Build Smart Bots with Bot Framework and Microsoft Cognitive Services

Hands-on lab

Natural Campaign

Greg Degruy
Adebisi Oje
CONTENTS

OVERVIEW ........................................................................................................................................... 3

Objectives ........................................................................................................................................... 4

System Requirements and Setup ........................................................................................................... 5

EXERCISE 1: CREATE A BOT .................................................................................................................. 6

Task 1 – Register Bot ............................................................................................................................ 6

Task 2 – Set up Starter Bot Project ...................................................................................................... 6

Task 3 – Test the Bot in the Bot Emulator ............................................................................................ 7

EXERCISE 2: ADD A LUIS MODEL ......................................................................................................... 9

Task 1 – Create an Application ............................................................................................................ 9

Task 2 – Set up Model .......................................................................................................................... 10

Task 3 – Train Model ............................................................................................................................ 11

Task 4 – Publish Model ....................................................................................................................... 13

EXERCISE 3: CODE THE BOT ............................................................................................................. 14

Task 1 – Connect LUIS Model to Bot Code ....................................................................................... 14

Task 2 – Add News Search capabilities ............................................................................................... 15

Task 3 – Add Sentiment Analysis capabilities .................................................................................... 17

Task 4 – Test bot e2e in Bot Emulator ............................................................................................... 17

EXERCISE 4: CONNECT BOT TO SKYPE .......................................................................................... 21

Task 1 – Create Azure Web App ......................................................................................................... 21

Task 2 – Deploy Bot to Azure ............................................................................................................. 21

Task 3 – Connect Bot to Skype .......................................................................................................... 22
Overview

Microsoft Bot Framework and Microsoft Cognitive Services provide a comprehensive offering to build and deploy high quality intelligent bots your users can enjoy in their favorite messaging experience.

**Microsoft Bot Framework:** Developers writing bots all face the same problems: bots require basic I/O; they must have language and dialog skills; they must be performant, responsive and scalable; and they must connect to users – ideally in any conversation experience and language the user chooses. Bot Framework provides just what you need to build, connect, manage and publish intelligent bots that interact naturally wherever your users are talking – from text/sms to Skype, Slack, Facebook Messenger, Kik, Office 365 mail and other popular services.

**Microsoft Cognitive Services:** A growing collection of powerful intelligence APIs, which allow you to quickly add state of the art artificial intelligence to your experiences with a few lines of code. They are a product of years of R&D by experts in the fields of computer vision, speech, natural language processing, knowledge retrieval and web search. All APIs are RESTful and can be easily integrated into any app or service in the programming language and platform of your choice. The APIs are constantly improving, learning and getting smarter, so your experience is always using the latest innovations coming from the Microsoft engineering teams. Simply drop the API call into your bot’s code and you are set.

Objectives

Your goal in this hands-on-lab is to build a *sentimental* news bot using Bot Framework and several intelligent APIs available in Microsoft Cognitive Services. When you complete the assignment, your bot will be able to:
• Understand and process user queries in natural language using Microsoft LUIS (Language Understanding Intelligent Service).
• Search for news articles around the world using the Bing News Search API
• Extract the sentiment of news articles (negative/positive) using the Text Analytics API.
• Return rich results to users, including Carousels and Skype Emoticons.

The sentimental news bot can find relevant news articles around the world, and give you a hint about each article’s sentiment by analyzing its description.

At the end of this lab, you will be comfortable developing your own AI-powered bots using Bot Framework and Microsoft Cognitive Services! We can’t wait to see what you’ll build next!

