Microsoft

Build Smart Bots with Microsoft Bot Framework and Cognitive Services

Microsoft
Bot Framework
& Cognitive
Services

Hands-on lab

John Psaroudakis (ioanp@microsoft.com)

Chris McConnell (chrimc@microsoft.com)

Lars Liden (laliden@microsoft.com)

Lauren Mills (lamil@microsoft.com)

Cindy Noteboom (cynotebo@microsoft.com)

Will Portnoy (wportnoy@microsoft.com)

CONTENTS

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.

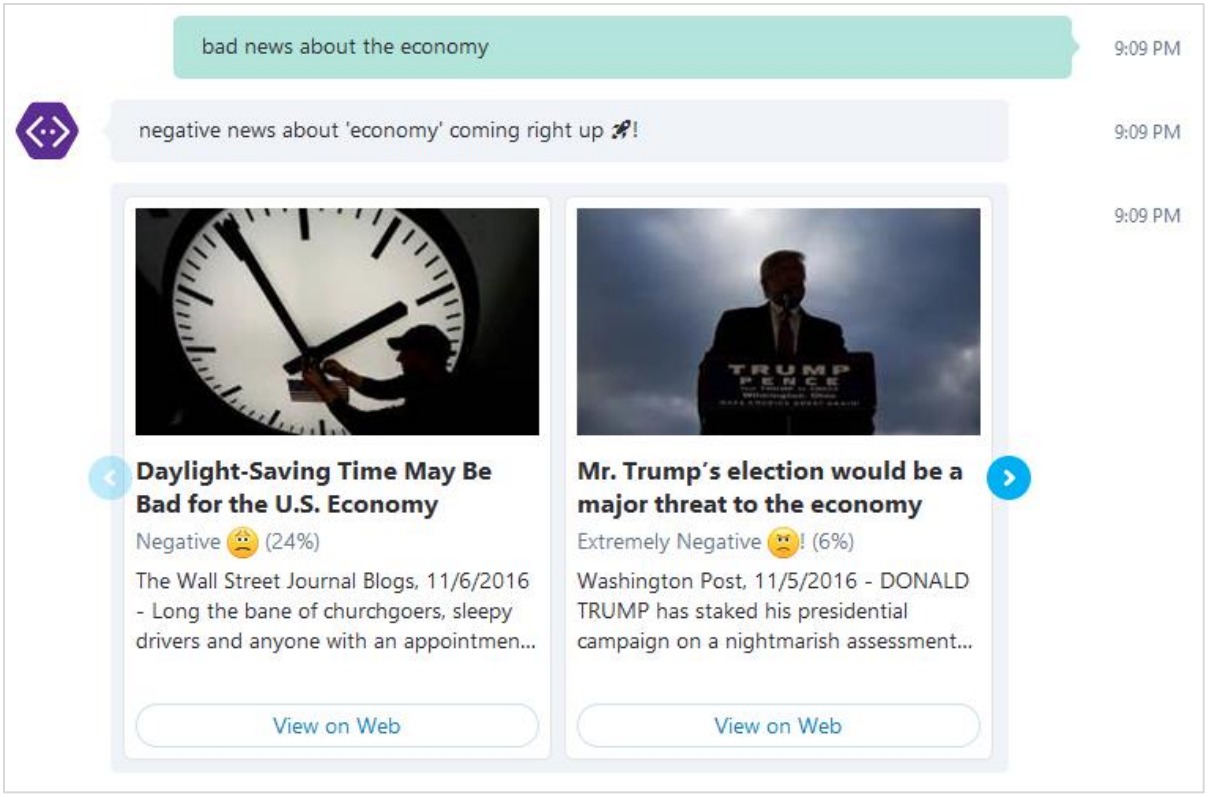
Microsoft Cognitive Services: A growing collection of powerful intelligence APIs, which allow you to quickly add state of the art artificial intelligence to your experiences with a few lines of code. They are a product of years of R&D by experts in the fields of computer vision, speech, natural language 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 LUIS.
- 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.



bad news about the economy 9:09 PM

negative news about 'economy' coming right up 🚀! 9:09 PM

9:09 PM

Daylight-Saving Time May Be Bad for the U.S. Economy
Negative 😟 (24%)
The Wall Street Journal Blogs, 11/6/2016 - Long the bane of churchgoers, sleepy drivers and anyone with an appointmen...
View on Web

Mr. Trump's election would be a major threat to the economy
Extremely Negative 😠! (6%)
Washington Post, 11/5/2016 - DONALD TRUMP has staked his presidential campaign on a nightmarish assessment...
View on Web

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
- 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 (update all VS extensions to their latest versions).
2. Install Visual Studio Tools for Azure
3. Download the starter VS project from https://aka.ms/MvpSummit16Bot
4. Install the Bot Framework Emulator
5. Sign up to Microsoft Cognitive Services using your personal MSA 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 MSA 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 small description.

3. In the Configuration section:

- use a placeholder url 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 App ID (**write this down**).

