Create a Bot, check-in to Team Services (creates GitHub repository)

# Disclaimer

1. Azure is always evolving. These steps are valid as of 07/21/2018. My first LUIS bot was created in 11/2017 when it was in Preview mode. I ran into a lot of issues with permissions, because I did all of the steps in the wrong order. Eventually I figured out the correct order. Now more than six months later I've revisited the process to find it's no longer in Preview mode.

# What is Language Understanding (LUIS)?

<https://docs.microsoft.com/en-us/azure/cognitive-services/luis/home>

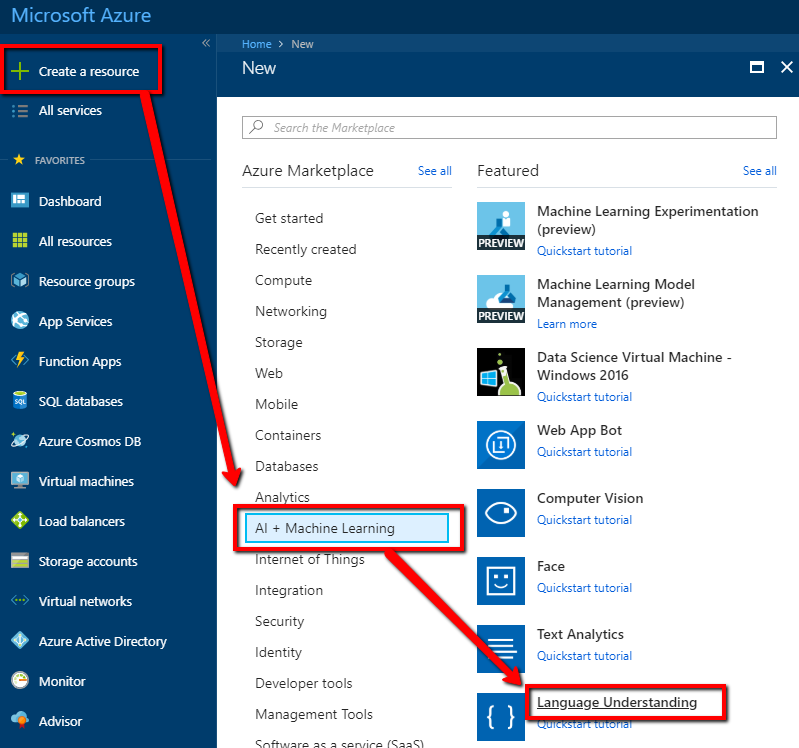
Per Azure:

Language Understanding (LUIS) is a cloud-based service that applies custom machine-learning to a user's conversational, natural language text to predict overall meaning, and pull out relevant, detailed information.

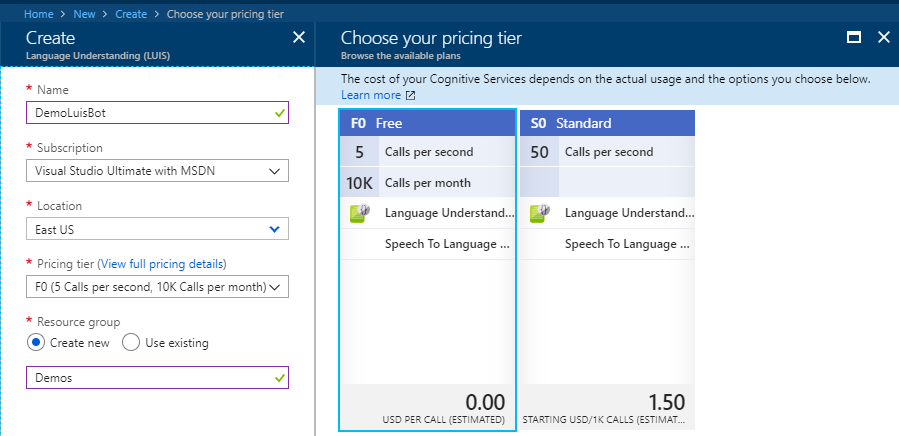
A client application for LUIS can be any conversational application that communicates with a user in natural language to complete a task. Examples of client applications include social media apps, chatbots, and speech-enabled desktop applications.

# Create a LUIS Bot

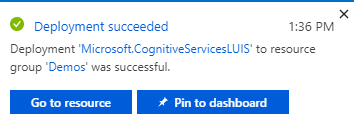
1. <https://portal.azure.com>
2. Sign in with your MSDN account
3. Click "Create a resource" > "AI + Machine Learning" > "Language Understanding"



1. Fill out the form
   1. Name: DemoLuisBot
   2. Subscription: Visual Studio Ultimate with MSDN
   3. Location: East US
   4. Pricing tier: F0 (5 Calls per second, 10K Calls per month) – Free account
   5. Resource Group:
      1. Select "Create new"
      2. Name: Demos
2. Click "Create"

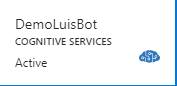


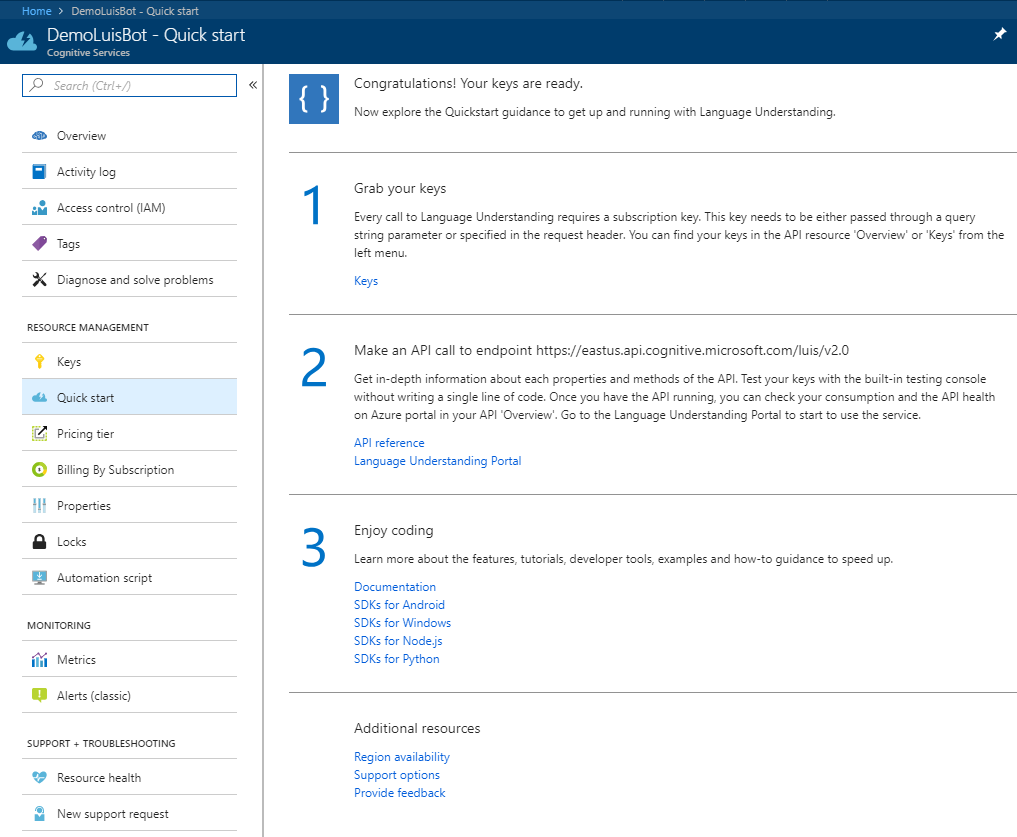
1. When prompted click "Pin to dashboard"



# Open the Quick Start page

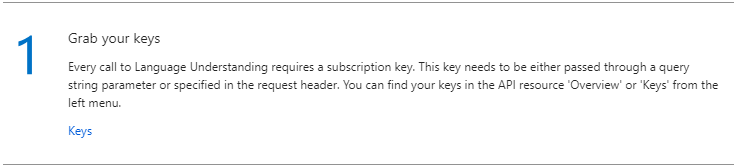
1. Click the DemoLuisBot panel on your Azure dashboard





# Grab your keys

1. [Open the Quick Start page](#_Open_the_Quick)
2. Step 1: Grab your keys

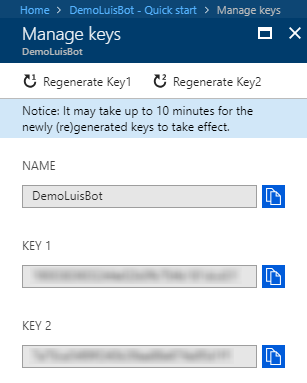


1. Per Azure:

Grab your keys

Every call to Language Understanding requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header. You can find your keys in the API resource 'Overview' or 'Keys' from the left menu.

1. Click "Keys" > Record them so you don't have to keep coming back to this page.



# Make an API call to endpoint

<https://eastus.api.cognitive.microsoft.com/luis/v2.0>

<https://westus.dev.cognitive.microsoft.com/docs/services/5890b47c39e2bb17b84a55ff/operations/5890b47c39e2bb052c5b9c2f>

1. [Open the Quick Start page](#_Open_the_Quick)
2. Step 2: Make an API call to endpoint
3. Per Azure:

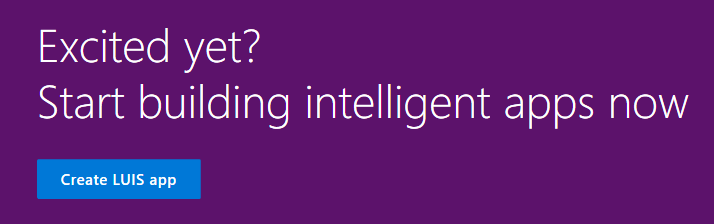
Make an API call to endpoint https://eastus.api.cognitive.microsoft.com/luis/v2.0

Get in-depth information about each properties and methods of the API. Test your keys with the built-in testing console without writing a single line of code. Once you have the API running, you can check your consumption and the API health on Azure portal in your API 'Overview'. Go to the Language Understanding Portal to start to use the service.

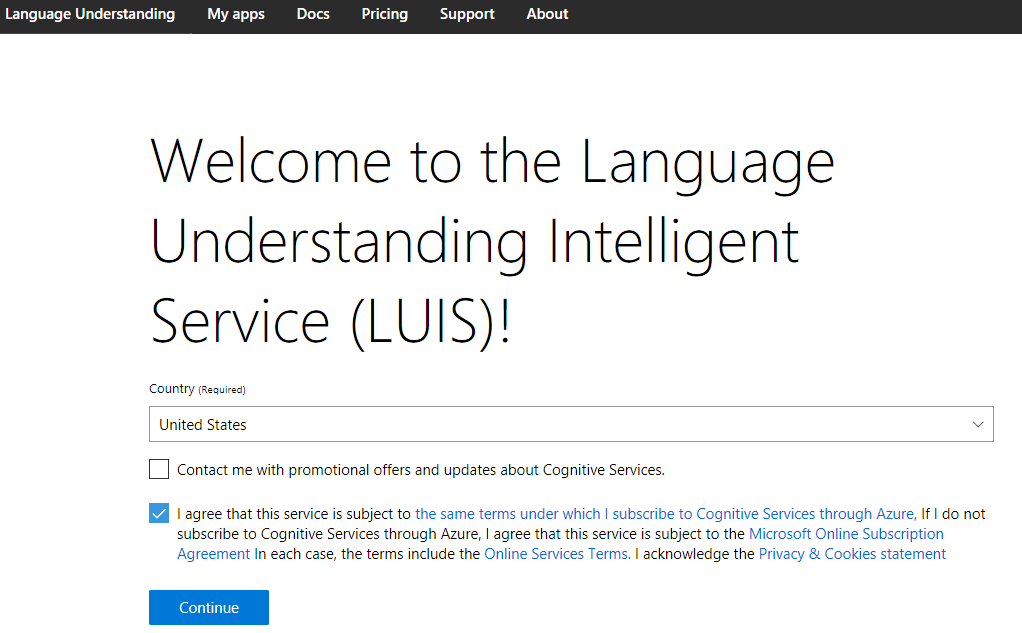
Did I miss something here?!