System requirements

You need the following to complete this lab:

• Microsoft Visual Studio 2015 or 2017

• Microsoft Azure Subscription
• Microsoft Account (e.g. outlook.com)
• Visual Studio Tools for Azure (optional, but recommended for the last exercise)
• Skype (latest version)

Setup

Perform the following steps to prepare your computer for this lab:

1. Install Microsoft Visual Studio 2015 or 2017 (update all VS extensions) to their latest versions).
2. Install Visual Studio 2015 Tools for Azure * The Azure Workload in Visual Studio 2017 includes all the tools and features you need to bring the power of Azure to your application
3. Download the starter VS project “StarterProject_SentimentalNewsBot.zip” from https://aka.ms/MvpSummit16Bot
4. Install the Bot Framework Emulator
5. Sign up to Microsoft Cognitive Services using your personal Microsoft account

Exercises

This Hands-on lab includes the following exercises:

1. Setting up a simple Bot with the Bot Builder SDK (5 minutes)
2. Adding a LUIS model to give your bot natural language processing capabilities (15 minutes)
3. Integrating Cognitive Services APIs to add smarts to your bot (30 minutes)
4. Deploying the bot to Azure and connecting it to Skype (10 minutes)

Estimated time to complete this lab: 60 minutes.

Exercise 1: Create a Bot

Estimated time to complete: 5 minutes

Task 1 – Register your Bot
Register a new bot in the bot framework website.

1. Go to https://dev.botframework.com/, sign in with your personal Microsoft account and then click “Register a bot” at the top of the screen.

2. Give your bot a Name, a Bot handle, and provide a short Description.

3. In the Configuration section:
   - Add a placeholder URL with an https protocol, like “https://www.contoso.com”, for the messaging endpoint (you’ll edit this when you deploy your bot to Azure).
   - Click “Create Microsoft App ID and password”. The next page will display your Microsoft App ID (write this down). Also stay on this page.
   - Click “Generate a password to continue”. Write down this Microsoft app password and keep it safe! This is the only time the password be displayed! Click “ok” and then click “Finish and go back to Bot Framework”

4. Fill out the rest of the required fields (Admin section is optional), also agree to the Privacy statement, Terms of use, and Code of conduct. Then click “Register”.

Task 2 – Set up the Starter Bot Project

1. Unzip the starter VS project “StarterProject_SentimentalNewsBot.zip” you downloaded from https://aka.ms/mvpsummit16bot to a folder of your choice and open the .sln solution file in Visual Studio.
   a. Hit OK and trust this prompt if you get it.

2. Set your Solution Configuration to Debug and your Solution Platform to Any CPU. Select Microsoft Edge (or your favorite browser) from the Debug Target dropdown menu.
3. Click the green run button to Build and run your app. You will see a blank app browser tab displaying the applications default.htm. Keep note of the port your Bot is running on (in our case port 3979) as well as the API URL to be used for testing your Bot.

4. Then, return to Visual Studio and click the red stop button to exit debugging.

**Task 3 – Test the Bot in the Bot Emulator**

The Bot Framework provides an emulator that lets you test your bot in your local machine.

1. Open the Bot Framework Emulator. You installed this in our Setup step.

2. Configure the Emulator to interact with you by opening Visual Studio back into view.
   b. Add your MicrosoftAppPassword to the value field in the Web.Config file value field. These values are placed in fields I've blocked with blue boxes.
c. Bring the emulator back into view. Add your Microsoft App Id and Microsoft App Password in the Emulator fields seen in the screenshot below

i. Ensure the URL in the Emulator matches the URL displayed in your web browser when you run your from Visual Studio and add “/api/messages” if not there already to the end of the URL. For instance, http://localhost:3979/api/messages as seen in the screenshot below

3. Run the project and test your Bot by typing a message in the Emulator's text box. The bot is very simple at the moment. It will send a brief description (twice!) when you add it to a conversation (tip: you can click on New Conversation hidden behind the 3 dots on the Emulator toolbar to emulate the start of a new conversation). Then it will simply repeat any message you send to it.
4. That's all for this exercise. In the next exercise will figure out how to make our bot smarter.
Exercise 2: Add a LUIS model

Estimated time to complete: 15 minutes

LUIS ([luis.ai](https://www.luis.ai)) is Microsoft’s Natural Language Processing toolkit. It provides a fast and effective way of adding language understanding to applications – an essential skill for bots. In this exercise you will build a LUIS model and integrate it into your bot’s code.