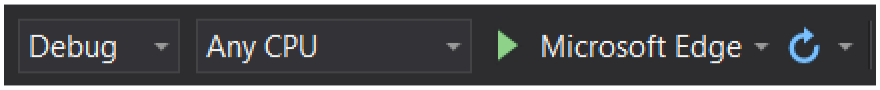
- Click “Generate a password to continue”. **Write down this password and keep it safe! This is the only time the password be displayed!** Click “ok” and then click “Finish and go back to Bot Framework”

4. Fill out the rest of the required fields. Then click “Register”.

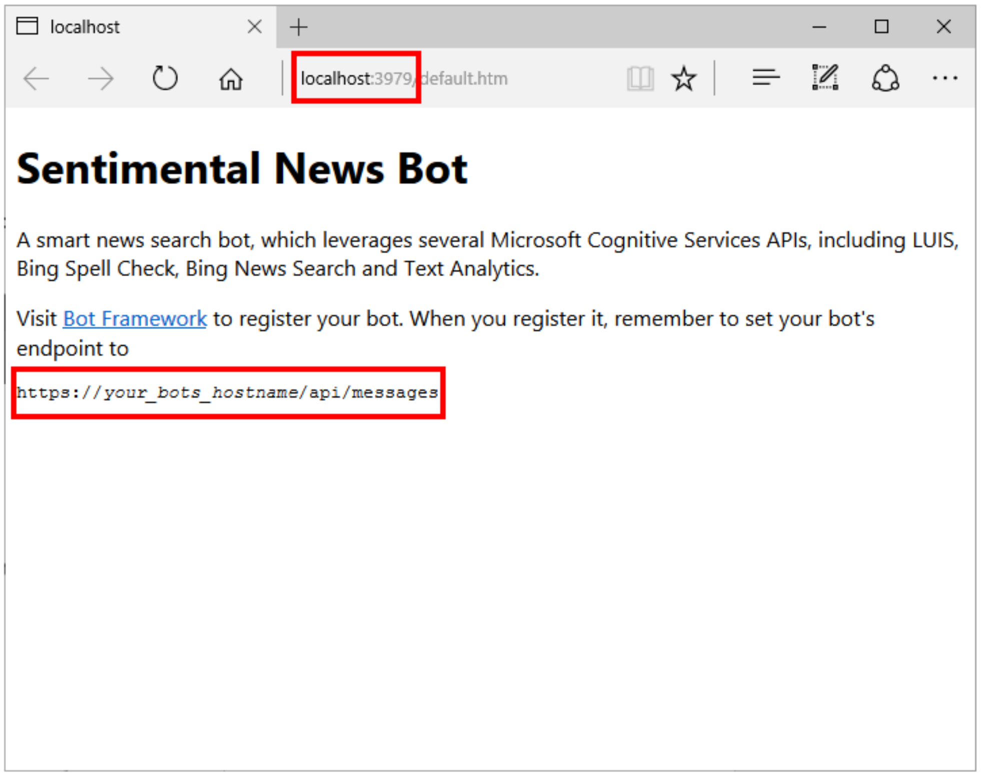
Task 2 – Set up the Starter Bot Project

1. Unzip the starter VS project you downloaded from https://aka.ms/mvpsummit16bot to a folder of your choice and open the solution file in Visual Studio.

2. Set your Solution Configuration to **Debug** and your Solution Platform to **Any CPU**. Select **Microsoft Edge** (or your favourite browser) from the Debug Target dropdown menu.



3. 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 as well as the API URL to be used for testing your Bot.** Then, return to Visual Studio and stop debugging.



Sentimental News Bot

A smart news search bot, which leverages several Microsoft Cognitive Services APIs, including LUIS, Bing Spell Check, Bing News Search and Text Analytics.

Visit Bot Framework to register your bot. When you register it, remember to set your bot's endpoint to

```
https://your_bots_hostname/api/messages
```

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.

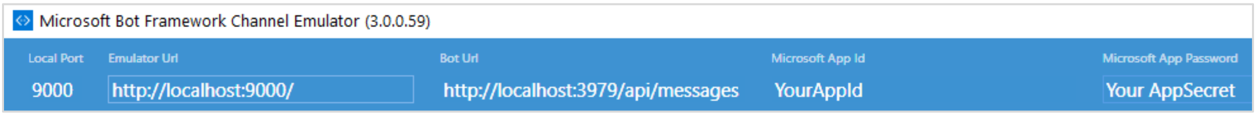
2. Configure the Emulator to interact with you Bot.

   a. Ensure the URL in the Emulator matches the URL displayed in your web browser and add "/api/messages" to the end of the URL. For instance, **http://localhost:3979*/api/messages***

   b. Add your **AppId** to the Web.Config file (located in your solution directory) and to the App Id in the Emulator.

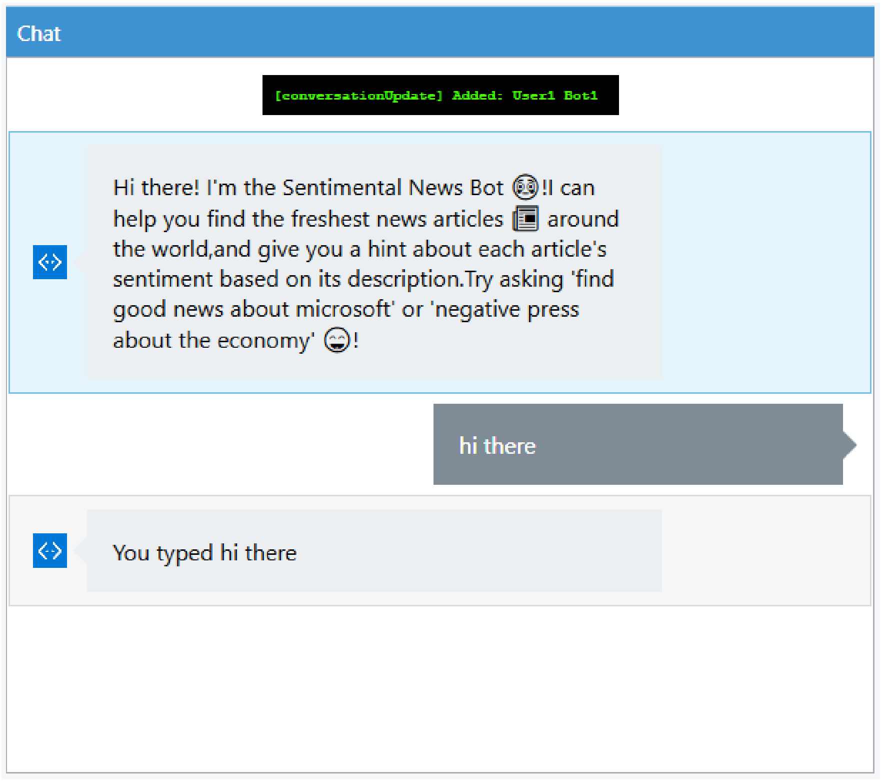
   c. Add your **AppSecret** to the Web.Config file and to the App Id in the Emulator.

```
<appSettings>
  <!-- update these with your BotId, Microsoft App Id and your Microsoft App Password-->
  <add key="BotId" value="YourBotId" />
  <add key="MicrosoftAppId" value="YourAppId" />
  <add key="MicrosoftAppPassword" value="YourAppSecret" />
</appSettings>
```