# Create a Luis.ai Service

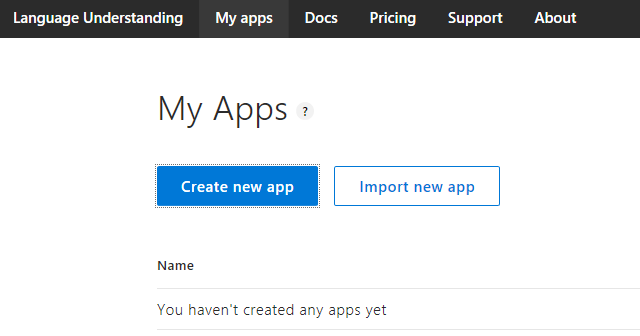
1. <https://www.luis.ai/home>
2. Sign in with your MSDN account
3. Scroll to the bottom > Create LUIS app



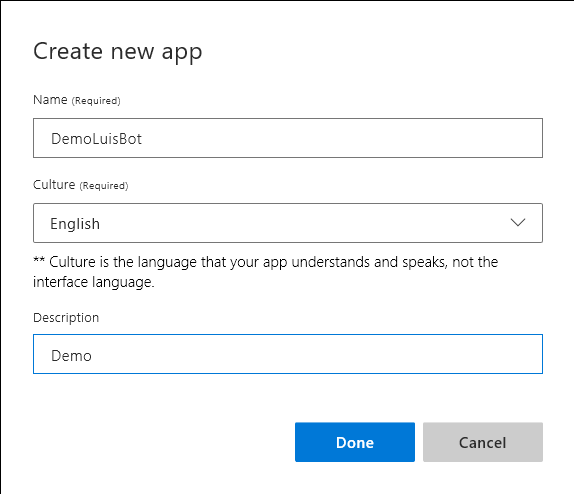
1. Fill out the form
   1. Country: United State
   2. Contact: Uncheck
   3. I agree: Check
      1. Understand what you are agreeing to



1. Click "Create new app"

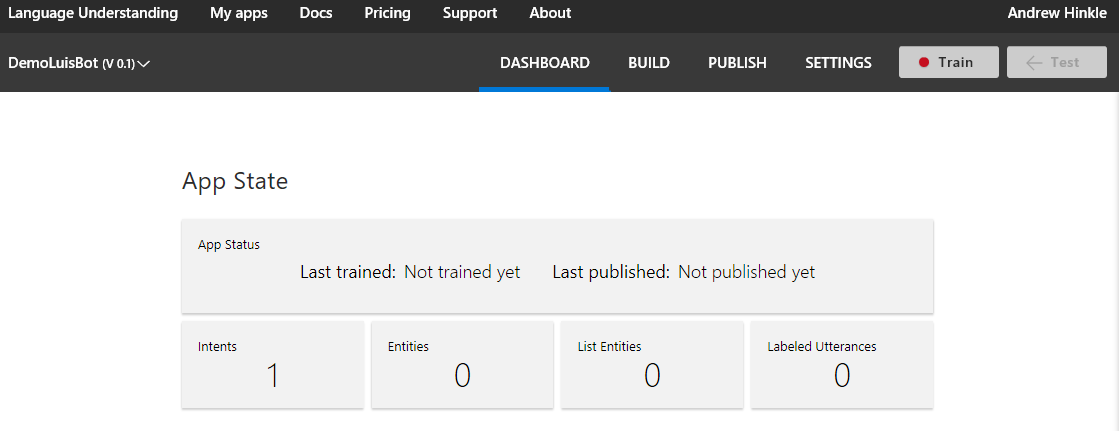


1. Fill out form
   1. Name: DemoLuisBot
   2. Culture: English
   3. Description: Demo
2. Click Done

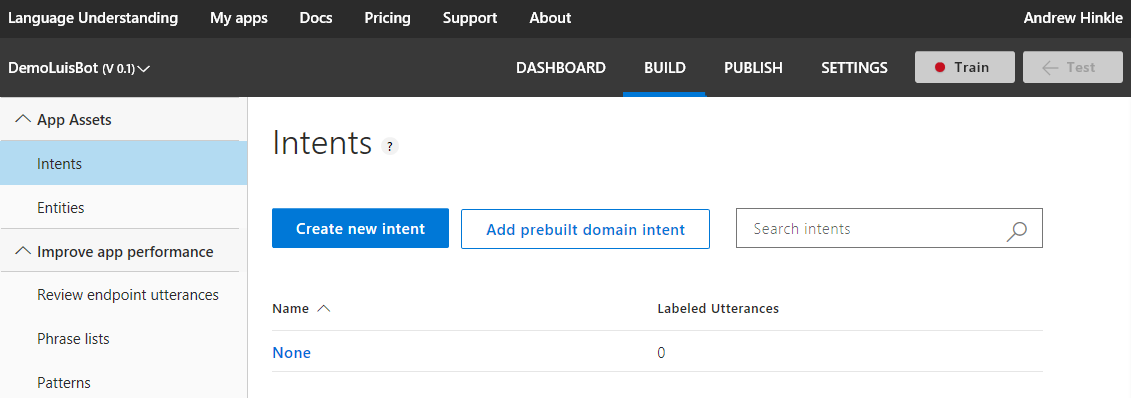


# Luis Overview

## Dashboard



## Build



1. Intents: An action you want to process
   1. Ex. Search, Define, Get status, Change status, etc.
2. Entities: Variables
3. Utterances: A phrase associated with the intent and a variable
4. Phrase Lists: Fine tune the decisions for choosing the correct intent and entities by giving specific examples of utterances.
5. Prebuilt domains (preview): Sample intents, utterances, and entities for common domains such as calendars, entertainments, gaming, reminders, places, web navigation, etc.

## Publish

1. Publish App: Publish your LUIS service to Staging or Production

## Settings

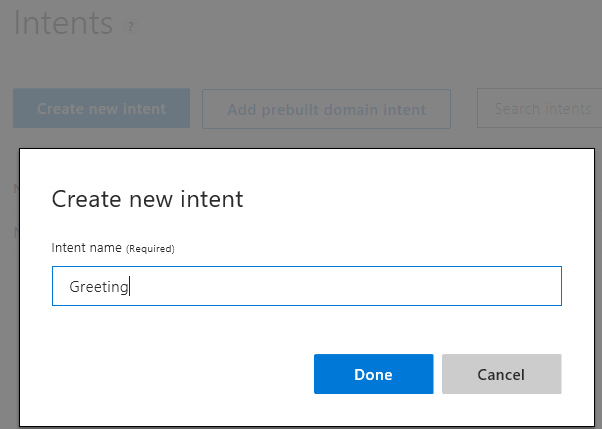
1. Application ID, display name, admin, collaborators, import new versions

## Train

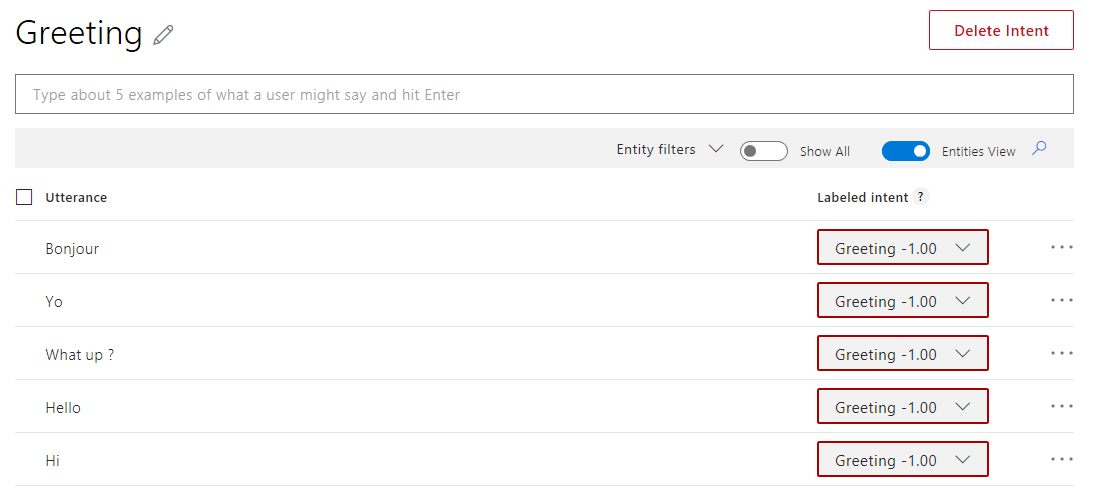
1. Train & Test: Enter some utterances to verify you have trained it properly

# Create an intent, utterance, and entity

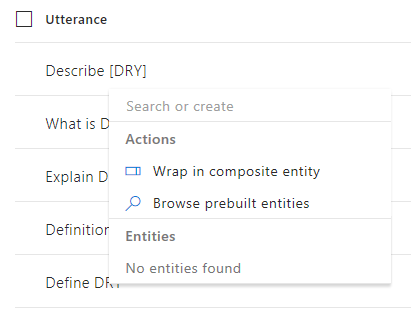
1. Click Build > Intents > Create new intent > Intent name: "Greeting" > Done

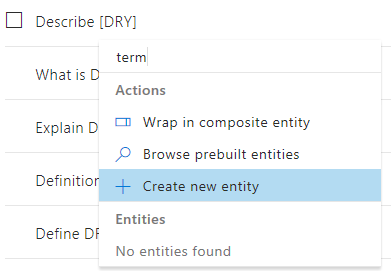


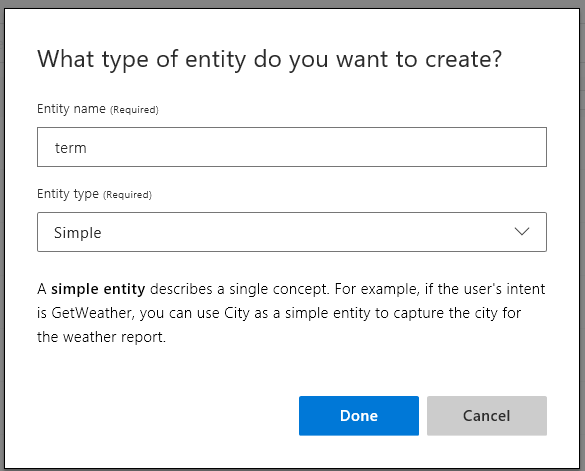
1. Type the following utterances and press enter after each one
   1. Hi
   2. Hello
   3. What up?
   4. Yo
   5. Bonjour