Creating a LUIS model can be summarized in three steps; create a LUIS application, set up your LUIS model and train it, and finally publish the model to the cloud.

**Task 1 – Create an Application**

1. Go to [www.luis.ai](https://www.luis.ai) and log in with your Microsoft account.

2. Click on the “My apps” tab and select “New App”.

3. Fill out the required information in the form from the pop-up. Using “SentimentalNewsBot” for the name and choose “English” as the App Culture and click “Create”.

![Microsoft Cognitive Services](https://example.com/microsoft_cognitive_services.png)

![My Apps](https://example.com/my_apps.png)
Task 2 – Create LUIS model

For the sentimental news bot, you will build a natural language processing model that responds to two user intents (“News” and “Help”). Furthermore, for user queries with “find news” intent, the model will be able to identify what is the “news topic” the user is interested in, as well as whether he is looking for news with “positive” or “negative” sentiment. LUIS refers to information extracted from natural language queries as entities.

Add Intents

1. You should now be on your App Dashboard. An alert dialog will notify you that no Intents exist in your app, so select “Create an Intent” to get started.

2. Use “News” as the first Intent name and click Save. Then enter an example Utterance that would trigger this intent, like “find news about gaming”, and press the Enter key. Be sure to hit the Save button before moving forward. (see example screenshots below)
Add Intent

Intent name (REQUIRED)

News

Save Cancel

News

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more

Utterances Entities in use Suggested utterances

find news about gaming

News

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ... Learn more

Utterances Entities in use Suggested utterances

Type a new utterance & press Enter ...

find news about gaming
3. Repeat the same process to add the second intent. Click the Intents label from the left side bar and then click the Add Intent button. Use “Help” as the Intent name and click Save. Then enter an example Utterance that would trigger this intent, like “need help” and press the Enter key. Be sure to hit the Save button before moving forward.

4. Click the Intents label from the left side bar and you should now see your new Intents!

Add Entities

Next, you will add the entities, i.e. the information the model will extract from the natural language queries. This will become more clear in the next step when you start training your model.

1. From the left sidebar, click on the Entities label. The side bar is under the label for shortened version of your app name, in my example screenshot the shortened app name is “Sentimental...”.

2. Click Add custom entity on the Entities page. Use “NewsTopic” as the entity name and keep the Simple Entity type selected. Then click Save.
3. Repeat the same process and add two more Entities named “PositiveSentiment” and “NegativeSentiment”
After you complete the previous steps, Entities list on the Entities page should look like this:

![Entities List](image)

**Task 3 - Train Model**

To start training the model you should provide sample utterances and label them.

1. Select the Intents label from the left side bar, then let's select the **News** Intent to start training. You should see the “find news about gaming” Utterance text added from the beginning of this exercise as seen in the screenshots below.

![Intents](image)
3. The utterance “find news about gaming” will be presented for labelling. Click on the words in the utterance to highlight them and label the entities for NewsTopic, PositiveSentiment, NegativeSentiment, as shown in the examples below. Click on Save after you completely label each utterance.
   a. To train the NewsTopic Utterance
      i. Select the word “gaming”

4. i. Choose the NewsTopic entity. This specifies gaming as a news topic to look for in future utterances.
5. i. You'll now see gaming has been linked to the NewsTopic Entity, represented by the [SNewsTopic] that now appears in our example utterance.
   
   ii. Click Save to complete the association.

b. NegativeSentiment Utterance
   
i. Add “negative news about gaming” in the “Type a new utterance & press Enter” box and press enter. Here we'll add 2 Entities to our training model.

6. i. You'll see our new utterance has been successfully added, now hit Save.
ii. Select the word “gaming”

iii. Select the word “negative”.

v. Select the word “negative”.
vi. Choose the NegativeSentiment entity. This specifies negative as a negative sentiment to look for in future utterances.