Microsoft Bot Framework Channel Emulator (3.0.0.59)

Local Port	Emulator Url	Bot Url	Microsoft App Id	Microsoft App Password
9000	http://localhost:9000/	http://localhost:3979/api/messages	YourAppId	Your AppSecret

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 when you add it to a conversation (tip: you can click on New from the Emulator toolbar to emulate the start of a new conversation). Then it will simply repeat any message you send to it.



```
[conversationUpdate] Added: User1 Bot1
```

Hi there! I'm the Sentimental News Bot 😨!I can help you find the freshest news articles 📰 around the world,and give you a hint about each article's sentiment based on its description.Try asking 'find good news about microsoft' or 'negative press about the economy' 😀!

hi there

You typed hi there

Exercise 2: Add a LUIS model

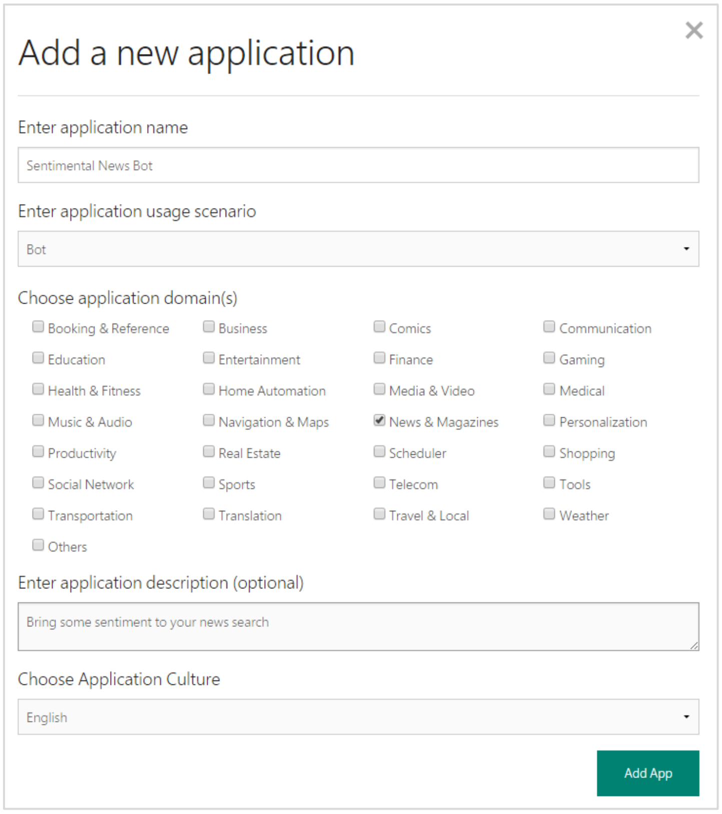
Estimated time to complete: 15' minutes

LUIS (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 and log in with your MSA account.
2. Click on "New App" and select "New Application" from the drop down menu.
3. Fill out the form in the pop-up. Choose 'English' as the App Culture and click on "Add App"



Add a new application

Enter application name: Sentimental News Bot

Enter application usage scenario: Bot

Choose application domain(s)

☐ Booking & Reference	☐ Business	☐ Comics	☐ Communication
☐ Education	☐ Entertainment	☐ Finance	☐ Gaming
☐ Health & Fitness	☐ Home Automation	☐ Media & Video	☐ Medical
☐ Music & Audio	☐ Navigation & Maps	☑ News & Magazines	☐ Personalization
☐ Productivity	☐ Real Estate	☐ Scheduler	☐ Shopping
☐ Social Network	☐ Sports	☐ Telecom	☐ Tools
☐ Transportation	☐ Translation	☐ Travel & Local	☐ Weather
☐ Others			

Enter application description (optional): Bring some sentiment to your news search

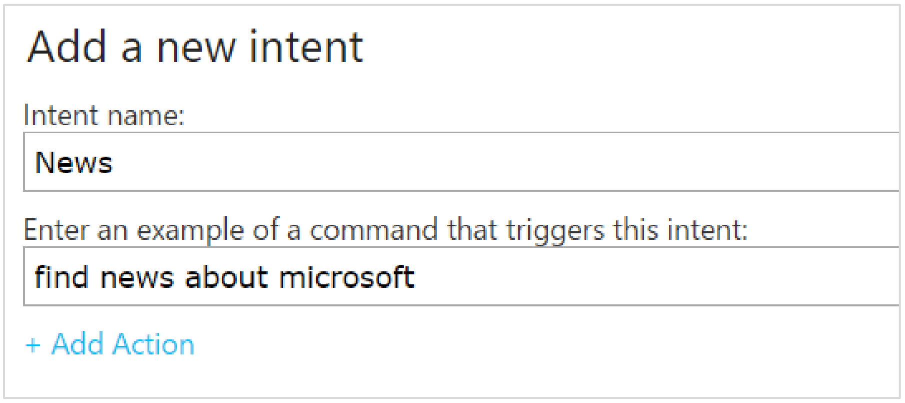
Choose Application Culture: English

Task 2 – Create LUIS model

For the sentimental news bot, you will build a natural language processing model that responds to two user intents ("find news" and "get 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. From the left sidebar, click on the + icon next to the Intents label. Use "News" as the Intent name and enter an example query that would trigger this intent (see example in screenshot).
2. Repeat the same process to add the second intent. Use "Help" as the Intent name.



