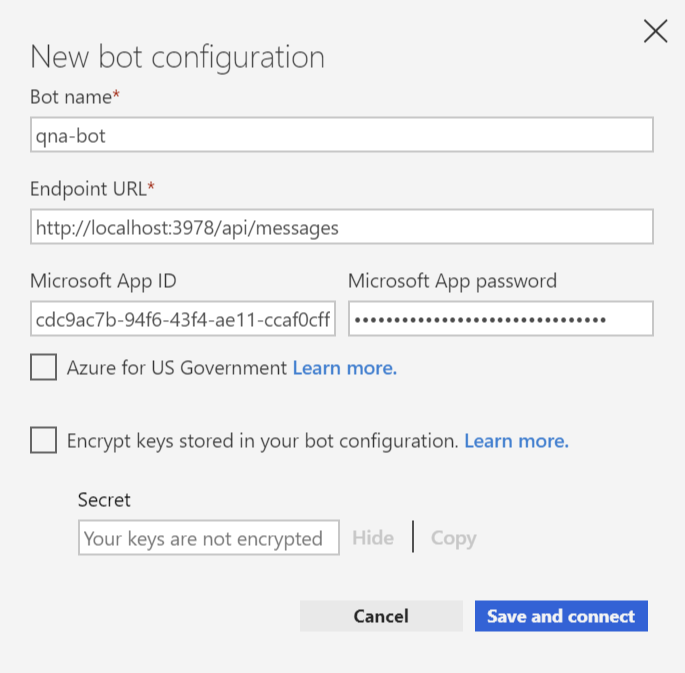
* Open VSCode, open the “1\_qna-bot” folder from git repository.
* ****
* 
* **Navigate to a new Terminal window**
* ****
* **Run command: npm install**
* ****
* Open the “.env” file and fill in configuration values using from previous steps.
  + AppID and Password - [Instructions](#_Bot_Deployment)
  + QnA Configs - [Instructions](#_Save_Settings.)
* 
* **Now go to Debug menu to Start Debugging**
* ****
* **Select “node.js” environment.**
* ****
* **Verify bot is running in “Debug Console”**
* ****
* **Next open Bot Emulator to connect to bot.**
* **Create a New Bot Configuration:**
* ****

**Bot name: qna-bot**

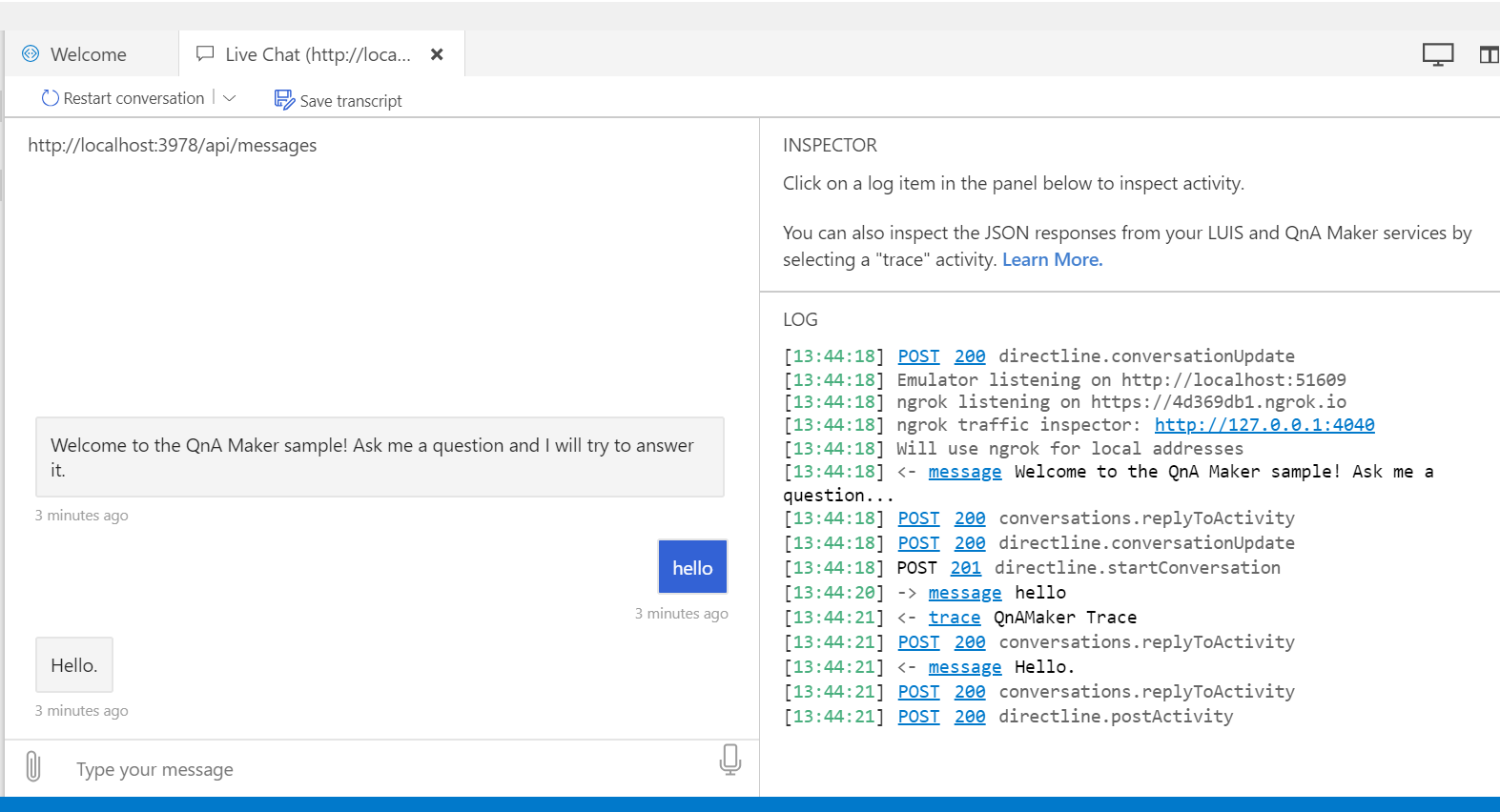
**Host:** [**http://localhost:3978/api/messages**](http://localhost:3978/api/messages)