News

Here you are in full control of this intent; you can manage its utterances, used entities and suggested utterances ...

Utterances (2)  Entities in use (1)  Suggested utterances

Type a new utterance & press Enter ...

Save  Discard  Delete
Reassign Intent

[ negative ] news about [ $NewsTopic ]  0.86

Find news

vii. You’ll now see “negative” has been linked to the NegativeSentiment Entity, represented by the [ $NegativeSentiment ] that now appears in our example utterance.

viii. Click Save to complete the association.

b. PositiveSentiment Utterance

i. Add “good news about gaming” as a new utterance to train.

ii. Follow the same steps from part b. where we defined our NegativeSentiment Utterance, except be sure to use PositiveSentiment when select good.

iii. Be sure to click Save to complete the association.

7. Continue adding as many utterances as you can (note: you need to add at least 5 examples of each intent before you can train your model for the first time). The more you add, the better the precision/recall of
your model will be. Note that you can re-train your model at any time so that it gets better over time – you can even review messages submitted to the bot by users and re-label them if they were mis-classified! Luis refers to this feature as **Active Learning**.

8. Select Intents again from the left side menu. Follow the same process to submit utterances for the “Help” Intent as well, adding an utterance like “I need help”. You don’t need to identify any entities for queries with Help intent, so only make sure that the Help intent is selected from the drop-down menu before you submit the utterance for training.

9. Finally, click on the Train & Test label from the left side menu.
10. Hit the Train Application button on the Test your application page. You will get a confirmation message and note on the last date and time the app was trained when the training is completed (usually takes a few seconds for simple models).
Test your application

Use this tool to test the current and published versions of your application, to check if you are progressing on the right track. Learn more.

---

**Train Application**  Please train your application before testing.

**Interactive Testing**  Batch Testing

---

---

---
**Task 4 - Publish Model**

Finally, you will publish your model to the cloud. In other words, your model will be accessible by your bot application via a REST endpoint. Click “Publish App” located on the left side menu. You well need to select an Endpoint Key. Your given "BootstrapKey" should suffice for this. Selecting My Keys at the top menu, you’ll see this BootstrapKey is really your Programmatic API Key.

Publish App

Publish your app as a web service or as a chat bot. You can publish a new app or an updated version of a published app ... Learn more

Essentials

Latest publish: You haven’t published your application yet

Endpoint Key [REQUIRED]

Choose a key to assign to application ...

Add a new key to your account

Once published you’ll see your Luis id and Luis Subscription key (color coded below) embedded in the URL endpoint.

**Endpoint url:**

https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/f32590c3-5c01-4776-b22e-abdd01537ec8?subscription-key=5d7817feda724399aaf69441f3fb18eb&timezoneOffset=0.0&verbose=true&q=

You will use them later in your bot code, so take note of them. From the same pop-up, you can submit some test queries to test the REST endpoint.
Exercise 3: Code the News Bot

Estimated time to complete: 30 minutes

Switch to your bot project in Visual Studio to start coding your bot.