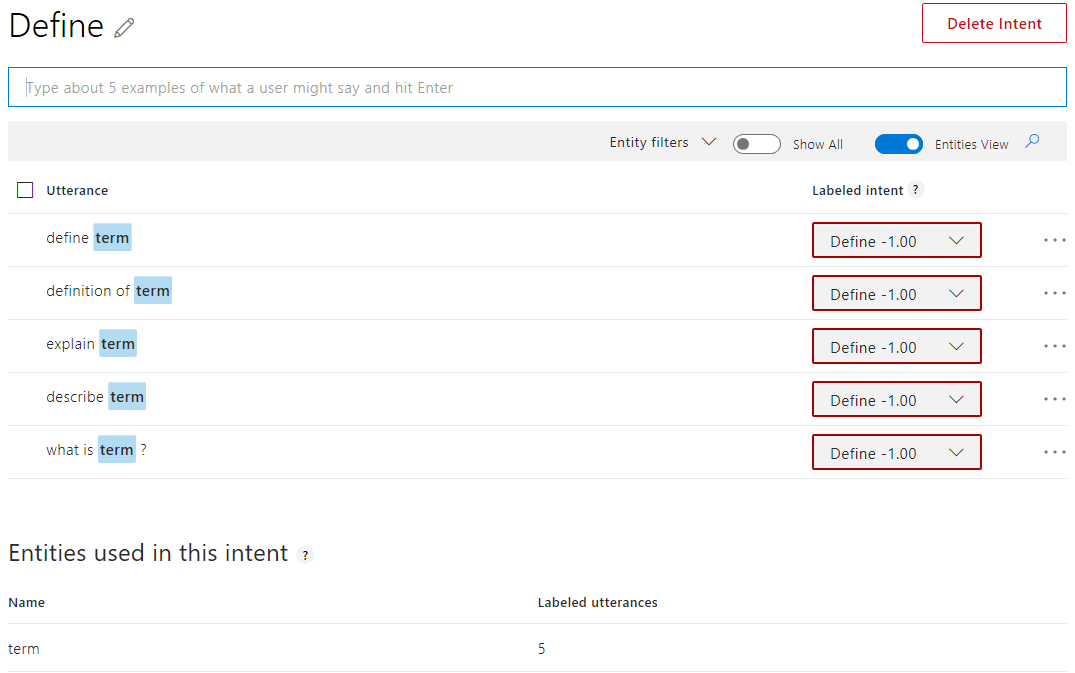


1. Click Intents > Add Intent > Intent name: "Define" > Save
2. Type the following utterances and press enter after each one
   1. Define DRY
   2. Definition of DRY
   3. Explain DRY
   4. What is DRY?
   5. Describe DRY
3. For each utterance, select "dry" and then click "term", so they look as follows. The first entry, in the Search or create field you'll need to type "term".
   1. Define [term]
   2. Definition of [term]
   3. Explain [term]
   4. What is [term]?
   5. Describe [term]
   6. You can select multiple words by clicking the first word and then the last word. If you have multiple words selected and you want to unselect, then click an empty spot on the page.



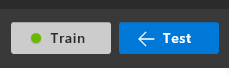




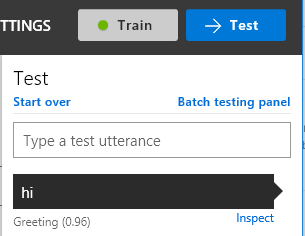


## Train and Test

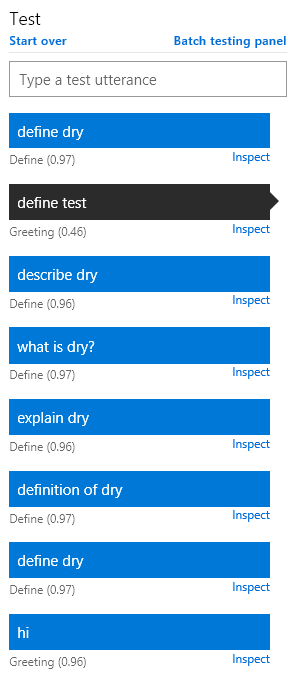
1. Click the Train button
   1. The Train button indicator will turn from red to green
2. Click the Test button



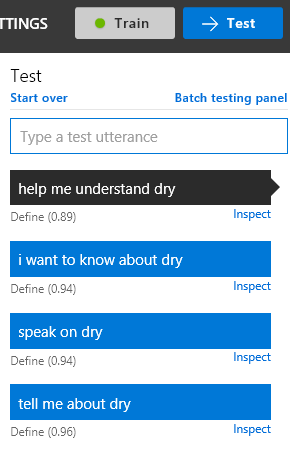
1. Type "hi" and verify the Intent is "Greeting"



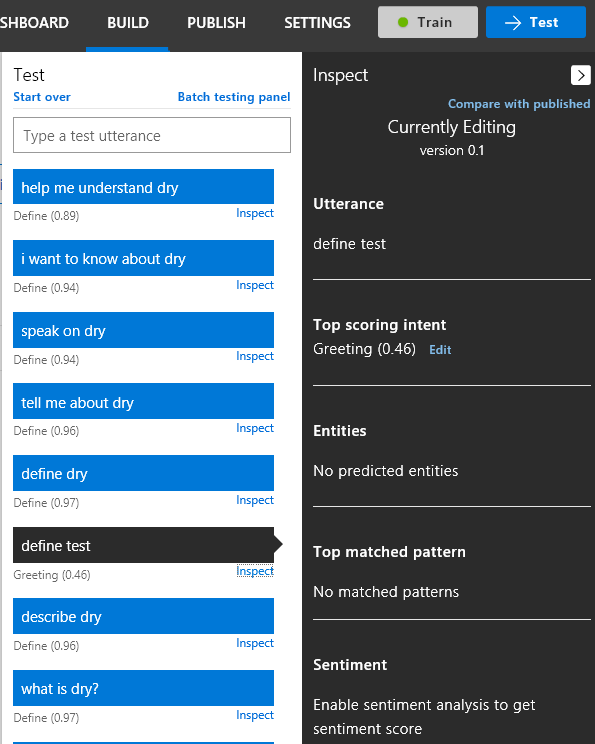
1. Try out each utterance to see if the intent is "Define" and entity is [$term]
   1. Define DRY
   2. Definition of DRY
   3. Explain DRY
   4. What is DRY?
   5. Describe DRY
   6. Define test
   7. Define DRY



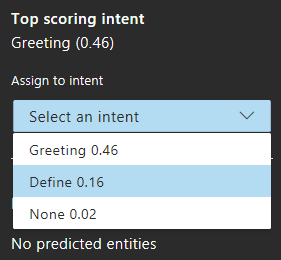
1. Notice the "define test" case did not work as expected. It was assigned to greeting. We'll come back to that case.
2. What about some phrases we didn't add as utterances?
3. Tell me about DRY
4. Speak on DRY
5. I want to know about DRY
6. Help me understand DRY



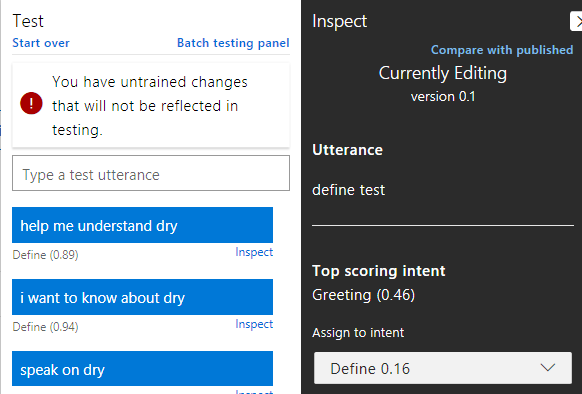
1. All of these phrases work just as well. Keep in mind we only have one intent at this time, so after adding more intents you may need to fine toon the process under Phrase Lists and Patterns
2. So why did the "define test" return the intent of "Greeting"? Click Inspect next to the test to get more information



1. The top scoring intent was "Greeting (0.46)"
2. Click Edit and select "Define"

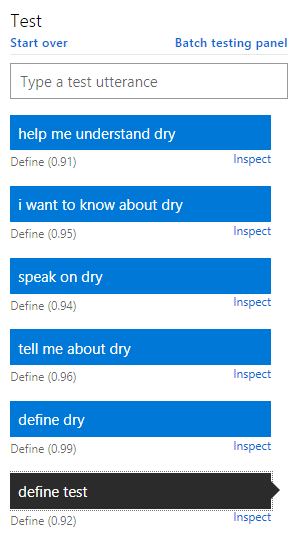


1. Now you have untrained changes, click Train

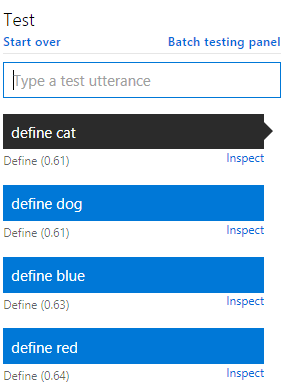




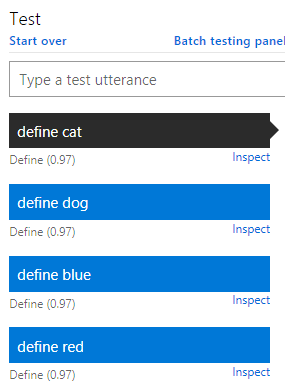
1. Under Test click the "Start over" link and rerun your tests



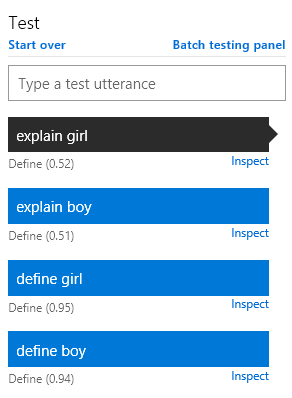
1. Now that's better, but we should do some more testing with different words
   1. Define red
   2. Define blue
   3. Define dog
   4. Define cat



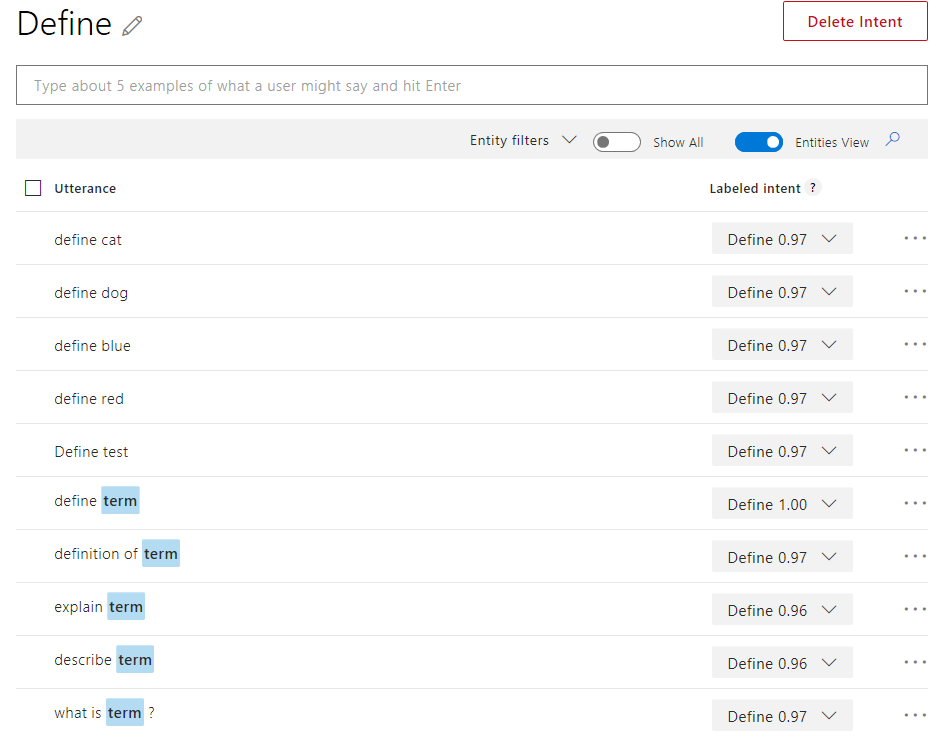
1. Notice that the bot is only sure roughly around 60% of the time
2. For each of these cases
   1. Click Inspect
   2. Assign the "Define" intent
   3. Train
   4. Rerun the test



