

Tutorial: Use a Web App Bot enabled with Language Understanding in C#

06/22/2020 • 8 minutes to read •  +1

In this article

[Prerequisites](#)

[Create a web app bot resource](#)

[The bot has a Language Understanding model](#)

[Test the bot in Web Chat](#)

[Download the web app bot source code](#)

[Review code to send utterance to LUIS and get response](#)

[Start the bot code in Visual Studio](#)

[Use the bot emulator to test the bot](#)

[More information about bots](#)

[Next steps](#)

Use C# to build a chat bot integrated with language understanding (LUIS). The bot is built with the Azure [Web app bot](#) resource and [Bot Framework version V4](#).

In this tutorial, you learn how to:

- ✓ Create a web app bot. This process creates a new LUIS app for you.
- ✓ Download the bot project created by the Web bot service
- ✓ Start bot & emulator locally on your computer
- ✓ View utterance results in bot

Prerequisites

- [Bot emulator](#)
- [Visual Studio](#)

Create a web app bot resource

1. In the [Azure portal](#), select **Create new resource**.
2. In the search box, search for and select **Web App Bot**. Select **Create**.
3. In **Bot Service**, provide the required information:

Setting	Purpose	Suggested setting
Bot handle	Resource name	luis-csharp-bot- + <your-name>, for example, luis-csharp-bot-johnsmith
Subscription	Subscription where to create bot.	Your primary subscription.
Resource group	Logical group of Azure resources	Create a new group to store all resources used with this bot, name the group luis-csharp-bot-resource-group.
Location	Azure region - this doesn't have to be the same as the LUIS authoring or publishing region.	westus
Pricing tier	Used for service request limits and billing.	F0 is the free tier.
App name	The name is used as the subdomain when your bot is deployed to the cloud (for example, humanresourcesbot.azurewebsites.net).	luis-csharp-bot- + <your-name>, for example, luis-csharp-bot-johnsmith
Bot template	Bot framework settings - see next table	
LUIS App location	Must be the same as the LUIS resource region	westus
App service plan/Location	Do not change from provided default value.	
Application Insights	Do not change from provided default value.	
Microsoft App ID and password	Do not change from provided default value.	

4. In the **Bot template**, select the following, then choose the **Select** button under these settings:

Setting	Purpose	Selection
SDK language	Programming language of bot	C#
Bot	Type of bot	Basic bot

5. Select **Create**. This creates and deploys the bot service to Azure. Part of this process creates a LUIS app named `luis-csharp-bot-xxxx`. This name is based on the /Azure Bot Service app name.

The screenshot shows the 'Web App Bot' configuration interface in the Azure portal. The left pane is titled 'Web App Bot' and contains the following settings:

- Bot handle ***: `luis-csharp-bot-johnsmith` (with a green checkmark)
- Subscription ***: `documentationteam` (dropdown)
- Resource group ***: `documentationteam` (dropdown)
- Location ***: `West US` (dropdown)
- Pricing tier**: `S1 (1K Premium Msgs/Unit)` (dropdown)
- App name ***: `luis-csharp-bot-johnsmith.azurewebsites.net` (with a green checkmark)
- *Bot template**: `Basic Bot (C#)` (dropdown)
- LUIS App location ***: `West US` (dropdown)
- LUIS Accounts ***: `luis-csharp-bot-johnsmith` (dropdown)
- *App service plan/Location**: `(new) luis-csharp-bot-luis/West ...` (dropdown)
- Application Insights**: `On` (toggle)
- Application Insights Location ***: `West US` (dropdown)
- Microsoft App ID and password**: `Auto create App ID and password` (dropdown)

The right pane is titled 'Bot template' and shows the 'Basic Bot' template selected. The 'Basic Bot' template is highlighted with a blue border and contains the following description:

Basic Bot
C#
This bot template contains the following services: **Language Understanding** and **Bot Analytics**.

At the bottom of the left pane is a blue 'Create' button. At the bottom of the right pane is a blue 'OK' button.

Wait until the bot service is created before continuing.

6. Select `Go to resource` in the notification to go to your web app bot page.

The bot has a Language Understanding model


The bot service creation process also creates a new LUIS app with intents and example utterances. The bot provides intent mapping to the new LUIS app for the following intents:

Basic bot LUIS intents	example utterance
Book flight	Travel to Paris
Cancel	bye
GetWeather	what's the weather like?
None	Anything outside the domain of the app.

Test the bot in Web Chat

1. While still in the Azure portal for the new bot, select **Test in Web Chat**.
2. In the **Type your message** textbox, enter the text Book a flight from Seattle to Berlin tomorrow. The bot responds with verification that you want to book a flight.

Test [Start over](#)



Welcome to Bot Framework!

Now that you have successfully run your bot, follow the links in this Adaptive Card to expand your knowledge of Bot Framework.