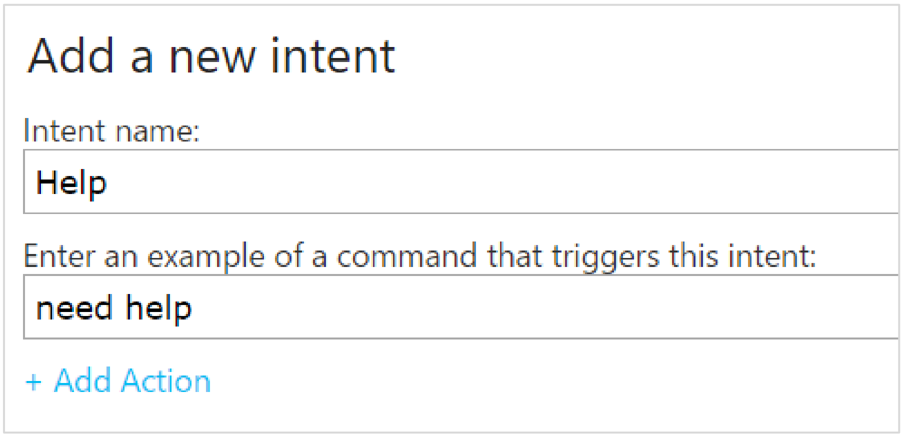
Add a new intent

Intent name:

News

Enter an example of a command that triggers this intent:

find news about microsoft



Add a new intent

Intent name:

Help

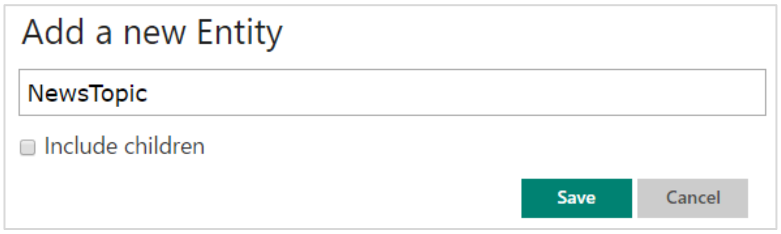
Enter an example of a command that triggers this intent:

need help

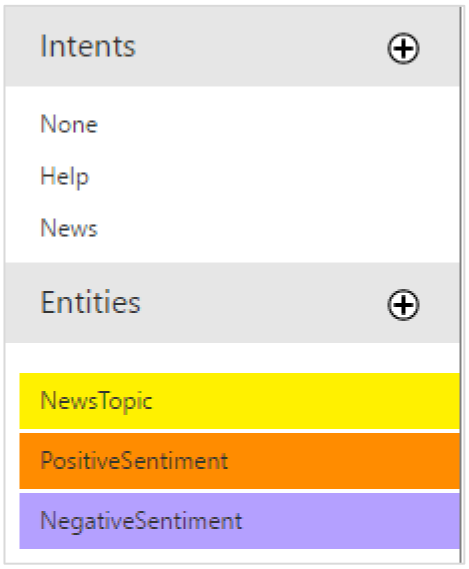
Add Entities

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 + icon next to the Entities label. Use "NewsTopic" as the entity name. Leave the 'Include children' checkbox unchecked.
2. Repeat the same process and add two more Entities named "PositiveSentiment" and "NegativeSentiment"



After you complete the previous steps, the left sidebar in your LUIS app should look like this:

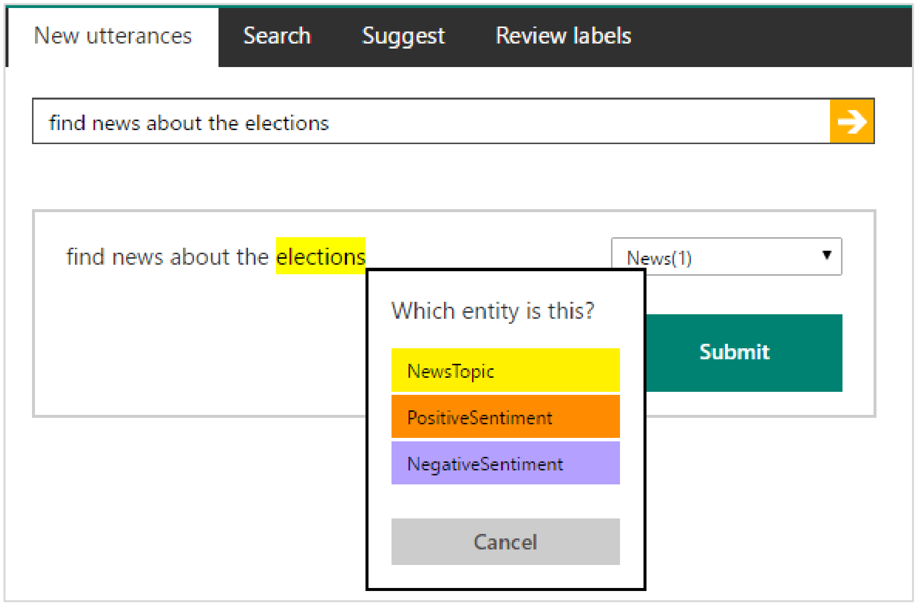


Task 3 - Train Model

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

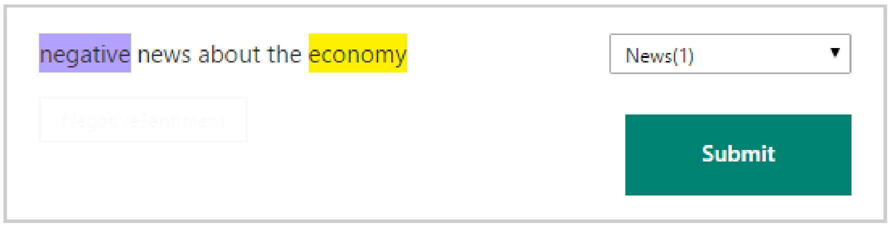
1. In the main window, make sure that the "New utterances" tab is selected
2. Start typing example utterances in the text box. Press enter to submit the utterance.