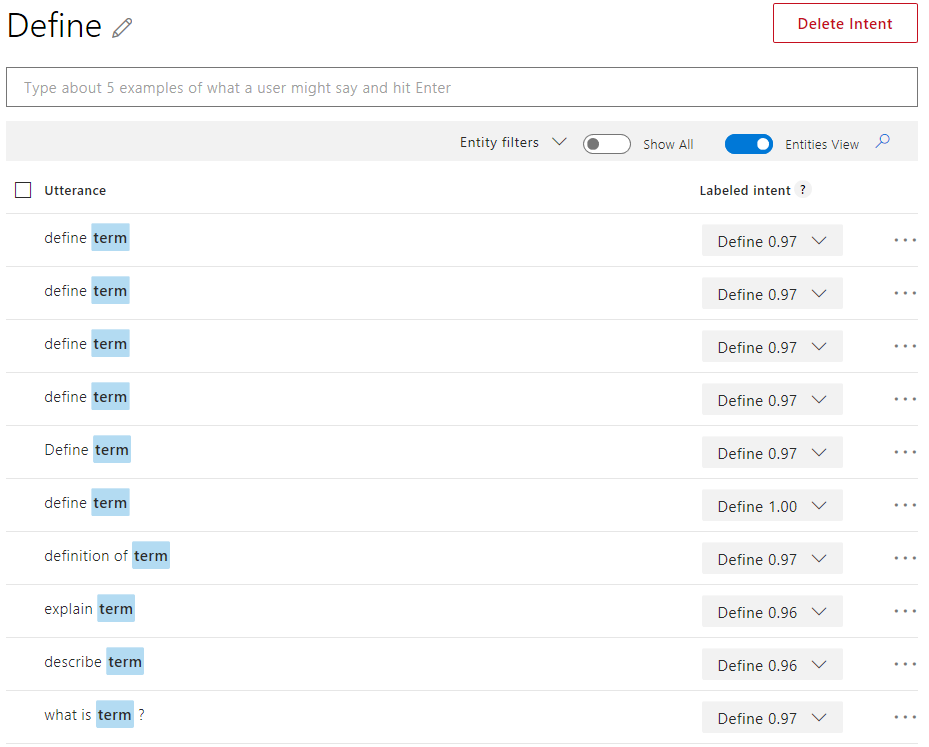
1. That's great so try a couple variations
   1. Define boy
   2. Define girl
   3. Explain boy
   4. Explain girl



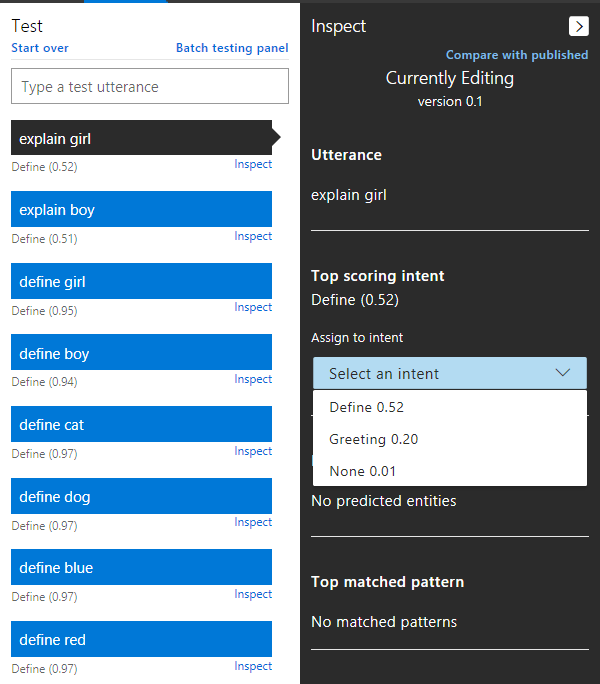
1. We only improved the "Define" utterance, not the "Explain" utterance
2. What did our changes earlier do? Let's take a look under "Intents" > "Define"



1. The other "define" utterances we added did not reference the term. Do that now.



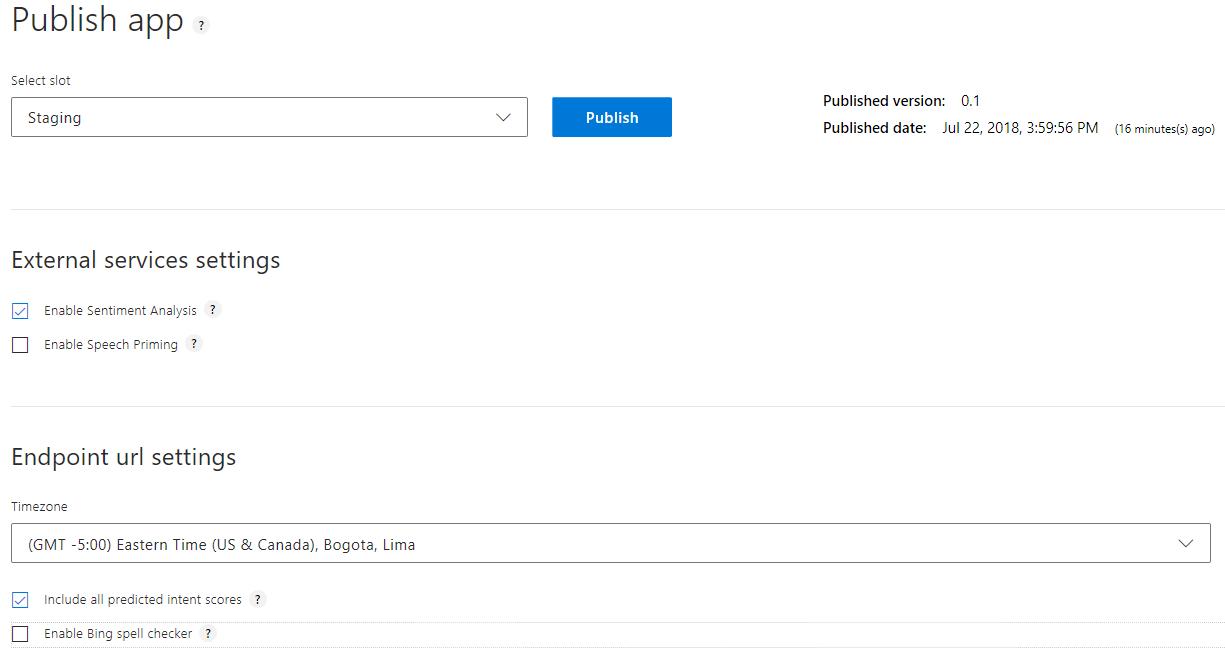
1. Train and Test just the last entries for now
   1. Define red
   2. Define blue
   3. Define dog
   4. Define cat
   5. Define boy
   6. Define girl
   7. Explain boy
   8. Explain girl



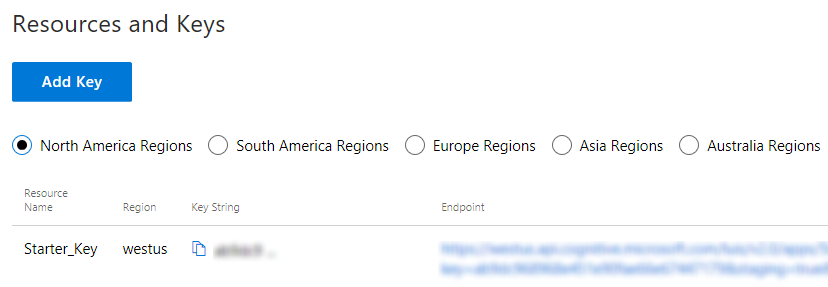
1. Our intent is getting confused with Greeting causing the confidence level to go down
2. Obviously more work can be done to tighten up the expectations and training, but this is good enough for the demo to move on
3. Now our confidence level is much higher
   1. Remember to test, adjust, train, test frequently until you get the results you want
   2. On a regular basis retrain and retest to ensure you still get the expected results

# Publish App and Test

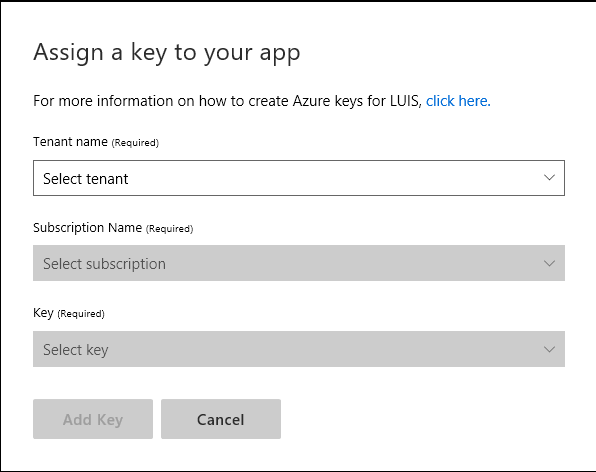
1. Click Publish
2. Fill out the form
   1. Publish app
      1. Select a slot (Environment): Staging
         1. Production requires a Paid Tier
   2. External services settings
      1. Enable Sentiment Analysis: Check
      2. Enable Speech Priming: Uncheck
         1. This is just testing, so no need to populate this test information to the Speech platform, because it won't help anyone
   3. Endpoint url settings
      1. Timezone: Eastern Time
      2. Include all predicted intent scores: Check
      3. Enable Bing spell checker: Uncheck
         1. Requires a Bing spell check API key

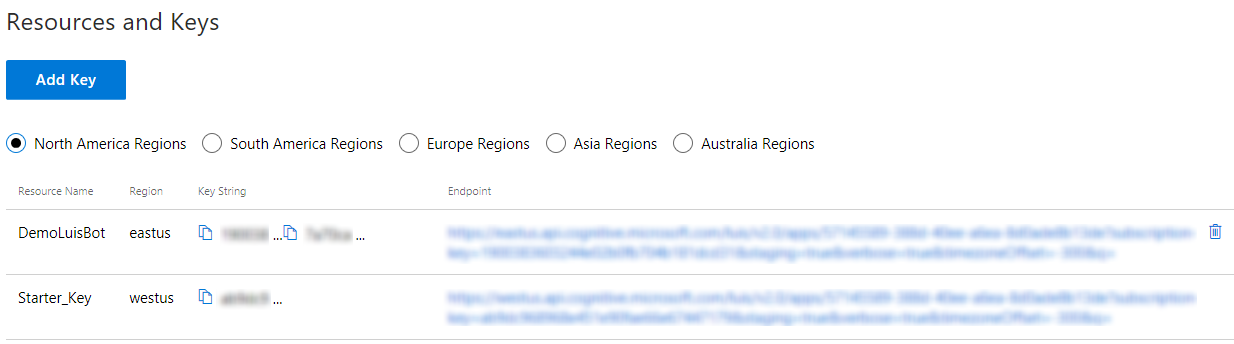


* 1. Resources and Keys
     1. Regions: North America Regions



* + 1. We don't want to limit ourselves to the starter key, so we'll add the app key we created in Azure.
    2. Click "Add Key"
       1. Note: if the tenant drop down is empty the first time like it did for me, try signing out and signing back in. Refreshing the page may also help.
    3. Fill out the form
       1. Tenant
          1. For testing use your MSDN email address, however, for production use the organization's Azure account
          2. {[Your organization id associated with your Azure account](https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-resources-faq)}
       2. Subscription: Visual Studio Ultimate with MSDN
          1. Or whatever subscription you have
       3. Key: DemoLuisBot – (eastus)
          1. It references the name of the key, not the actual key, so if the key's change in the future presumably you don't have to update the key as well.





1. Click Publish
2. Copy the Endpoint url to notepad and add "Define DRY" to the end
   1. … q=Define DRY
3. Copy the update url to a browser and run it. You'll get a response like the following.