**App ID and App Password from “.env” file**

**Save and connect.**



Now you should be connected to your QnA Knowledge Base:



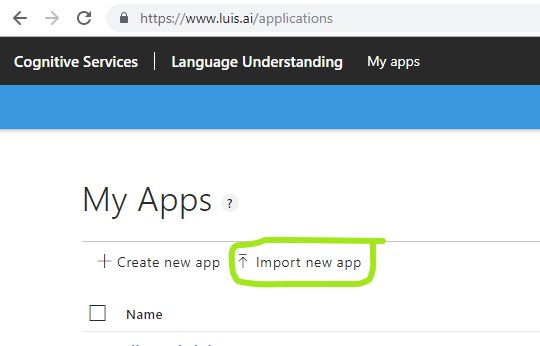
To Deploy QnA Bot see instructions here: [Link](#_Bot_Deployment)

# Lab 2 - Dispatch-bot

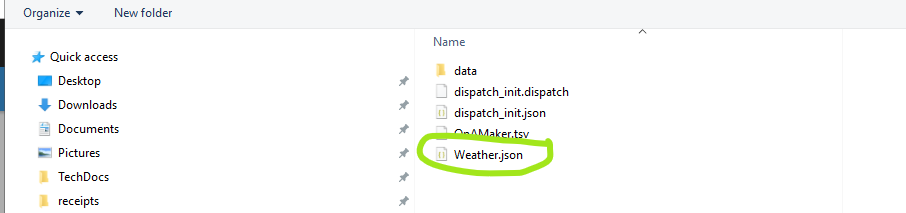
In this lab we will extend our previous QnA bot to include Language Understanding, through our LUIS service. In addition to simple questions LUIS can parse complex questions to understand intents and entities using natural language processing capabilities.