[Get an overview](#)

[Ask a question](#)

[Learn how to deploy](#)

What can I help you with today?
Say something like "Book a flight from Paris to Berlin on March 22, 2020"

Just now

Book a flight from Seattle to Berlin tomorrow



Just now

Please confirm, I have you traveling to: Berlin from: Seattle on: 2019-09-06. Is this correct?

Just now

Yes

No

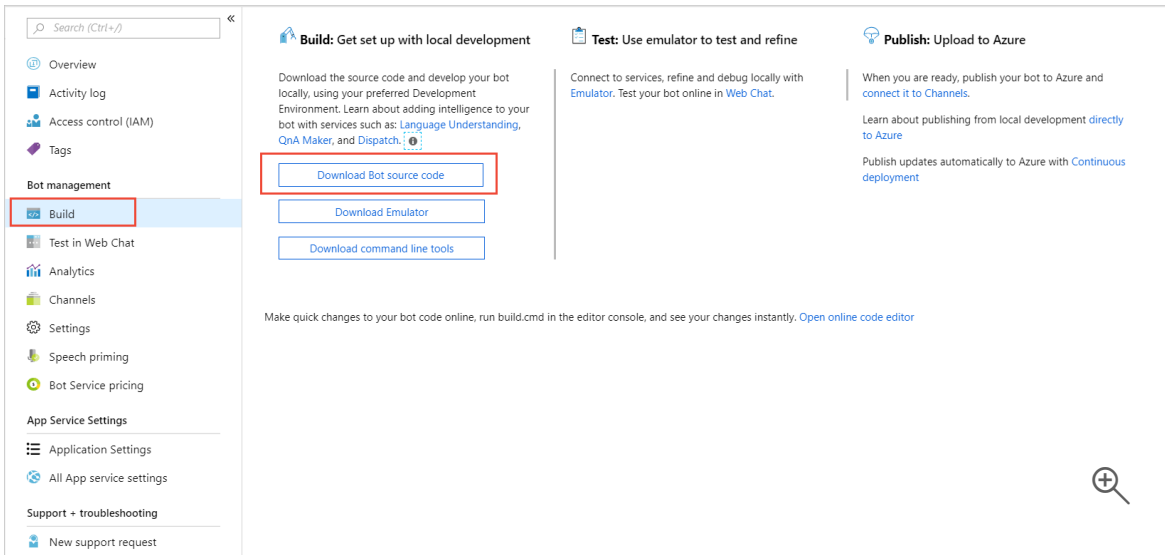
 Type your message 

You can use the test functionality to quickly testing your bot. For more complete testing, including debugging, download the bot code and use Visual Studio.

Download the web app bot source code

In order to develop the web app bot code, download the code and use on your local computer.

1. In the Azure portal, select **Build** from the **Bot management** section.
2. Select **Download Bot source code**.



3. When the pop-up dialog asks **Include app settings in the downloaded zip file?**, select **Yes**.
4. When the source code is zipped, a message will provide a link to download the code. Select the link.
5. Save the zip file to your local computer and extract the files. Open the project with Visual Studio.

Review code to send utterance to LUIS and get response

1. To send the user utterance to the LUIS prediction endpoint, open the **FlightBookingRecognizer.cs** file. This is where the user utterance entered into the bot is sent to LUIS. The response from LUIS is returned from the **RecognizeAsync** method.

C#	Copy
<pre>// Copyright (c) Microsoft Corporation. All rights reserved. // Licensed under the MIT License.</pre>	

```

using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.AI.Luis;
using Microsoft.Extensions.Configuration;

namespace Microsoft.BotBuilderSamples
{
    public class FlightBookingRecognizer : IRecognizer
    {
        private readonly LuisRecognizer _recognizer;

        public FlightBookingRecognizer(IConfiguration configuration)
        {
            var luisIsConfigured =
!string.IsNullOrEmpty(configuration["LuisAppId"]) &&
!string.IsNullOrEmpty(configuration["LuisAPIKey"]) &&
!string.IsNullOrEmpty(configuration["LuisAPIHostName"]);
            if (luisIsConfigured)
            {
                var luisApplication = new LuisApplication(
                    configuration["LuisAppId"],
                    configuration["LuisAPIKey"],
                    "https://" + configuration["LuisAPIHostName"]);

                _recognizer = new LuisRecognizer(luisApplication);
            }
        }

        // Returns true if luis is configured in the appsettings.json
        and initialized.
        public virtual bool IsConfigured => _recognizer != null;

        public virtual async Task<RecognizerResult>
RecognizeAsync(ITurnContext turnContext, CancellationToken
cancellationTokens)
            => await _recognizer.RecognizeAsync(turnContext,
cancellationTokens);

        public virtual async Task<T> RecognizeAsync<T>(ITurnContext
turnContext, CancellationToken cancellationTokens)
            where T : IRecognizerConvert, new()
            => await _recognizer.RecognizeAsync<T>(turnContext,
cancellationTokens);
    }
}