Task 1 – Connect LUIS model to Bot

1. Go to the Controllers folder and MessagesController.cs file and replace the code in the Post task with the following snippet. The code simply routes all incoming user messages to the RootDialog where you will do most of the coding.

```csharp
public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
{
    if (activity.Type == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new RootDialog());
    }
    else
    {
        await this.HandleSystemMessage(activity);
    }
    var response = Request.CreateResponse(HttpStatusCode.OK);
}
2. Go to the Dialogs folder and **RootDialog.cs** file. Add the **Luis id** and **Luis subscription key** of your model that you got in Exercise 2 to the LuisModel parameter on top of the RootDialog class as shown below.

   ```csharp
   [Serializable]
   [LuisModel("YourLuisId", "YourLuisSubscriptionKey")]
   public class RootDialog : LuisDialog<object>
   ```

   **Tip:** In the code file, you can also find the Luis Id and subscription key for the Luis model we used for our bot (commented out by default). You can use these instead if you like to test your bot with our pre-built model.

3. The RootDialog is now hooked up to your LUIS model. Whenever the user sends a message to the bot, your bot service will route it to the RootDialog. The dialog will in turn call your LUIS model to identify the message’s intent and entities and finally call the corresponding function/task to handle the bot’s response. For example, if the user message has a “Help” intent, the Help task will be called (note how the function that is called is controlled by the **LuisIntent parameter** as shown below.

   ```csharp
   [LuisIntent("Help")]
   public async Task Help(IDialogContext context, LuisResult result)
   {
       string response = "$I can help you find the freshest news articles...
       // more string message data
       await context.PostAsync(response);
       context.Wait(this.MessageReceived);
   }
   ```

**Task 2 – Code the News Search Task**

You’ll now add the code that will run the Bing news search task whenever the user sends a request with a News intent. The news results will be presented to the user in a News Carousel.

1. In the **RootDialog.cs** file, create a new task named `getBingNews` responsible for fetching the news from Bing for a given query. Replace `this` with the **Bing Search API key**, which can be found in your Microsoft Cognitive Services **subscription page**.

   ![Subscribe to new free trials](image)

   Tab into the power of machine learning with easy to use REST APIs
private async Task<BingNews> getBingNews(string query)
{
    BingNews bingNews;
    String bingUri = "https://api.cognitive.microsoft.com/bing/v5.0/news/search/?count=50&q=" + query;
    String rawResponse;

    HttpClient httpClient = new HttpClient()
    {
        DefaultRequestHeaders = {
            {"Ocp-Apim-Subscription-Key", "YourApiKey"},
            {"Accept", "application/json"}
        }
    };
    try
    {
        rawResponse = await httpClient.GetStringAsync(bingUri);
        bingNews = JsonConvert.DeserializeObject<BingNews>(rawResponse);
    }
    catch (Exception e)
    {
        return null;
    }
    return bingNews;
}

Note: The starter project already includes the model for the Bing News API response. If you are curious, you can find the model in the BingNews.cs file in your project folder.

2. Now, head to the Search task, and replace the current code there with the below. The code first ensures the task is called for any query that has a News intent. It then parses the NewsTopic entity (if available) and performs a Bing News search to retrieve the news results.

[LuisIntent("News")]
public async Task Search(IDialogContext context, IAwaitable<IMessageActivity> activity, LuisResult result)
{
    var message = await activity;
    var reply = context.MakeMessage();
    EntityRecommendation newsEntity, sentimentEntity;

    if (result.TryFindEntity("NewsTopic", out newsEntity))
    {
        var findPositive = result.TryFindEntity("PositiveSentiment", out sentimentEntity);
        var findNegative = result.TryFindEntity("NegativeSentiment", out sentimentEntity);

        await context.PostAsync((findPositive ? "positive " : (findNegative ? "negative " : "") + "news about " + newsEntity.Entity + " coming right up \U0001F680!");
BingNews bingNews = await getBingNews(newsEntity.Entity);

if (bingNews == null || bingNews.totalEstimatedMatches == 0) {
    reply.Text = "Sorry, couldn't find any news about '" + newsEntity.Entity + "' \U0001F61E."
;
}
else {