## Create LUIS Model

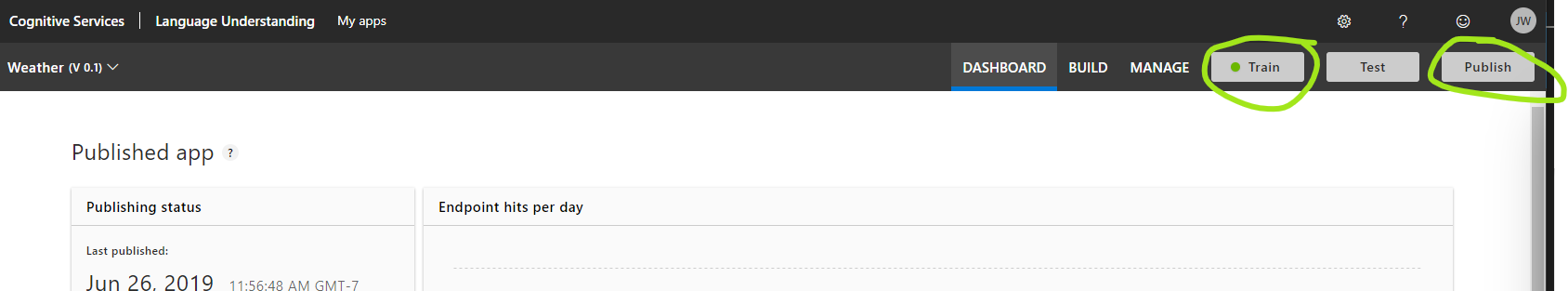
* Go to [www.luis.ai](http://www.luis.ai)
* Login using Azure account credentials.
* Select import new app.



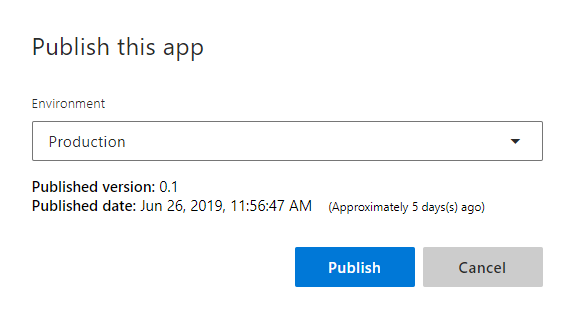
Open the weather.json file from the folder: \2\_dispatch-bot\cognitiveModels



* Next open the “Weather” app and Train the model



* After training publish the model to Production.



## Create dispatch model

Now to use the newly created LUIS model with our existing QnA Model, we will use a tool called dispatch. The CLI interface for the dispatch tool creates the model for dispatching to the correct LUIS or QnA Maker app.