3. The utterance will be presented for labelling. Use the drop-down to select the intent for each utterance. In addition, click on the words in the utterance to highlight them and label the entities as shown in the examples below. Click on **Submit** after you label each utterance.

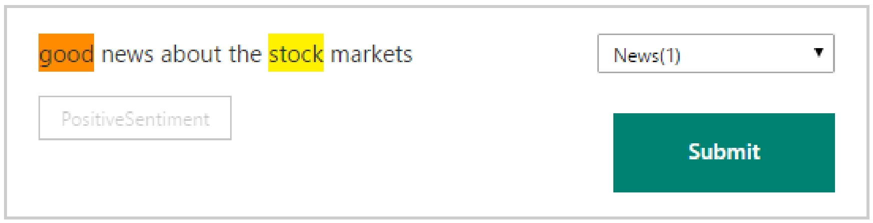


find news about the elections

find news about the elections



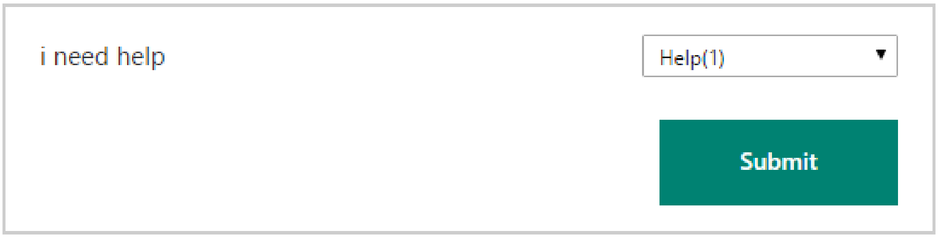
negative news about the economy



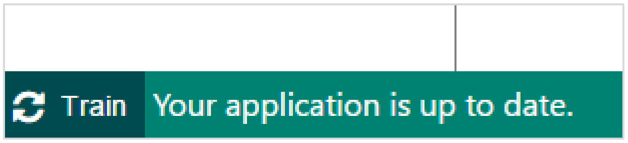
good news about the stock markets

4. 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**.

5. Follow the same process to submit utterances for the "Help" Intent as well. 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.

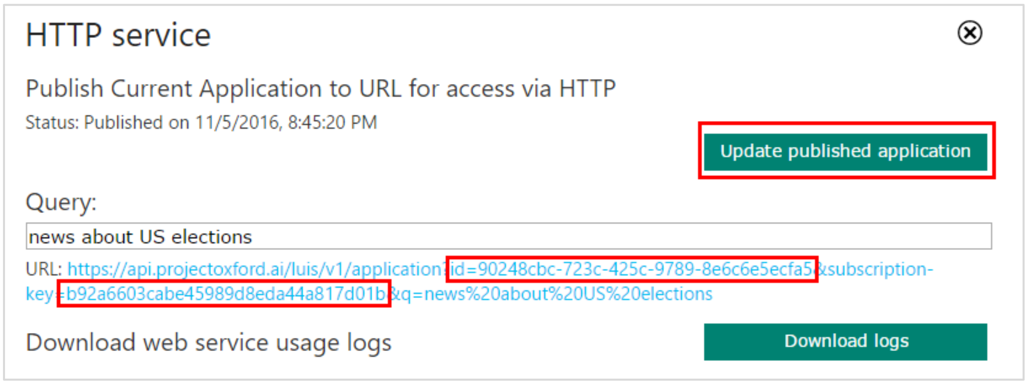


6. Finally, hit the Train button located at the bottom-left side of the screen. You will get a confirmation message when the training is completed (usually takes a few seconds for simple models)


Train
Your application is up to date.

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" located at the top left of the screen. On the pop-up, click first on **Update published application**. Write down the **Luis id** and **Luis Subscription key** in the URL endpoint. You will use them later in your bot code. From the same pop-up, you can submit some test queries to test the REST endpoint.


HTTP service

Publish Current Application to URL for access via HTTP

Status: Published on 11/5/2016, 8:45:20 PM

Update published application

Query:

news about US elections

URL: https://api.projectoxford.ai/luis/v1/application?id=90248cbc-723c-425c-9789-8e6c6e5ecfa5&subscription-key=b92a6603cabe45989d8eda44a817d01b&q=news%20about%20US%20elections

Download web service usage logs

Download logs

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 **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.

```
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);
    return response;
}
```

2. Go to the **RootDialog.cs** file. Add the **Luis id** and **subscription key** of your model that you got in Exercise 2 to the LuisModel parameter on top of the RootDialog class as shown below. **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.

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

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

```
[LuisIntent("Help")]
public async Task Help(IDialogContext context, LuisResult result)
{
    string response = $"I can help you find the freshest news articles…”
    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.

```
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 add the following code. 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.