{

"query": "define dry",

"topScoringIntent": {

"intent": "Define",

"score": 0.9979334

},

"intents": [

{

"intent": "Define",

"score": 0.9979334

},

{

"intent": "Greeting",

"score": 0.00362568558

},

{

"intent": "None",

"score": 0.003582179

}

],

"entities": [

{

"entity": "dry",

"type": "term",

"startIndex": 7,

"endIndex": 9,

"score": 0.9962388

}

],

"sentimentAnalysis": {

"label": "negative",

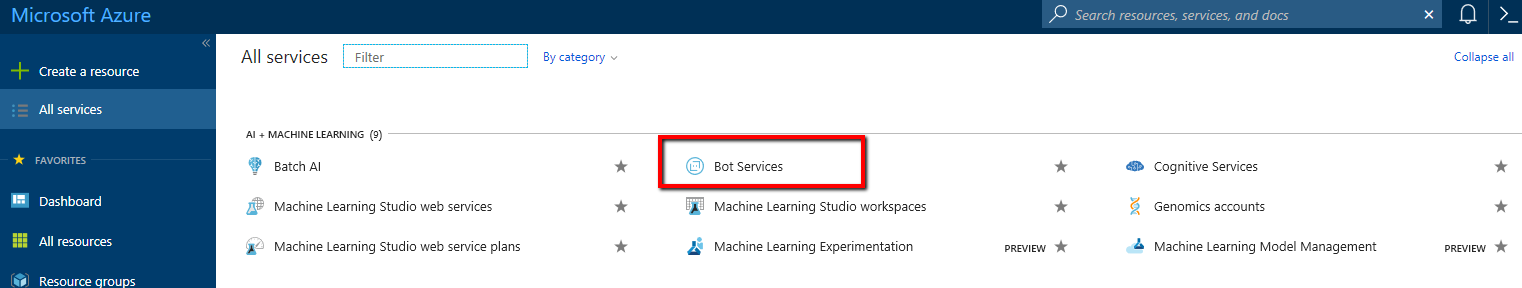
"score": 0.22325629

}

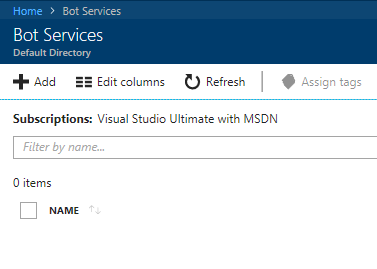
}

# Create a Web App Bot

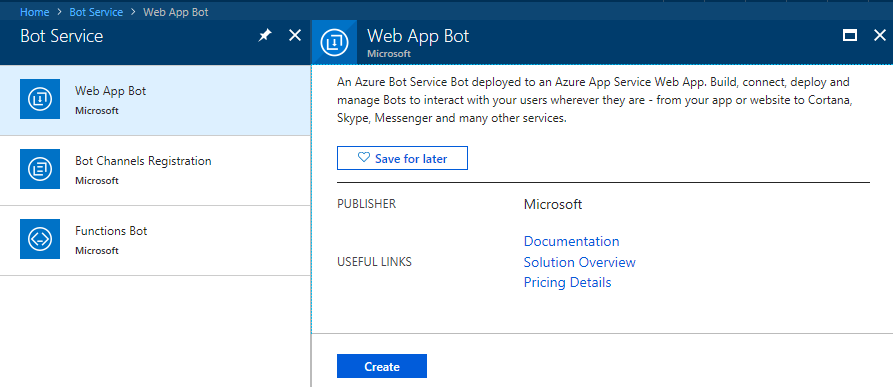
1. <https://portal.azure.com>
2. Sign in with your MSDN account
3. Click All services > Bot Services



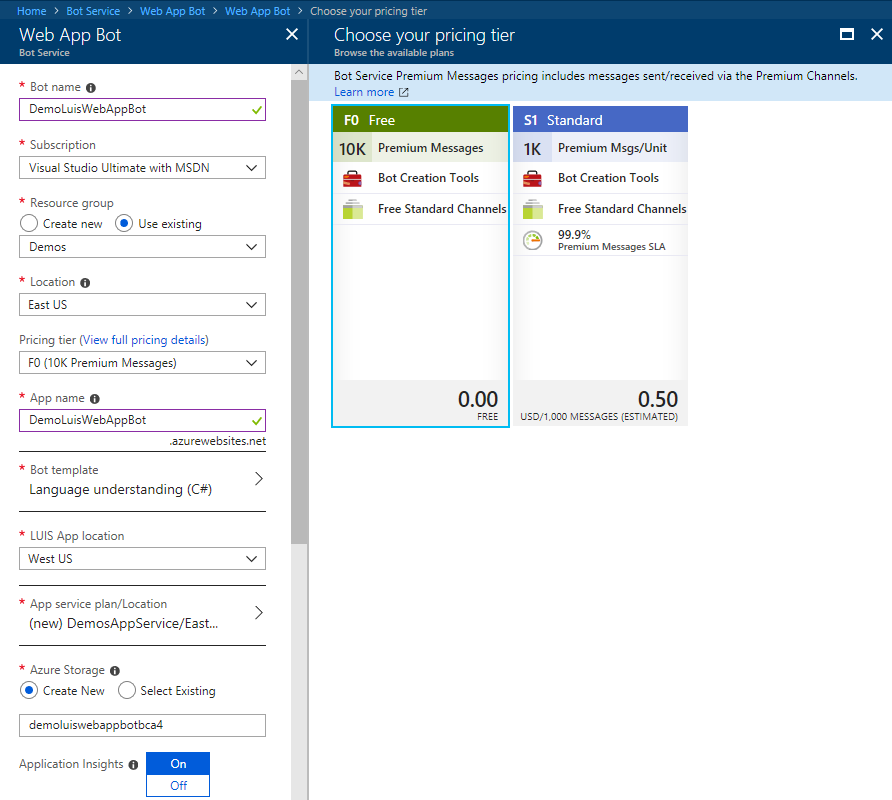
1. Click Add

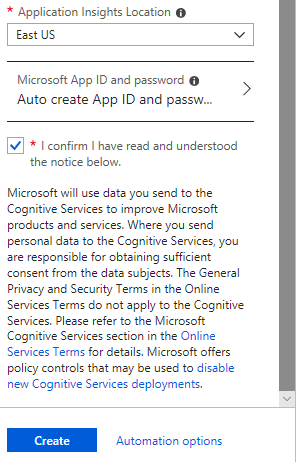


1. Click Web App Bot > Create

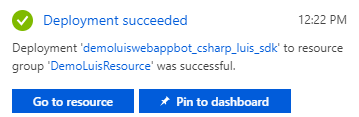


1. Fill out the form
   1. Bot name: DemoLuisWebAppBot
   2. Subscription: Visual Studio Ultimate with MSDN
   3. Resource group: Create: DemoLuisResource
   4. Location: East US
   5. Pricing Tier: F0 (10K Premium Messages)
   6. App name: DemoLuisWebAppBot
      1. .azurewebsites.net
   7. Bot template: Language understanding (C#)
   8. LUIS App Location: West US
   9. App service plan/Location:
      1. Create New:
         1. App service plan name: DemoLuisAppServicePlan
         2. Location: East US
         3. Click Create
            1. Defaults to S1 pricing plan. We'll change it to free soon
   10. Azure Storage:
       1. Create New: demoluiswebappbotad90
          1. Auto populated; Maybe different for you
   11. Application Insights: On
   12. Application Insights Location: East US
   13. Microsoft App ID and password: Auto create App ID and password
   14. I confirm: Check
       1. Read the terms and conditions
2. Click Create

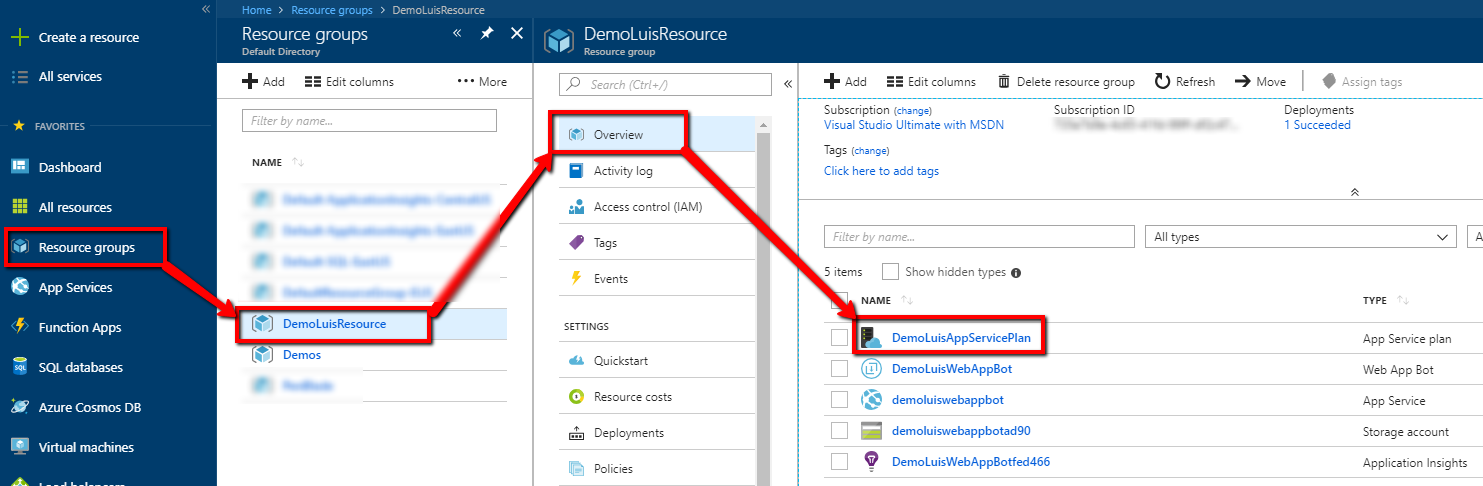




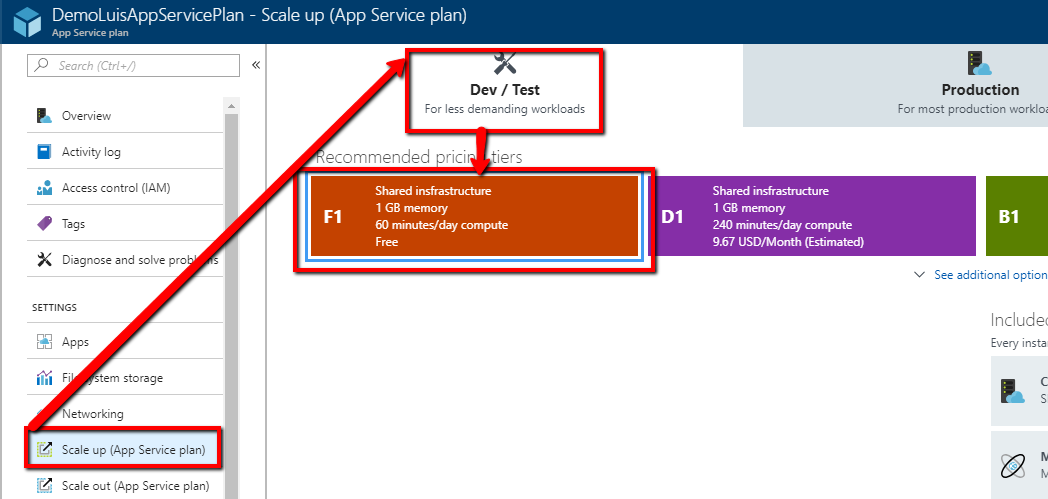
1. When the resource is created click the "Pin to dashboard"



1. From the main menu > Click Resource Groups > DemoLuisResouce > Overview > DemoLuisAppServicePlan



1. Click "Scale up (App Service plan)" > "Dev/ Test" tab > F1 (Shared infrastructure, 1GB memory, 60 minutes/day compute/Free) > Apply

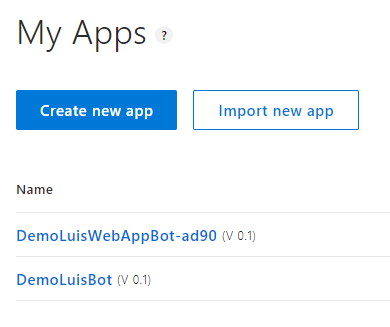


# Move the cognitive services into the new resource group

1. From the main menu > Click Resource Groups > Demos > check "DemoLuisBot" (type: Cognitive Services) > move to "DemoLuisResource"
   1. Because I moved the cognitive services to a different resource group, anything that referenced it must be updated. In this case that meant the luis.ai below.