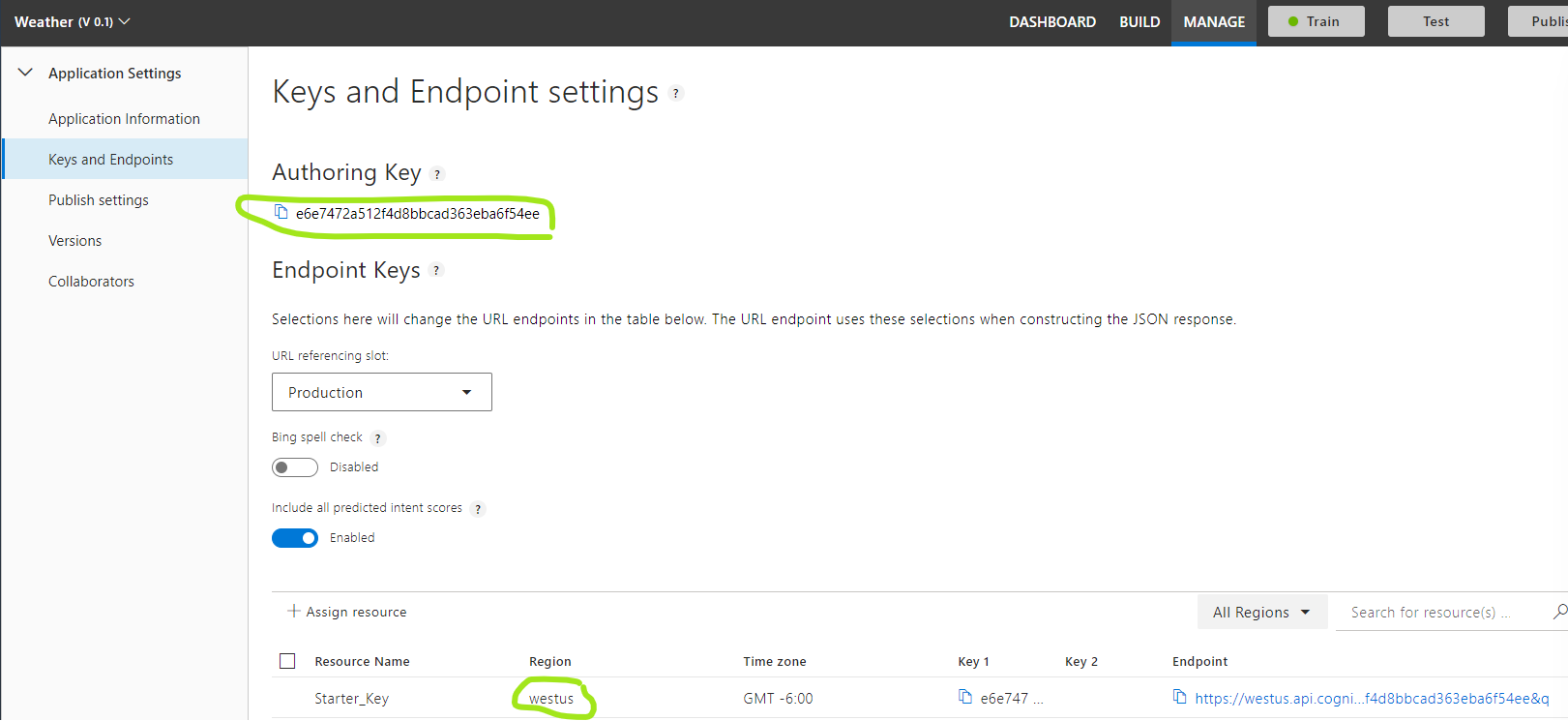
## Install dispatch

To install dispatch application open terminal window and run command:

npm install -g botdispatch

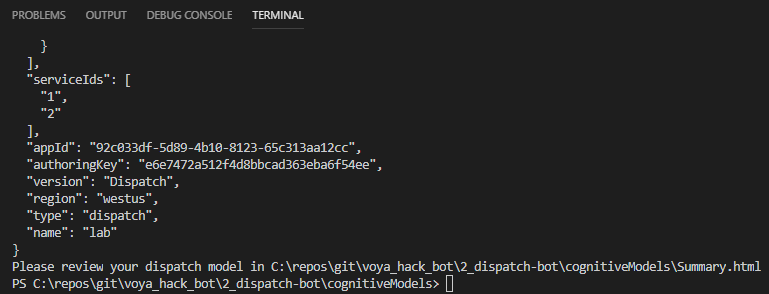
## Retrieve Keys

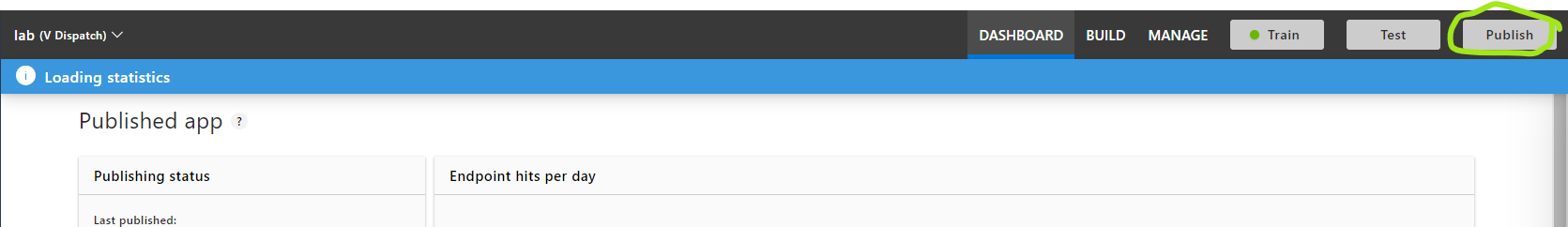
* Open the Keys and Endpoints page in LUIS portal.