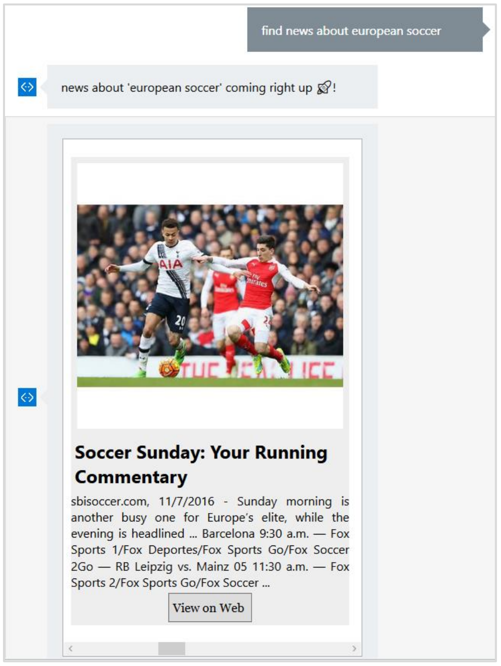
```
[LuisIntent("News")]
public async Task Search(IDialogContext context, IAwaitable<IMessageActivity>
       activity, LuisResult result)
 {
     var message = await activity;
     var reply = context.MakeMessage();
     EntityRecommendation newsEntity, sentimentEntity;

     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!";
        }
        await context.PostAsync(reply);
        context.Wait(this.MessageReceived);
    }
```

4. Build the project and run it. Head to the emulator and test your bot by sending a query with a news intent. If all goes well, you will see results similar to the screenshot below.



find news about european soccer

news about 'european soccer' coming right up 🚀!

Soccer Sunday: Your Running Commentary

sbisoccer.com, 11/7/2016 - Sunday morning is another busy one for Europe's elite, while the evening is headlined ... Barcelona 9:30 a.m. — Fox Sports 1/Fox Deportes/Fox Sports Go/Fox Soccer 2Go — RB Leipzig vs. Mainz 05 11:30 a.m. — Fox Sports 2/Fox Sports Go/Fox Soccer ...

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.

```
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;
}
```