    reply.AttachmentLayout = AttachmentLayoutTypes.Carousel;
    reply.Attachments = new List<Attachment>();
    ...
}

3. Continue from where you left off, and add the following code to complete the search task. The code renders the news results in a Carousel. It also returns a static message to the user whenever LUIS has not managed to successfully parse the NewsTopic entity. Note also that most messaging channels, including Skype, limit the number of cards bots can show in a carousel to 10, which is why we used this limit here as well.

for (int i = 0; i < 10 && i < (bingNews?.totalEstimatedMatches ?? 0);
i++)
{
    var article = bingNews.value[i];
    HeroCard attachment = new HeroCard()
    {
        Title = article.name.Length > 60 ?
            article.name.Substring(0,57) + "..." : article.name,
        Text = article.provider[0].name + "", " +
            article.datePublished.ToString("d") + " - " +
            article.description,
        Images = new List<CardImage>() { new CardImage(article.image?.thumbnail?.contentUrl +
            "&w=400&h=400") },
        Buttons = new List<CardAction>() { new CardAction(
            ActionTypes.OpenUrl,
            title: "View on Web",
            value: article.url)});
    reply.Attachments.Add(attachment.ToAttachment());
}
else
{
    reply.Text = "$I understand you want to search for news, but I couldn't understand the topic you're looking for \U0001F633. "$;
    reply.Text += "$Rephrase your question or re-train your LUIS model!";
}
4. Build the project and run it. Head to the emulator and test your bot by sending a query, with a news intent, like “find news about gaming”. If all goes well, you will see results similar to the screenshot below.

![Bot Framework Channel Emulator](image)

**Task 3 – Add Sentiment Analysis in News Search**

For the last coding task, you will add sentiment analysis capabilities to the news bot.

1. Create a new task in the RootDialog class called getSentimentScore. Add the following code. The task receives a string (in our case, the article’s headline description), and sends it to the Text Analytics API to process the sentiment of the string. The **Text Analytics API** returns a sentiment score that spans from 0 (very negative) to 1 (very positive). Remember to replace this with your personal Text Analytics key, which can be found in your Microsoft Cognitive Services subscription page. Search for Text Analytics - Preview and click the Subscribe button.
private async Task<double> getSentimentScore(string documentText)
{
    string queryUri =
    "https://westus.api.cognitive.microsoft.com/text/analytics/v2.0/sentiment";

    HttpClient client = new HttpClient()
    {
        DefaultRequestHeaders = {{"Ocp-Apim-Subscription-Key", "YourAPIKey"},
                                  {"Accept", "application/json"}}
    };

    var textInput = new BatchInput
    {
        documents = new List<DocumentInput> {
            new DocumentInput
            {
                id = 1,
                text = documentText,
            }
        }
    };

    var jsonInput = JsonConvert.SerializeObject(textInput);

    HttpResponseMessage postMessage;
    BatchResult response;
    try {
        postMessage = await client.PostAsync(queryUri, new StringContent(jsonInput, Encoding.UTF8, "application/json"));
        response = JsonConvert.DeserializeObject<BatchResult>(await postMessage.Content.ReadAsStringAsync());
    }
    catch (Exception e)
    {
        return 0.0;
    }
    return response?.documents[0]?.score ?? 0.0;
}

2. Next, create another function in the RootDialog class named getSentimentLabel. The function simply returns a sentiment label based on the sentiment score of the article.

private string getSentimentLabel(double sentimentScore)
{
    string message;

    if (sentimentScore <= 0.1)
        message = "$Extremely Negative :@";
else if (sentimentScore <= 0.2)
    message = "Very Negative (facepalm)";
else if (sentimentScore < 0.4)
    message = "Negative :(";
else if (sentimentScore <= 0.6)
    message = "Neutral :^)";
else if (sentimentScore <= 0.8)
    message = "Positive :P";
else if (sentimentScore < 0.9)
    message = "Very Positive :D";
else
    message = "Extremely Positive (heart)";

message += " (" + (int)(sentimentScore * 100) + ")%";
return message;
}

3. Finally, make the following highlighted changes to the Search Task in RootDialog.cs. With these new changes, the bot will be able to understand if the user wants to search for news articles with positive or negative sentiment, and will also render the sentiment label of each article in the News Card results.