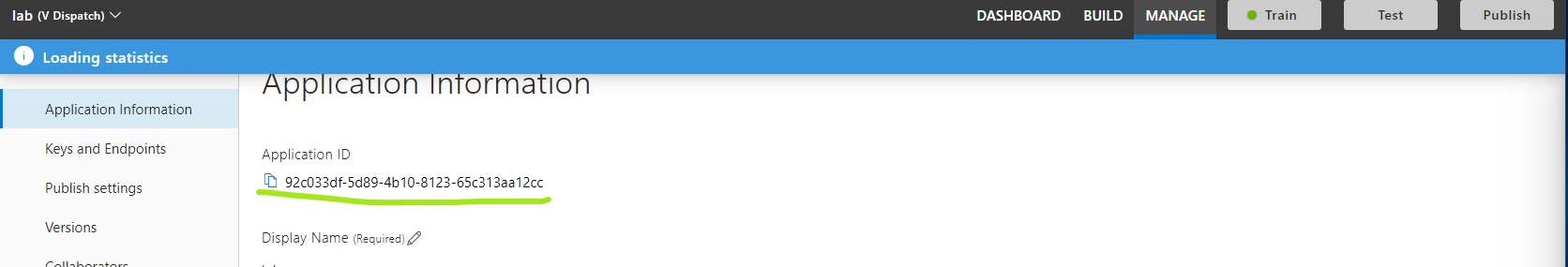
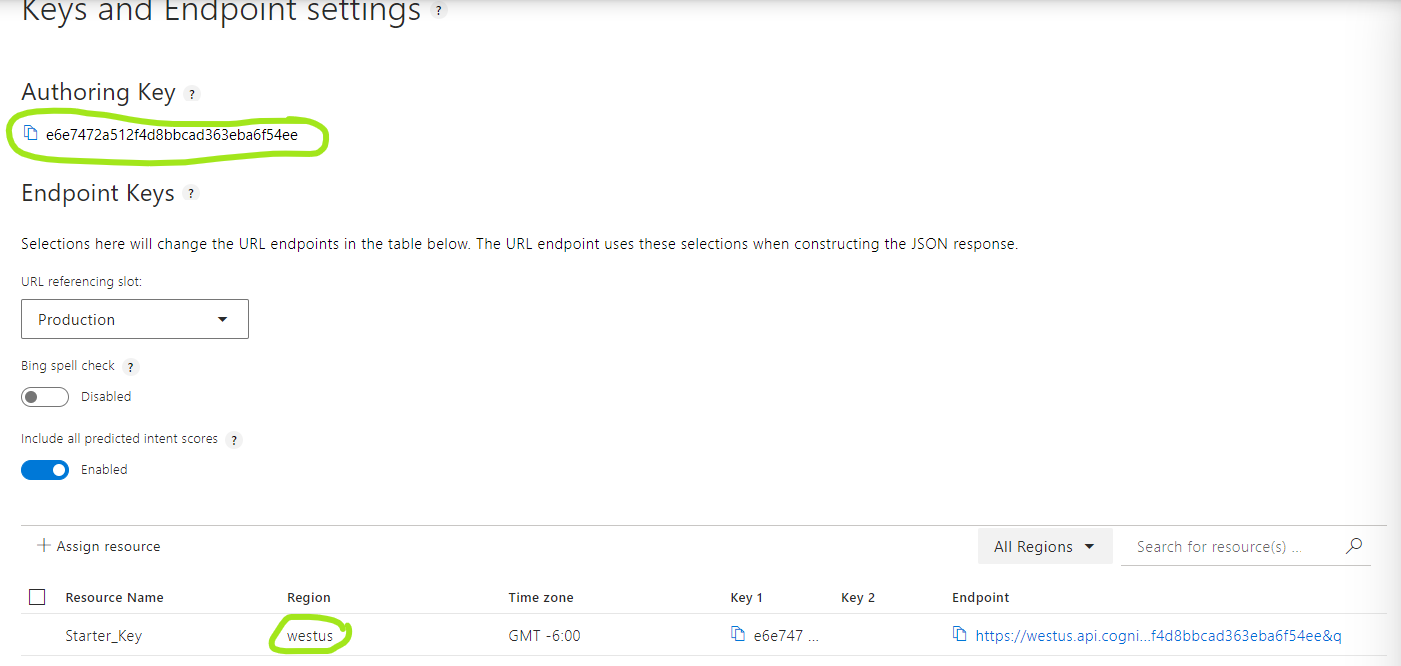
Take note of Authoring Key and Region as you will need these to create dispatch model.