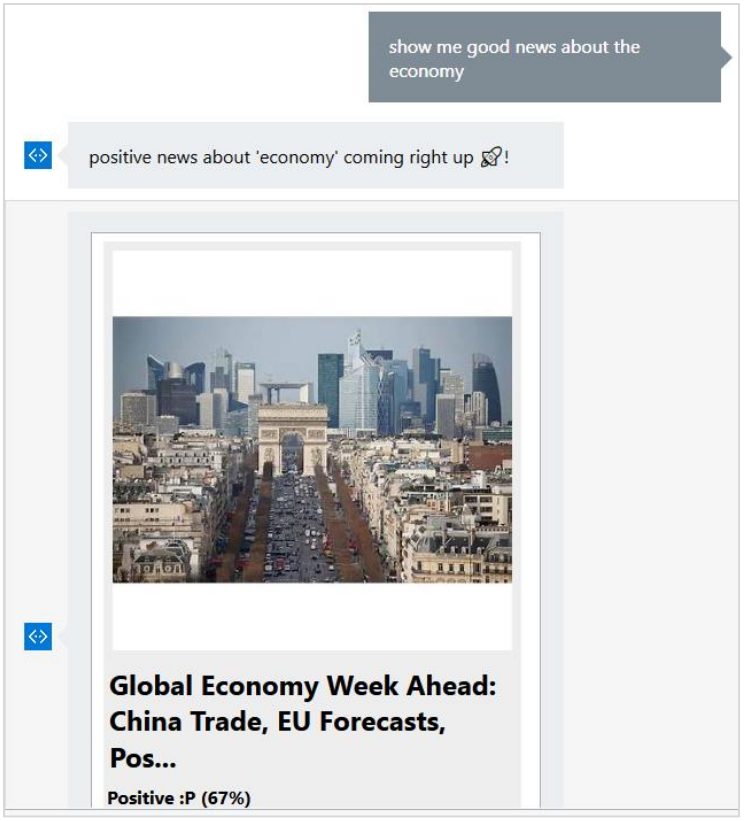
3. Finally, make the following highlighted changes to the Search Task. With these new changes, the bot will be able to understand if the user wants to search for news articles with positive or negative sentiment, and will also render the sentiment label of each article in the News Card results.

```
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 the economy"
   - "Positive press about the elections"
   - "Find negative news about MSFT stock"



show me good news about the economy

positive news about 'economy' coming right up 🚀!

Global Economy Week Ahead: China Trade, EU Forecasts, Pos...

Positive :P (67%)

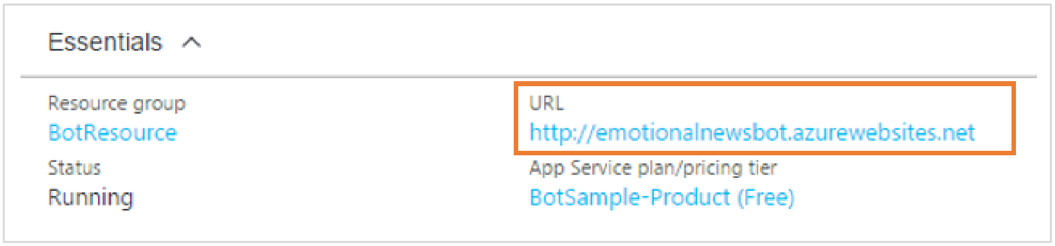
Exercise 4: Connect Bot to Skype

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

1. Log in to Azure Portal
2. Click on New -> Create Web App and follow the on-screen instructions
3. Once you create your app, copy your app’s URL. You will need it for the next steps.

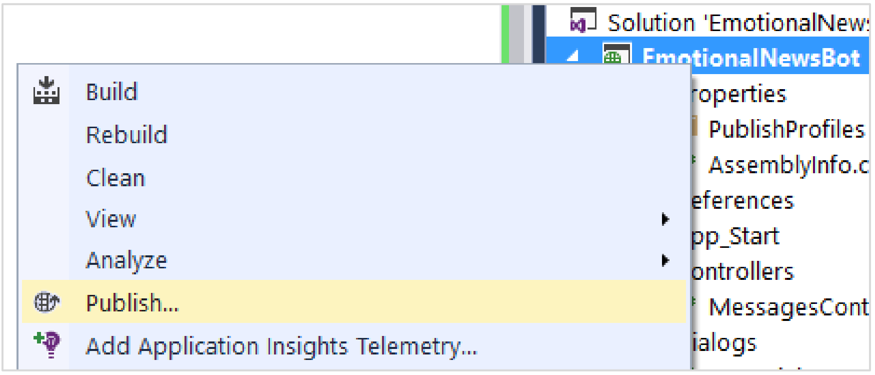


Essentials

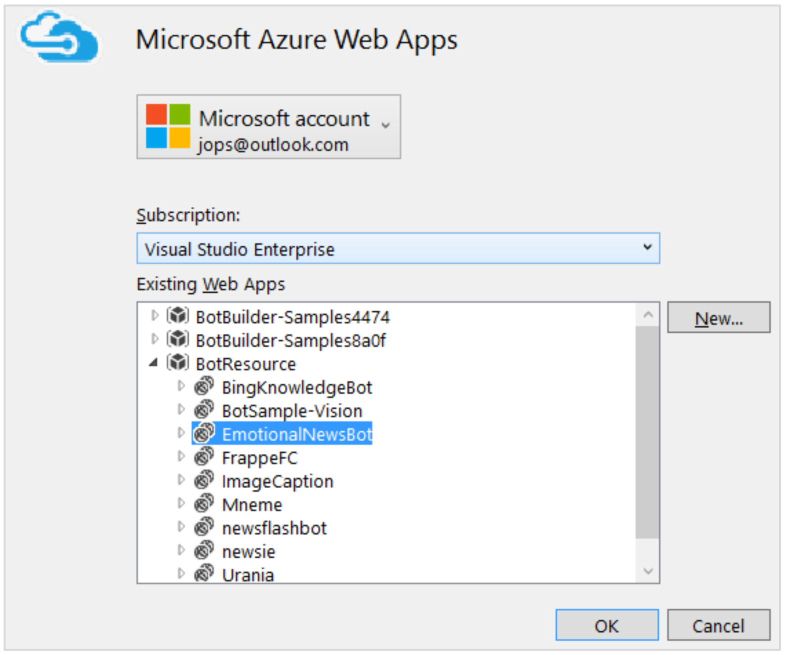
Resource group	URL
BotResource	http://emotionalnewsbot.azurewebsites.net
Status	App Service plan/pricing tier
Running	BotSample-Product (Free)

Task 2 – Deploy your Bot to Azure Web App

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



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.

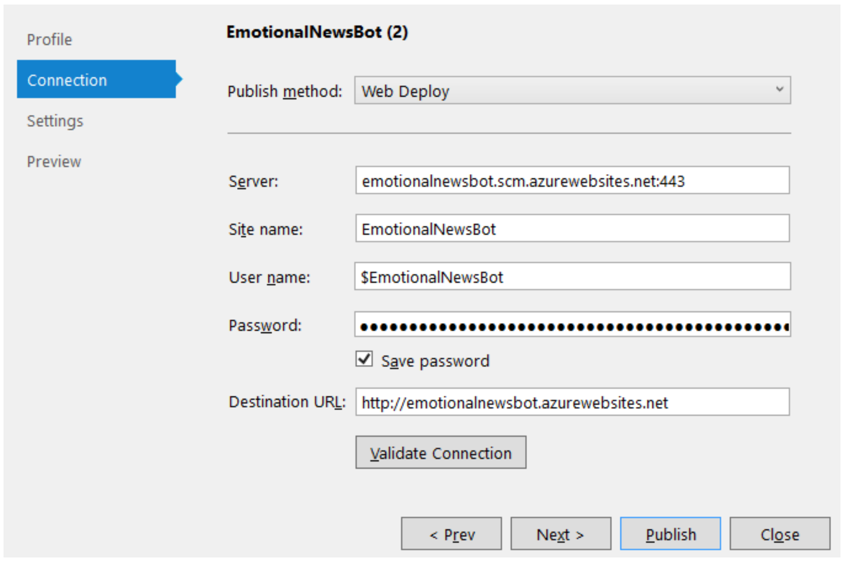

Microsoft Azure Web Apps

Microsoft account
jops@outlook.com

Subscription: Visual Studio Enterprise

Existing Web Apps
- BotBuilder-Samples4474
- BotBuilder-Samples8a0f
- BotResource
  - BingKnowledgeBot
  - BotSample-Vision
  - EmotionalNewsBot
  - FrappeFC
  - ImageCaption
  - Mneme
  - newsflashbot
  - newsie
  - Urania

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 succesfully published.



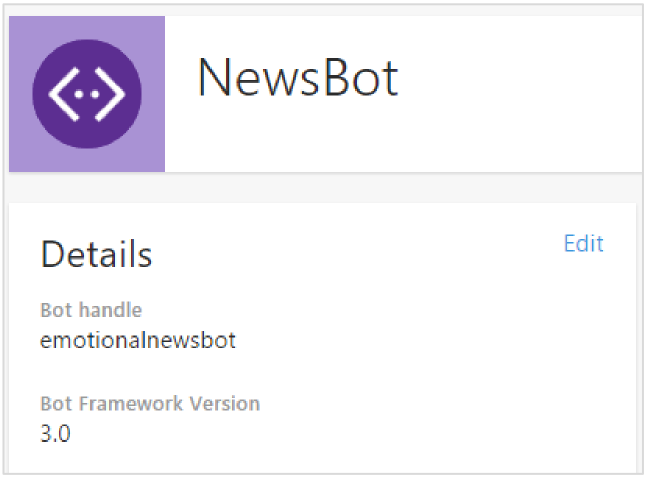
EmotionalNewsBot (2)

Publish method:	Web Deploy
Server:	emotionalnewsbot.scm.azurewebsites.net:443
Site name:	EmotionalNewsBot
User name:	$EmotionalNewsBot
Password:	●●
	✓ Save password
Destination URL:	http://emotionalnewsbot.azurewebsites.net

Task 3 – Connect Bot to Skype

1. Go to your bot’s page in https://dev.botframework.com.

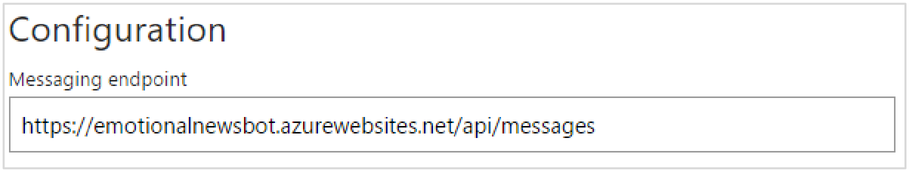
2. Click on Edit on the left side of the screen, next to the Details label.



Bot handle
emotionalnewsbot

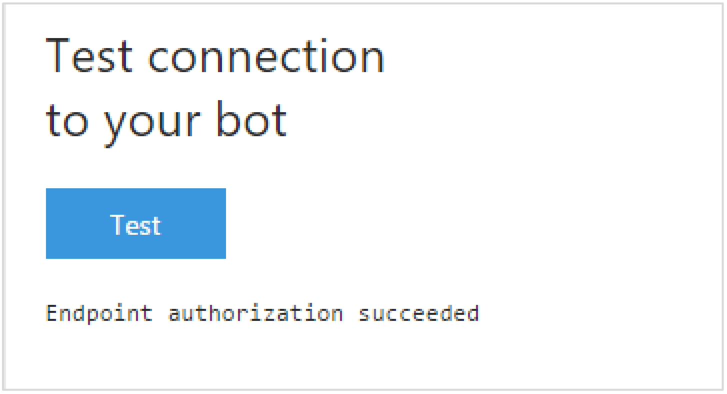
Bot Framework Version
3.0

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.



Messaging endpoint
https://emotionalnewsbot.azurewebsites.net/api/messages

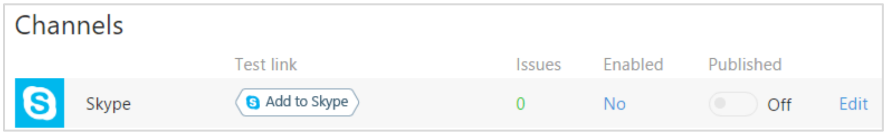
4. Go back to the main bot page, and click on Test to test the connection to your bot. If everything goes well, you will receive a confirmation message.