# New Luis App

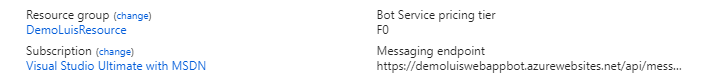
1. <https://www.luis.ai>
2. Login
3. A new luis app was created and bound to the web app bot



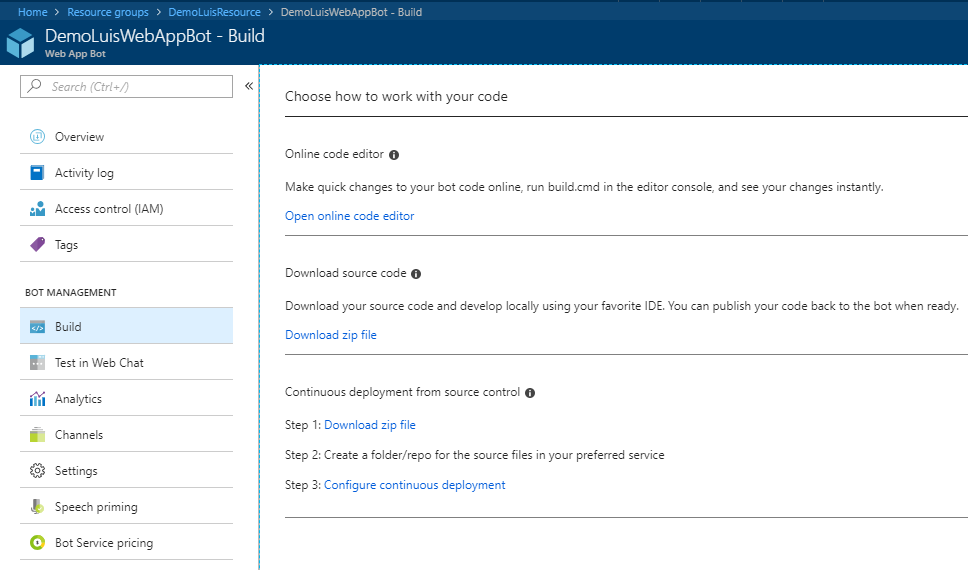
1. Repeat the steps adding intents, entities, utterances, etc. for the "define" intent performed during the luis.ai setup
   1. Greetings is done for you as part of the template
2. If I didn't move the cognitive resources to a different resource group, then it would have already been linked. Since I did move the resources, I followed the same steps to reassign the cognitive resources to the luis app.

# Get the Web App Bot source code

1. From the main menu > Click Resource Groups > DemoLuisResouce > Overview > DemoLuisWebAppBot (type: Web App Bot)
2. Copy the Messaging endpoint
   1. Ex. <https://demoluiswebappbot.azurewebsites.net/api/messages>



1. Click Build > Under "Continuous deployment from source control" > Download zip file



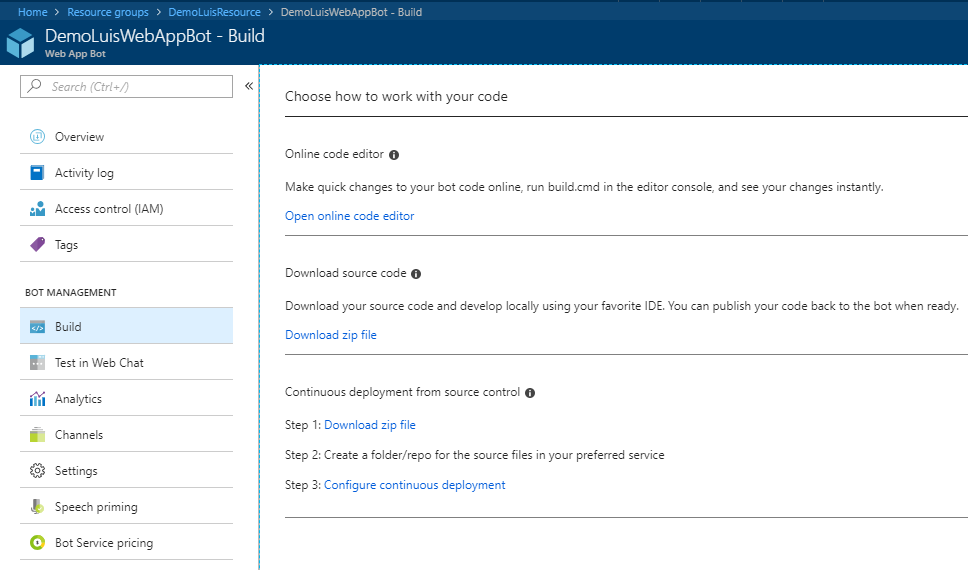
# (Skip) Add the Web App Bot to source control

1. Add it to your source control (VSTS, TFS, or GitHub)
   1. I used VSTS for this demo, however, if you want to setup continuous deployment, you'll need to use GitHub or VSTS Git
   2. Don't change anything yet, just get the initial download checked in

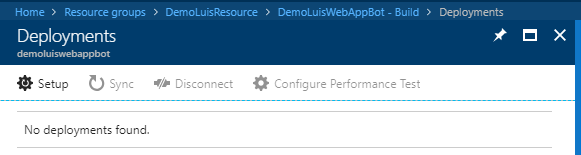
# (Skip) Setup continuous deployment for the Web App Bot

*Continuous deployment using GitHub is supported. Continuous deployment using Visual Studio only supports VSTS Git.*

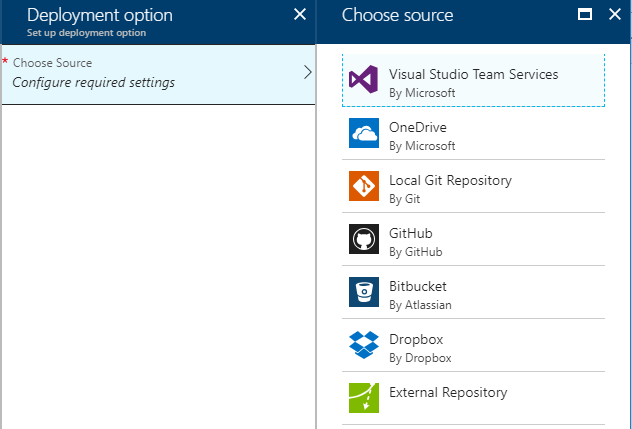
1. From the main menu > Click Resource Groups > DemoLuisResouce > Overview > DemoLuisWebAppBot (type: Web App Bot) > Click Configure continuous deployment



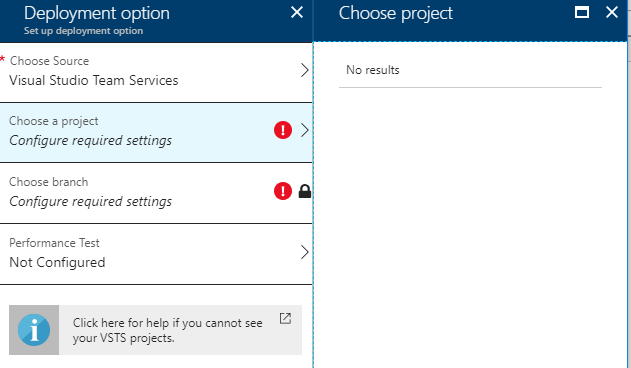
1. Click Setup



1. Click "Choose Source" > Choose your source control (Ex. VSTS)

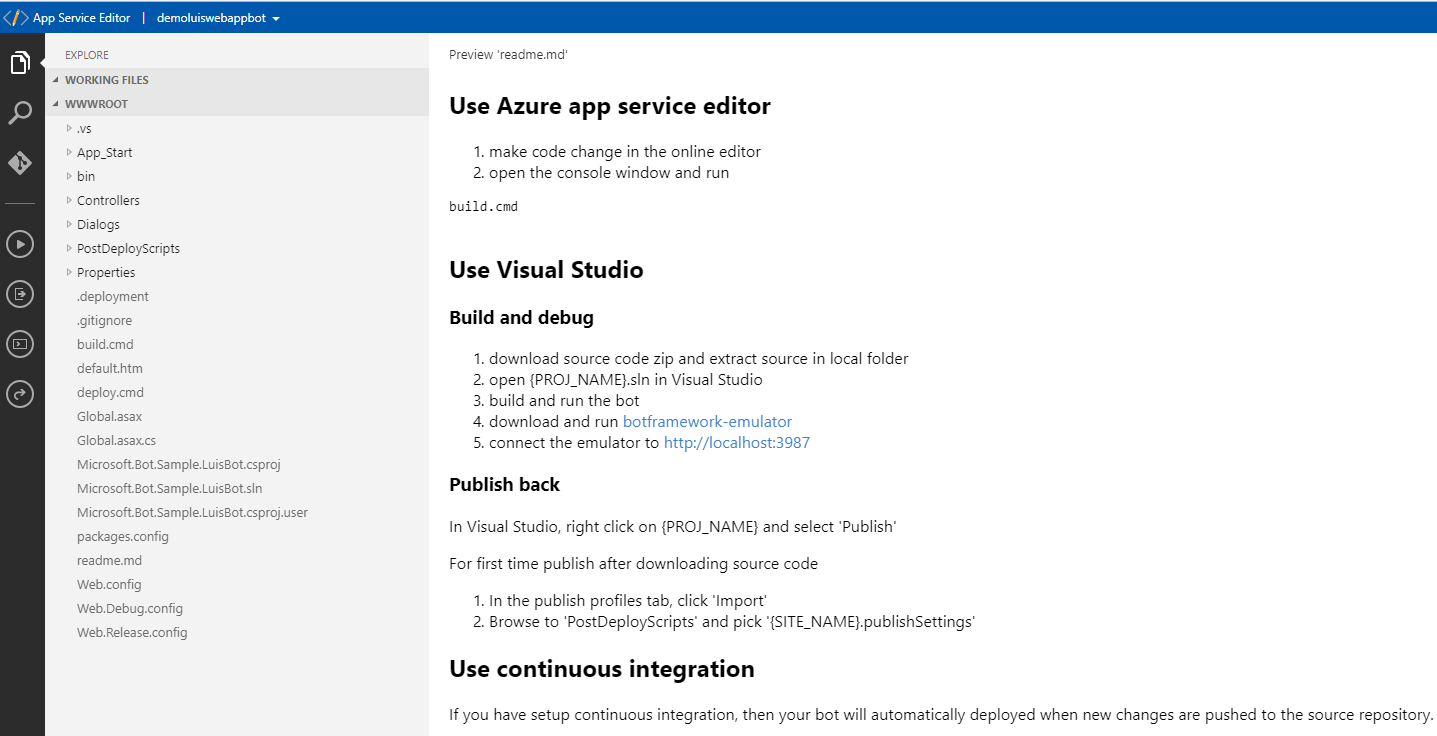


1. Click "Choose a project"
   1. If you don't see your projects you may need to link the VSTS Git project or Authorize GitHub access. For now we'll just use the online editor. Here are some additional documentation on setting up continuous deployment.
      1. [Setting up a VSTS account so it can deploy to a Web App](https://github.com/projectkudu/kudu/wiki/Setting-up-a-VSTS-account-so-it-can-deploy-to-a-Web-App)
      2. [Set up continuous deployment](https://docs.microsoft.com/en-us/azure/bot-service/bot-service-build-continuous-deployment?view=azure-bot-service-3.0)