## Build dispatch model

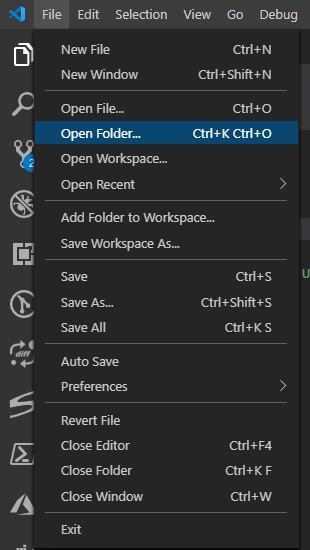
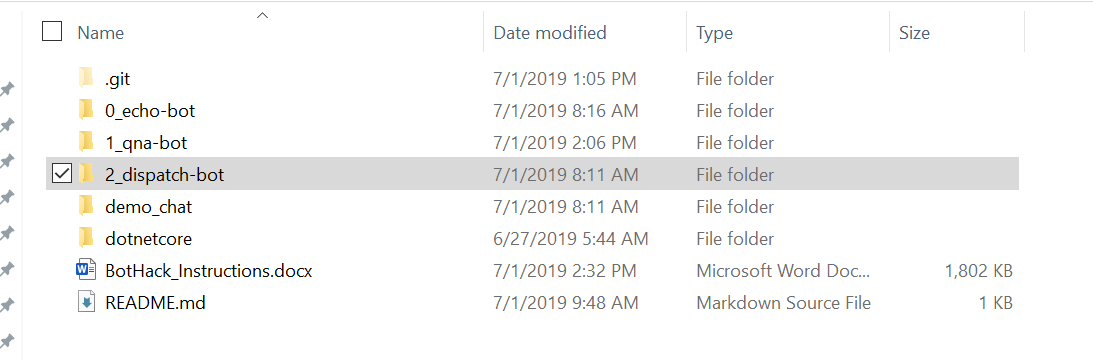
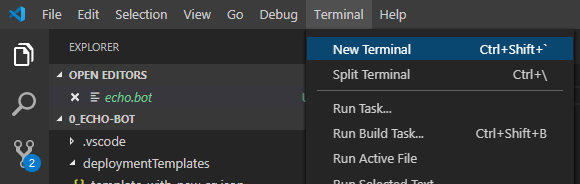
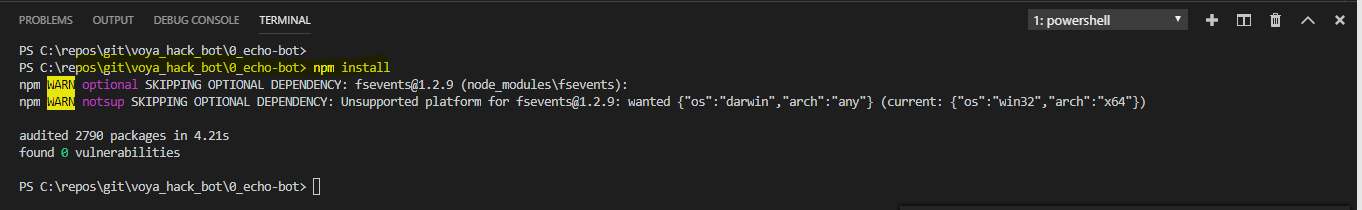
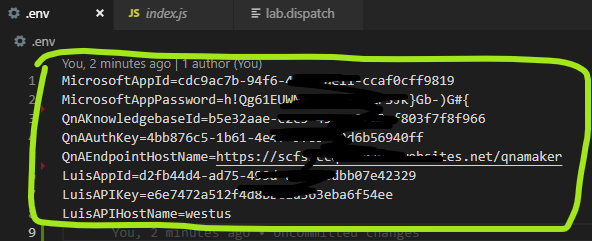
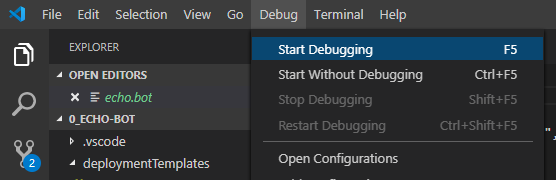
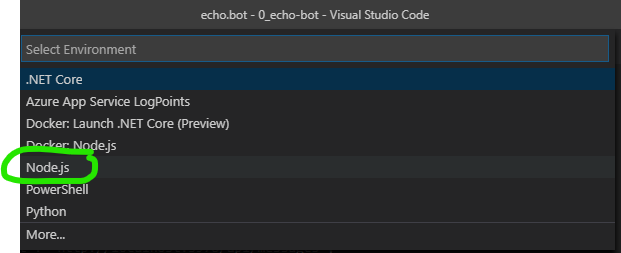
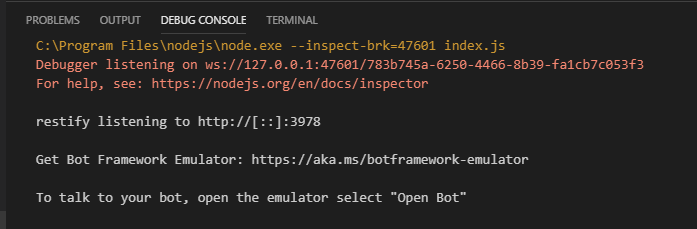
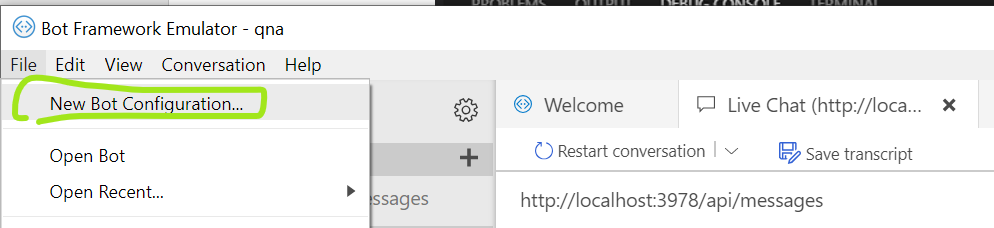
* Open a command prompt or terminal window, and change directories to the **CognitiveModels** directory
* create a lab.dispatch file for your dispatch model.
* dispatch init -n lab.dispatch --luisAuthoringKey "<your-luis-authoring-key>" --luisAuthoringRegion <your-region>
* Use dispatch add to add your LUIS apps and QnA Maker knowledge bases to the .dispatch file.
* dispatch add -t luis -i "<app-id-for-weather-app>" -n "<name-of-weather-app>" -v <app-version-number> -k "<your-luis-authoring-key>" --intentName l\_Weather
* dispatch add -t qna -i "<knowledge-base-id>" -n "<knowledge-base-name>" -k "<azure-qna-auth-key>" --intentName q\_sample-qna
* Use dispatch create to generate a dispatch model from the .dispatch file.
* dispatch create
* 
* Publish the dispatch LUIS app, just created.