```
Endpoint authorization succeeded
```

5. 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.

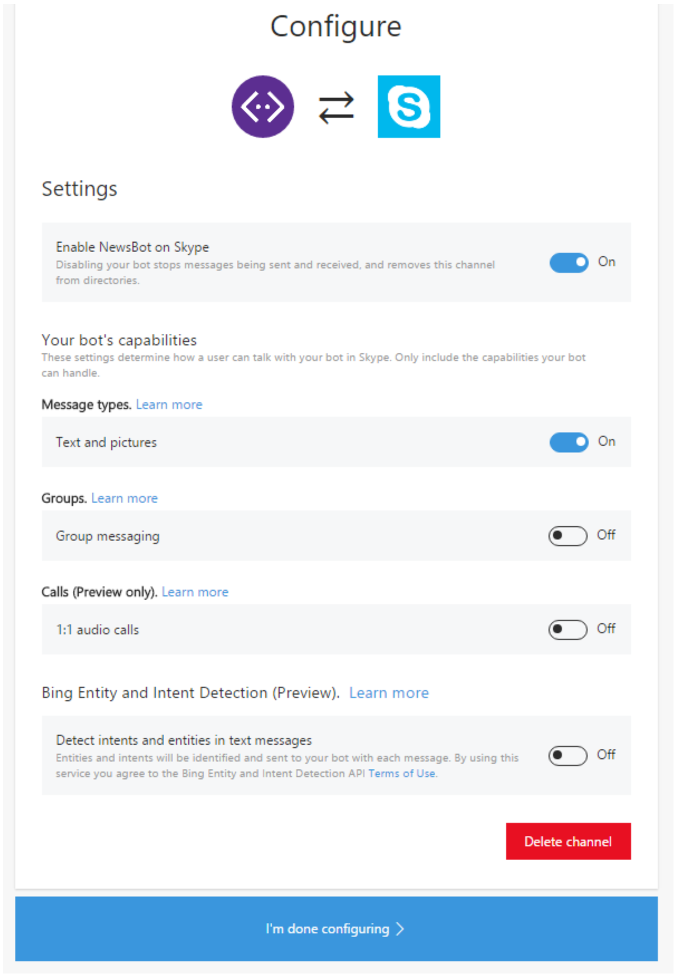
6. From the Channels list, find Skype and click on Edit



Channels

	Test link	Issues	Enabled	Published	
Skype	Add to Skype	0	No	Off	Edit

7. In the dialog that appears, select **Enable bot on Skype** and hit the **I'm done configuring** button.



Configure

Settings

Enable NewsBot on Skype — On
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.

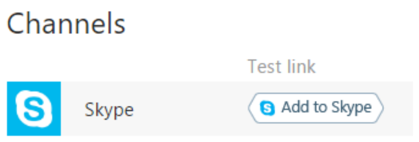
Message types. Learn more
Text and pictures — On

Groups. Learn more
Group messaging — Off

Calls (Preview only). Learn more
1:1 audio calls — Off

Bing Entity and Intent Detection (Preview). Learn more
Detect intents and entities in text messages — Off
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.

8. Finally, click on the **Add to Skype** button to add your bot to your Skype contacts and start interacting with it! Tip: You can share the Skype Join link with anyone that wants to play with your bot.



Channels

	Test link
Skype	Add to Skype

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.