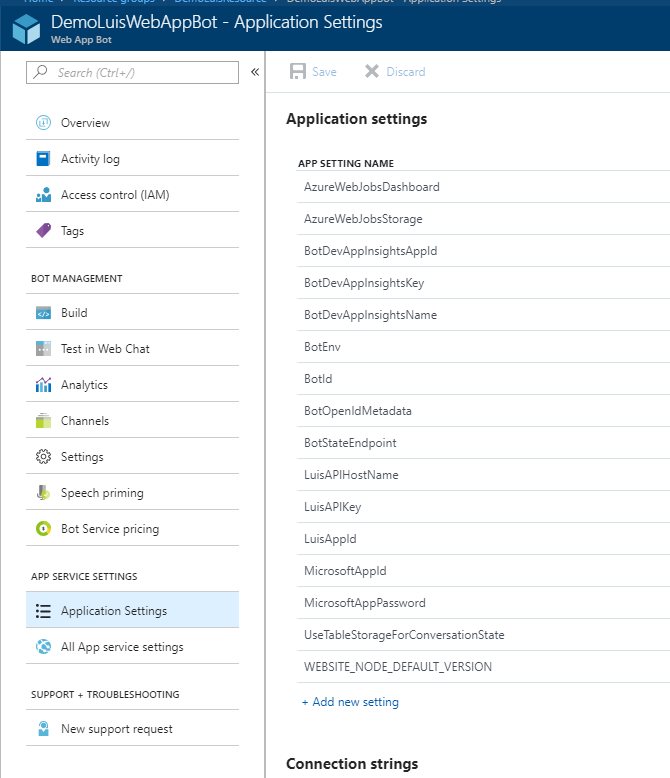
# Review the Web App Bot source code online for links

1. From the main menu > Click Resource Groups > DemoLuisResouce > Overview > DemoLuisWebAppBot (type: Web App Bot)
2. Click "Open online code editor"



# Get your Application Settings

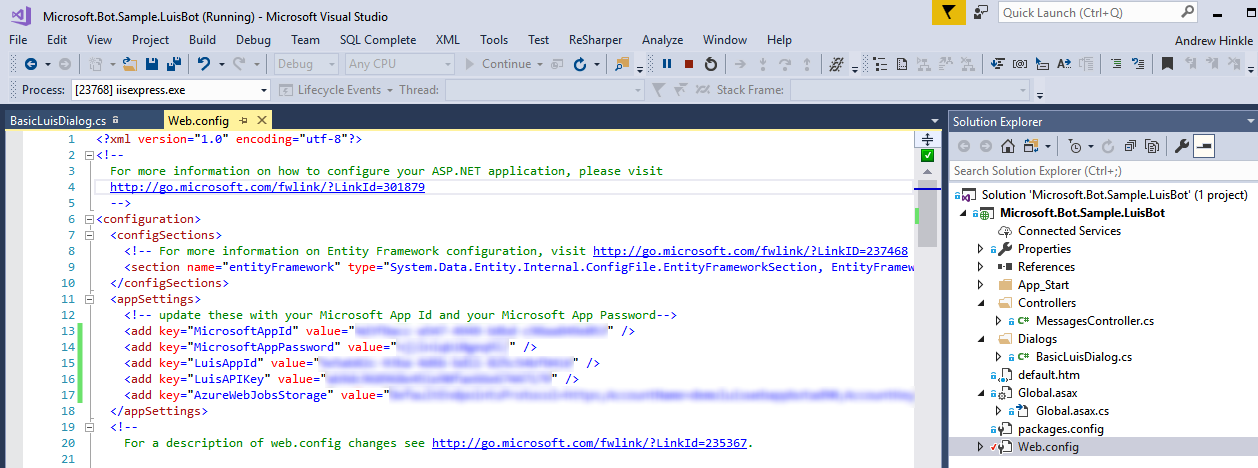
1. From the main menu > Click Resource Groups > DemoLuisResouce > Overview > DemoLuisWebAppBot (type: Web App Bot) > Click Application Settings



1. Record the following information
   1. MicrosoftAppId
   2. MicrosoftAppPassword
   3. LuisAppId
   4. LuisAPIKey
   5. AzureWebJobsStorage

# Setup the LuisBot solution

1. Extract the source zip if you haven't already
2. Open the solution in Visual Studio



Add the following entries

<appSettings>

<!-- update these with your Microsoft App Id and your Microsoft App Password-->

<add key="MicrosoftAppId" value="{Your MicrosoftAppId}" />

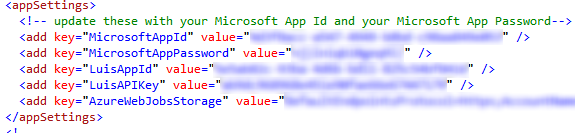
<add key="MicrosoftAppPassword" value="{Your MicrosoftAppPassword}" />

<add key="LuisAppId" value="{LuisAppId}" />

<add key="LuisAPIKey" value="{LuisAPIKey}" />

<add key="AzureWebJobsStorage" value="{AzureWebJobsStorage}" />

</appSettings>



1. Update BasicLuisDialog.cs

using System;

using System.Configuration;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.Bot.Builder.Dialogs;

using Microsoft.Bot.Builder.Luis;

using Microsoft.Bot.Builder.Luis.Models;

namespace Microsoft.Bot.Sample.LuisBot

{

// For more information about this template visit http://aka.ms/azurebots-csharp-luis

[Serializable]

public class BasicLuisDialog : LuisDialog<object>

{

public BasicLuisDialog() : base(new LuisService(new LuisModelAttribute(

ConfigurationManager.AppSettings["LuisAppId"],

ConfigurationManager.AppSettings["LuisAPIKey"],

domain: ConfigurationManager.AppSettings["LuisAPIHostName"])))

{

}

[LuisIntent("None")]

public async Task NoneIntent(IDialogContext context, LuisResult result)

{

await this.ShowLuisResult(context, CreateBasicMessage(result));

}

// Go to https://luis.ai and create a new intent, then train/publish your luis app.

// Finally replace "Greeting" with the name of your newly created intent in the following handler

[LuisIntent("Greeting")]

public async Task GreetingIntent(IDialogContext context, LuisResult result)

{

await this.ShowLuisResult(context, "Welcome to the Luis.ai Bot demo.");

}

[LuisIntent("Cancel")]

public async Task CancelIntent(IDialogContext context, LuisResult result)

{

await this.ShowLuisResult(context, CreateBasicMessage(result));

}

[LuisIntent("Help")]

public async Task HelpIntent(IDialogContext context, LuisResult result)

{

await this.ShowLuisResult(context, CreateBasicMessage(result));

}

[LuisIntent("Define")]

public async Task DefineIntent(IDialogContext context, LuisResult result)

{

var term = result?.Entities?.SingleOrDefault(x => x.Type == "term")?.Entity;

string message;

switch (term?.ToLower())

{

case "test":

message = "This is a test.";

break;

case "dry":

message = "DRY: Do not Repeat Yourself.";

break;

default:

message = CreateBasicMessage(result);

break;

}

await this.ShowLuisResult(context, message);

}

private static string CreateBasicMessage(LuisResult result)

{

var query = result?.Query;

var intent = result?.Intents?.FirstOrDefault()?.Intent;

var entity = result?.Entities?.FirstOrDefault()?.Entity;

var message = $"Query: {query}; " +

$"Intent: {intent}; " +

$"Entity: {entity}; ";

return message;

}

private async Task ShowLuisResult(IDialogContext context, string message)

{

await context.PostAsync(message);

context.Wait(MessageReceived);

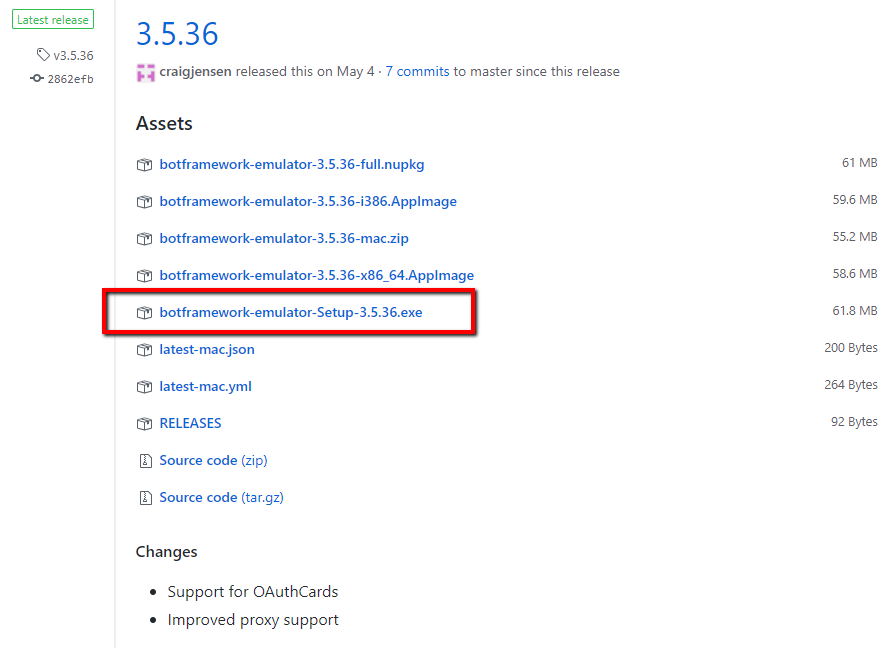
}

}

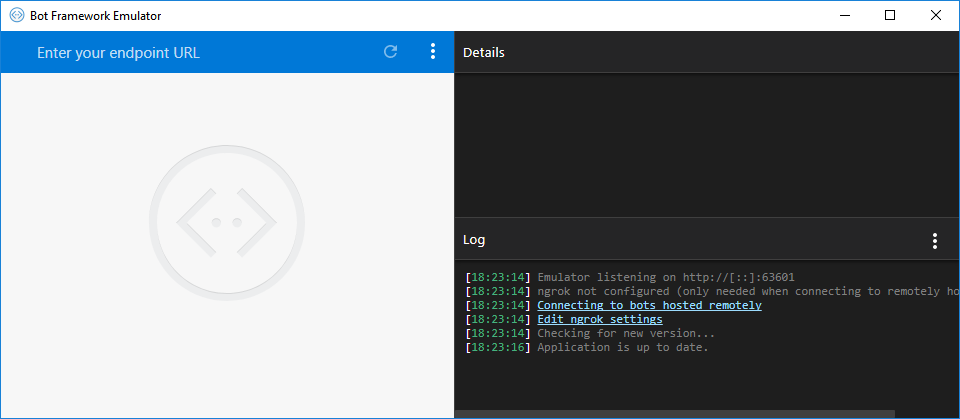
}

# Setup the Bot Framework Emulator

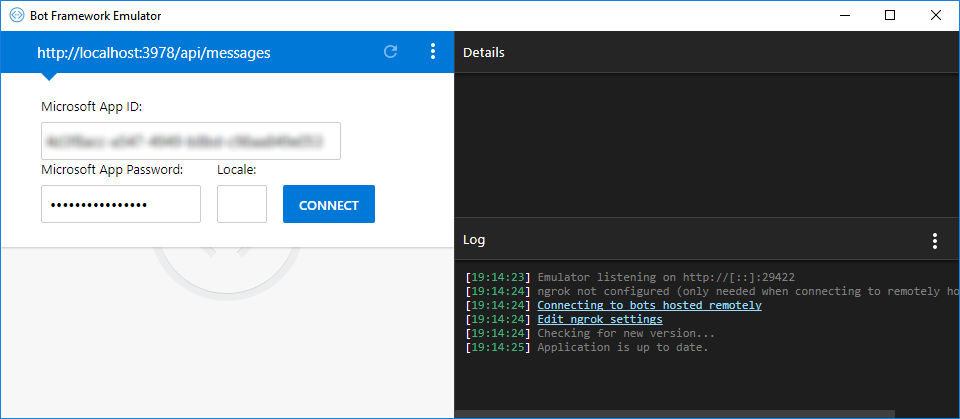
1. <https://github.com/Microsoft/BotFramework-Emulator/releases>
2. Scroll down to the "Latest release" > click "botframework-emulator-Setup-#.#.#.exe