## Get LUIS Settings

* Once model has been published open “Application Information” to retrieve LUIS App ID
* 
* API Key and Hostname are on the “Keys and Endpoints” blade.
* 
* Note: APIHostName is the region the bot is deployed to. In this case “westus”.

# Open dispatch-bot

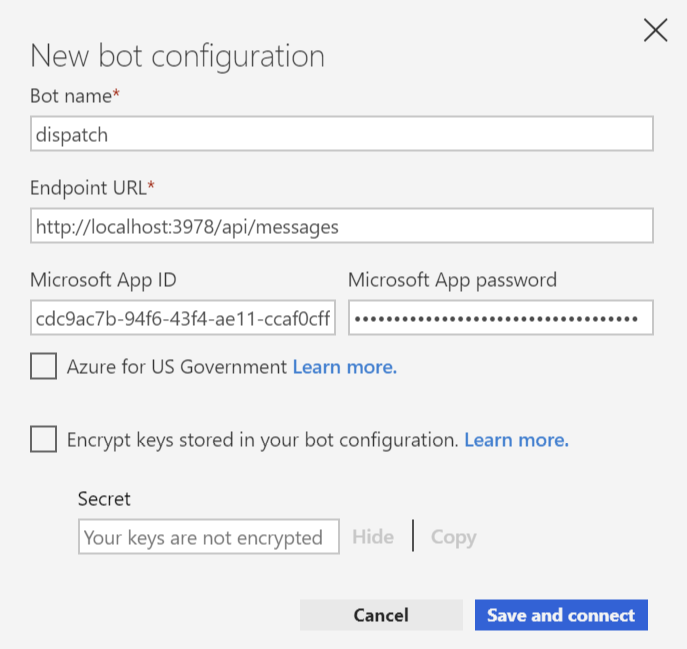
* Open VSCode, open the “2\_dispatch-bot” folder from git repository.
* ****
* 
* **Navigate to a new Terminal window**
* ****
* **Run command: npm install**
* ****
* Open the “.env” file and fill in configuration values using from previous steps.
  + AppID and Password - [Instructions](#_Bot_Deployment)
  + QnA Configs - [Instructions](#_Save_Settings.)
  + Luid Configs - [Instructions](#_Get_LUIS_Settings)
* 
* **Now go to Debug menu to Start Debugging**
* ****
* **Select “node.js” environment.**
* ****
* **Verify bot is running in “Debug Console”**
* ****
* **Next open Bot Emulator to connect to bot.**
* **Create a New Bot Configuration:**
* ****