```csharp
for (int i = 0; i < 10 && i < (bingNews?.totalEstimatedMatches ?? 0); i++)
{
    var article = bingNews.value[i];

    var sentimentScore = await getSentimentScore(article.description);

    if (findPositive && sentimentScore < 0.6)
    {
        continue;
    }
    else if (findNegative && sentimentScore > 0.4)
    {
        continue;
    }

    HeroCard attachment = new HeroCard()
    {
        Title = article.name.Length > 60 ? article.name.Substring(0, 57) + "..." : article.name,
        Subtitle = getSentimentLabel(sentimentScore),
        Text = article.provider[0].name + " , " + article.datePublished.ToString("d") + " - " + article.description,
        Images = new List<CardImage>() { new CardImage(article.image?.thumbnail?.contentUrl + ":w=400&h=400") },
        Buttons = new List<CardAction>() { new CardAction( ActionTypes.OpenUrl,
            title: "View on Web",
            value: article.url)
        };
    }

    reply.Attachments.Add(attachment.ToAttachment());
```
Task 4 – Test Bot E2E in Bot Emulator

1. Congratulations! You competed all coding steps. Build your project and run it to test the bot e2e in the Emulator. Here are a few examples you can try (reminder: if your bot is not able to parse correctly the intent/entities from the user messages, you can always go back to LUIS.ai and re-train your model in real-time using fresh utterances!)

- “Show me good news about gaming”

- “Positive press about gaming”
• “Find negative news about gaming”
Exercise 4: Connect Bot to Skype

Estimated time to complete: 10 minutes

Congratulations for making it this far! In this last exercise, you will deploy your bot to the cloud and connect it to Skype. For this step, we recommend installing the Visual Studio Tools for Azure to make the cloud deployment process easier.

Task 1 – Create a new Azure Web App

2. Log in to Azure Portal
3. Click on the Plus button

4. Select Web + Mobile from the list and then select Web App from the Featured Apps list. Follow the on-screen instructions.
   - Add an App Name
   - Choose your subscription
   - Keep Create a new Resource Group selected and provide a name for it
   - Click on App Service plan/location and select Create New
     i. Add a name for the App Service Plan
     ii. Add a Location
     iii. Choose pricing tier and scroll down until you see the Free tier
     iv. Click the Select button in the Choose your pricing tier
   - Hit the OK button in the New App Service Plan tab
• The Web App tab should now be the only one open. Keep Application Insights off, select Pin to dashboard at the button, and then click Create.

5. You should now be navigate to your Azure Dashboard. You’ll see your Web App tile and it is now in the deployment process. You Should be navigate to your Web App management page automatically, if not click on your new Web App tile on your dashboard once deployment is completed.

6. On your Web App copy your app’s URL. You will need it for the next steps. The URL is under the Essentials section on your Web App management page.
Task 2 – Deploy your Bot to Azure Web App

1. Return to Visual Studio, right-click your Solution “EmotionalNewsBot” in the Solution Explorer and select Publish

![Image of Visual Studio with Publish option highlighted]

2. The dialog will ask you to log in to your Azure account. Once you log in, select the subscription under which you created your Web App and then select your Web App from the list and click OK. The publishing experience in Visual Studio 2017 is much different than in 2015, we’ll note these differences below:

Visual Studio 2015

![Image of Microsoft Azure Web Apps]

Visual Studio 2017 * choose Microsoft Azure App Service, choose Select Existing, then click Publish.
You’ll then see the below screen, make sure you’ve logged into your Azure Account as I've done in the top right. Then select your subscription, choose Resource Group view, and then choose your Web App under the resource group you deployed it too from our portal walkthrough earlier and hit OK.
3. If everything goes well, the connection settings will already be pre-populated in the Connection dialog. Validate the connection and then hit publish. Once the publishing process completes, navigate to https://{yourwebappname}.azurewebsites.net to confirm your service is successfully published.

Error! Hyperlink reference not valid. The publishing experience in Visual Studio 2017 is much different then in 2015, we’ll note these differences below

Visual Studio 2015

![Visual Studio 2015 Publish Screen](image)