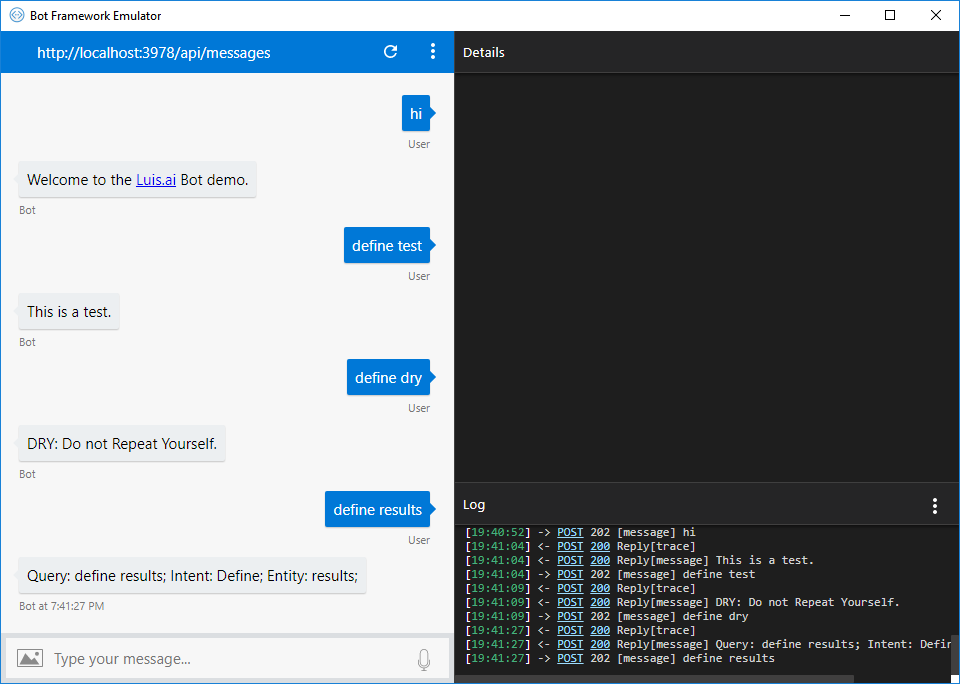
1. Run the exe



1. Fill in the same MicrosoftAppID and MicrosoftAppPassword, leave locale empty



1. Type:
   1. Hi
   2. Define test
   3. Define dry
   4. Define results

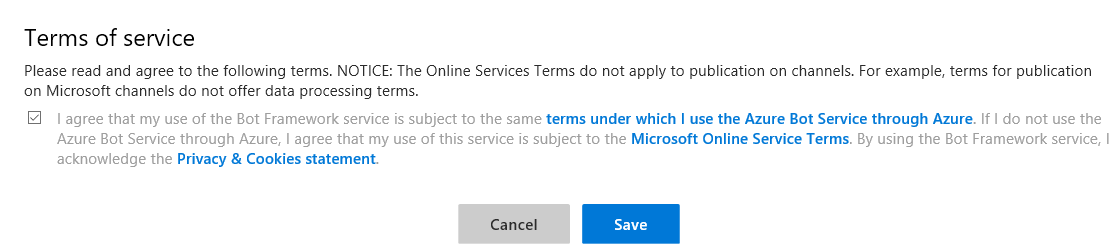


# Create an App Service

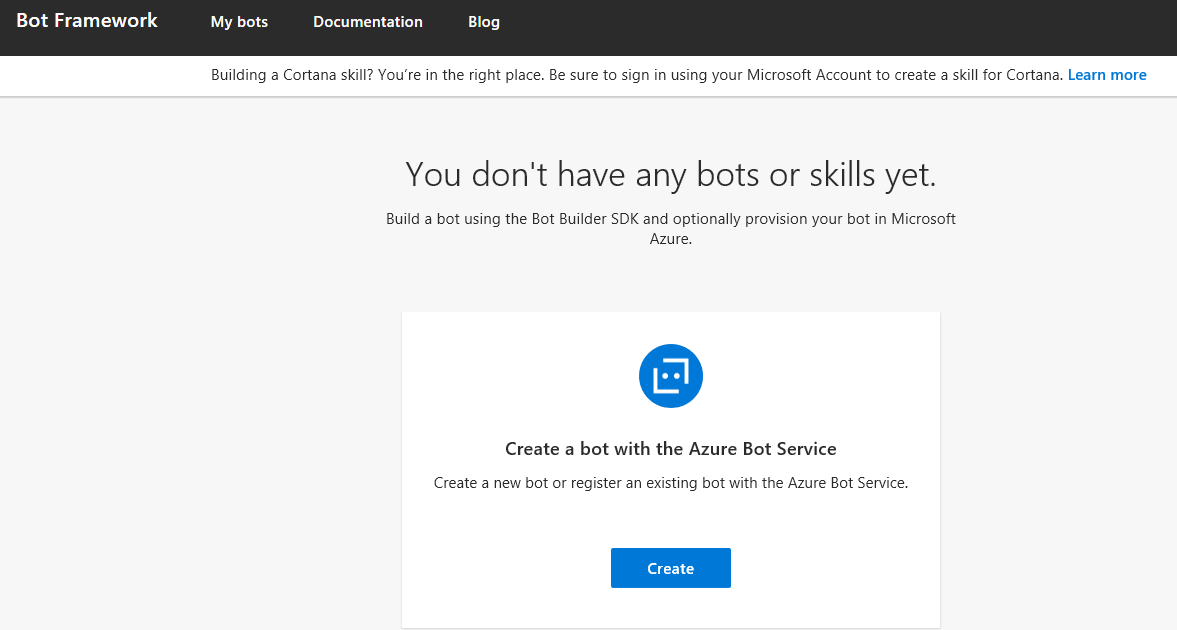
1. <https://portal.azure.com>
2. Sign in with your MSDN account
3. App Services > Add > Filter: app service > App Service Environment > Create
4. Fill out the form
   1. Name: DemoAppService
   2. Subscription:

# Create a Bot in the botframework

1. <https://dev.botframework.com/>
2. Sign in with your MSDN account
3. Agree to the Terms of service
   1. Read them, understand them



1. Click My bots > Create



1. You are redirected to portal.azure.com

# OLD NOTES

* 1. Hosting Plan: {App Service Plan}

1. Choose a template
   1. C#
   2. Language understanding
   3. Next
2. Create a Microsoft App ID
   1. Click Create a Microsoft App ID
   2. Sign in with your MSDN account
   3. Record the App name and App ID
   4. Click Generate an app password to continue
   5. Record the App password
   6. Click Finish and go back to Bot Framework
      1. If you actually get redirected back to the Bot Framework, instead go back to your portal.azure.com tab
3. Paste the App password
4. Check the two terms of use check boxes
   1. Be aware the terms are the same as Cognitive Services as mentioned above
5. Click Create Bot
6. Follow the on-screen instruction to create, register, and deploy the bot
   1. It will take a few minutes
7. Choose how to work with your code
   1. Continuous deployment from source control
   2. Fill out the form
      1. Source control provider: Visual Studio Team Services
      2. Url to your Visual Studio Team Services site
         1. Click View Visual Studio Profile to open your VSTS site, so you can copy the url for the account you want to use
         2. Ex. https://something.visualstudio.com
      3. Access Token
         1. [Create an access token](https://docs.microsoft.com/en-us/vsts/accounts/use-personal-access-tokens-to-authenticate) and assign it "Project and team (read, write, and manage" and "Code (full)" scope.
         2. Record the token
      4. Click Enable
      5. Click Got it
8. Continuous deployment from source control
   1. Click Configure continuous deployment
      1. Doesn't look like you need to do anything.
9. We'll download the source locally and open in VS2017 later.

# Verify your bot was registered

1. <https://dev.botframework.com/>
2. Sign in with your MSDN account > Click My bots
3. Your bot, {MyBot}, should be in the list

# Troubleshooting – 401 access denied error

The bot service is creating the Luis.ai service for you, so these steps are done for you automatically now. While you can still create your own Luis.ai service with these steps, you would need to update the bot application settings with the new LuisAppId and LuisAPIKey. However, I had nothing but problems with this as I got 401 access denied errors.

Download the source and update

# Get the Credentials

1. <https://portal.azure.com>
2. On the dashboard click on your {MyBot} > Click on Settings > Application settings: click Open > scroll down to App settings
3. Record the following credentials
   1. **MicrosoftAppId**: {guid}
   2. **MicrosoftAppPassword**: {password}
   3. **LuisAppId**: {guid}
   4. **LuisAPIKey**: {password}

# Get the Bot's solution and open in Visual Studio 2017

Assumption: You've already connected to your visualstudio.com site in VS2017. <https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-csharp-tutorial-build-bot-framework-sample>

1. Open VS2017 > Team Explorer > Manage Connections > Manage Connections > Connect to a Project > Expand the list to your Bot project > Click Connect
2. In the Solution Explorer double-click your .sln file

# Update the web.config with your credentials



# Add methods to the BasicLuisDialog.cs



Bind to the Luis.ai service

# Get the localhost url

1. Open Visual Studio > connect to the GitHub repository
2. Run the Microsoft.Bot.Sample.LuisBot.sln
3. Record the localhost url noting the port number
   1. Ex. <http://localhost:3979/api/messages>

# Setup the BotFramework Emulator

1. <https://github.com/Microsoft/BotFramework-Emulator/>
2. Download
   1. <https://github.com/Microsoft/BotFramework-Emulator/releases>
      1. [botframework-emulator-Setup-3.5.31.exe](https://github.com/Microsoft/BotFramework-Emulator/releases/download/v3.5.31/botframework-emulator-Setup-3.5.31.exe)
3. Install the EXE
4. Run the Emulator from the shortcut added to your desktop
5. [Fill out the form](https://github.com/Microsoft/BotFramework-Emulator/wiki/Getting-Started#connect-to-an-azure-bot-service)
   1. Uri: {See previous sections}
      1. Ex. http://localhost:3979/api/messages
      2. Enter your endpoint URL from your localhost with the correct port
   2. Microsoft App ID: {See previous sections}
   3. Microsoft App Password: {See previous sections}

Debug

# Local

1. In VS2017 add breakpoints in your NoneIntent method and DefineIntent method
2. Run the application in debug mode (F5)
3. Run the Bot Emulator
4. Type: hi
   1. Response: Query: hi; Intent: Greeting; Entity: ;
5. Type: define dry
   1. Response: Query: define dry; Intent: Define; Entity: dry;

# Commit and Sync code

1. If everything is working up to this point, then commit your changes and sync the code
2. If the continuous deployment is setup correctly to your TFS GitHub repository, then the bot should have also been deployed automatically.

# Production

1. <https://portal.azure.com>
2. On the dashboard click on your {MyBot} > Click on Channels > Click Get bot embed codes > Click the Web Chat icon > Click "Click here to open the Web Chat configuration page" > Copy the Url from the Embed code iframe src
   1. Ex. https://webchat.botframework.com/embed/{MyBot}?s=YOUR\_SECRET\_HERE
3. Click Show for the first Secret key > Replace YOUR\_SECRET\_HERE with the secret key
4. Paste that url into a browser and the web chat should open
5. Type: hi
   1. Response: Query: hi; Intent: Greeting; Entity: ;
6. Type: define dry
   1. Response: Query: define dry; Intent: Define; Entity: dry;

# Resources

1. <https://docs.microsoft.com/en-us/bot-framework/azure-bot-service-template-language-understanding>