**Bot name: dispatch**

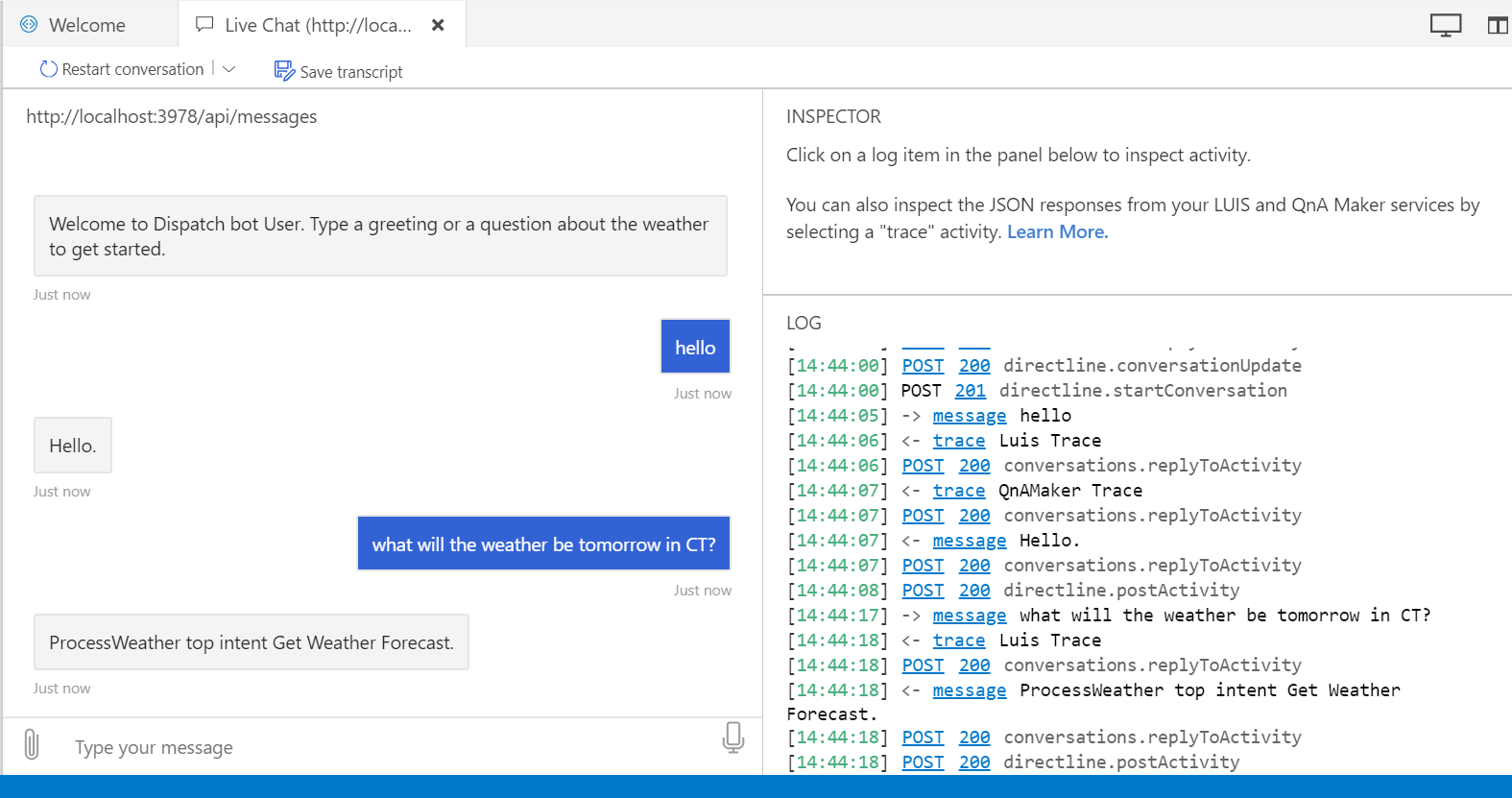
**Host:** [**http://localhost:3978/api/messages**](http://localhost:3978/api/messages)

**App ID and App Password from “.env” file**

**Save and connect.**



Now you should be connected to your dispatch bot. If you ask a question related to weather it should respond with Intent and Entities of query:



To Deploy QnA Bot see instructions here: [Link](#_Bot_Deployment)