Visual Studio 2017 * you’ll then see the below Publish screen and your web (notice the output dialog at the bottom) will deploy for you automatically, no need to hit Publish, unless you do not see the output Like ours below showing Build and Publish 1:succeeded each.

![Visual Studio 2017 Publish Screen](image)
**Task 3 – Connect Bot to Skype**

1. Go to your bot’s page in [https://dev.botframework.com/bots](https://dev.botframework.com/bots).

2. Click on Edit on the left side of the screen, next to the Details label.

   ![Image of bot details](160x384 to 452x638)

   **Details**
   
   Bot handle
   gregdegruyBot

   Bot Framework Version
   3.0

   Messaging endpoint
   [https://www.contoso.com](https://www.contoso.com)

3. Edit the Messaging endpoint. Add the URL of the Web App you just created and append the `/api/messages` endpoint at the end. It should look similar to this.

   ![Image of configuration](124x234 to 452x322)

   **Configuration**
   
   Messaging endpoint:
   
   [https://usdxatscale.azurewebsites.net/api/messages](https://usdxatscale.azurewebsites.net/api/messages)

4. Click Save changes at the bottom. You should be auto navigated back to the main bot page, if not go back to the main bot page, and click on Test to test the connection to your bot. If everything goes well, you will receive a confirmation message.
Test connection to your bot

Endpoint authorization succeeded

5. From the Channels list, find Skype and click on Edit

<table>
<thead>
<tr>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test link</td>
</tr>
<tr>
<td>Skype</td>
</tr>
</tbody>
</table>

6. In the dialog that appears, select **Enable [The name of your bot will appear here] on Skype** and hit the **I’m done configuring** button.
Settings

Enable GregDegruyBot on Skype
Disabling your bot stops messages being sent and received, and removes this channel from directories.

Your bot’s capabilities
These settings determine how a user can talk with your bot in Skype. Only include the capabilities your bot can handle.

Message types. Learn more

Text and pictures

Cards. Learn more

Enable autoplay of media content
Users will be notified in your bot profile that your bot may autoplay media content from third parties.

Groups. Learn more

Group messaging

Calls. Learn more

1:1 audio calls

Bing Entity and Intent Detection (Preview). Learn more

Detect intents and entities in text messages
Entities and intents will be identified and sent to your bot with each message. By using this service you agree to the Bing Entity and Intent Detection API Terms of Use.

Delete channel

I’m done configuring Skype ➤
8. Finally, click on the **Add to Skype** button to add your bot to your Skype contacts and start interacting with it! You may be prompted to download Skype preview, confirm this and install it. Tip: You can share the Skype Join link with anyone that wants to play with your bot.
Summary and More Resources

Congratulations on completing this Hands-on Lab! We hope you enjoyed the assignment. You should now be comfortable developing your own bots using Microsoft Bot Framework, and enriching them with AI intelligence via Microsoft Cognitive Services.

Here are some next steps to continue your journey of mastering bot development:

1. Visit the Bot Framework Docs site (https://docs.botframework.com/) to get in-depth information about all the capabilities supported by Bot Framework. Another good idea is to track the BotBuilder SDK github repo and participate in the technical discussions.

2. Continue refining your bot’s NLP model in https://www.luis.ai. Experiment by adding more intents/entities and integrating them into your bot’s code. Become an expert in LUIS by reading the LUIS docs and watching the 10’ getting started video.

3. Learn more about all the intelligence APIs offered by Microsoft at https://microsoft.com/cognitive.

4. Find bot code samples and fully functional open-source bots at the BotBuilder-Samples github repo. Do you have ideas for future bot samples or functional bots we can add to the collection? We’d love to hear from you. Simply file your request in our uservoice portal.

Feedback

Your feedback is extremely valuable for us, so don’t hesitate to reach out if you have ideas, questions or issues to report.

For Bot Framework feedback:

- Report issues in the Github issues tracker. For technical question, post a question in StackOverflow.com with the tag botframework.

For Microsoft Cognitive Services feedback:

- Use the Cognitive Services UserVoice portal to submit ideas, general feedback or questions.
- For technical questions, post a question in StackOverflow.com with the tag microsoft-cognitive.