```

2. Open **Dialogs -> MainDialog.cs** captures the utterance and sends it to the executeLuisQuery in the actStep method.

C#



Copy

```
// Copyright (c) Microsoft Corporation. All rights reserved.
// Licensed under the MIT License.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.Bot.Builder;
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Schema;
using Microsoft.Extensions.Logging;
using Microsoft.Recognizers.Text.DataTypes.TimexExpression;

namespace Microsoft.BotBuilderSamples.Dialogs
{
    public class MainDialog : ComponentDialog
    {
        private readonly FlightBookingRecognizer _luisRecognizer;
        protected readonly ILogger Logger;

        // Dependency injection uses this constructor to instantiate
        MainDialog
        public MainDialog(FlightBookingRecognizer luisRecognizer,
            BookingDialog bookingDialog, ILogger<MainDialog> logger)
            : base(nameof(MainDialog))
        {
            _luisRecognizer = luisRecognizer;
            Logger = logger;

            AddDialog(new TextPrompt(nameof(TextPrompt)));
            AddDialog(bookingDialog);
            AddDialog(new WaterfallDialog(nameof(WaterfallDialog), new
WaterfallStep[]
            {
                IntroStepAsync,
                ActStepAsync,
                FinalStepAsync,
            }));

            // The initial child Dialog to run.
            InitialDialogId = nameof(WaterfallDialog);
        }

        private async Task<DialogTurnResult>
IntroStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellation_token)
        {
            if (!_luisRecognizer.IsConfigured)
            {
                await stepContext.Context.SendActivityAsync(
                    MessageFactory.Text("NOTE: LUIS is not configured.
To enable all capabilities, add 'LuisAppId', 'LuisAPIKey' and
'LuisAPIHostName' to the appsettings.json file.", inputHint:

```



```

InputHints.IgnoringInput), cancellationToken);

        return await stepContext.NextAsync(null,
cancellationToken);
    }

    // Use the text provided in FinalStepAsync or the default
    if it is the first time.
    var messageText = stepContext.Options?.ToString() ?? "What
can I help you with today?\nSay something like \"Book a flight from
Paris to Berlin on March 22, 2020\"";
    var promptMessage = MessageFactory.Text(messageText,
messageText, InputHints.ExpectingInput);
    return await stepContext.PromptAsync(nameof(TextPrompt),
new PromptOptions { Prompt = promptMessage }, cancellationToken);
}

private async Task<DialogTurnResult>
ActStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationToken)
{
    if (!_luisRecognizer.IsConfigured)
    {
        // LUIS is not configured, we just run the
BookingDialog path with an empty BookingDetailsInstance.
        return await
stepContext.BeginDialogAsync(nameof(BookingDialog), new
BookingDetails(), cancellationToken);
    }

    // Call LUIS and gather any potential booking details.
    (Note the TurnContext has the response to the prompt.)
    var luisResult = await
_luisRecognizer.RecognizeAsync<FlightBooking>(stepContext.Context,
cancellationToken);
    switch (luisResult.TopIntent().intent)
    {
        case FlightBooking.Intent.BookFlight:
            await
ShowWarningForUnsupportedCities(stepContext.Context, luisResult,
cancellationToken);

            // Initialize BookingDetails with any entities we
may have found in the response.
            var bookingDetails = new BookingDetails()
            {
                // Get destination and origin from the
composite entities arrays.
                Destination = luisResult.ToEntities.Airport,
                Origin = luisResult.FromEntities.Airport,
                TravelDate = luisResult.TravelDate,
            };

            // Run the BookingDialog giving it whatever details
we have from the LUIS call, it will fill out the remainder.

```

```

        return await
stepContext.BeginDialogAsync(nameof(BookingDialog), bookingDetails,
cancellationTokens);

        case FlightBooking.Intent.GetWeather:
            // We haven't implemented the GetWeatherDialog so
            we just display a TODO message.
            var getWeatherMessageText = "TODO: get weather flow
            here";

            var getWeatherMessage =
MessageFactory.Text(getWeatherMessageText, getWeatherMessageText,
InputHints.IgnoringInput);
            await
stepContext.Context.SendActivityAsync(getWeatherMessage,
cancellationTokens);
            break;

        default:
            // Catch all for unhandled intents
            var didntUnderstandMessageText = $"Sorry, I didn't
            get that. Please try asking in a different way (intent was
            {luisResult.TopIntent().intent})";
            var didntUnderstandMessage =
MessageFactory.Text(didntUnderstandMessageText,
didntUnderstandMessageText, InputHints.IgnoringInput);
            await
stepContext.Context.SendActivityAsync(didntUnderstandMessage,
cancellationTokens);
            break;
    }

    return await stepContext.NextAsync(null,
cancellationTokens);
}

// Shows a warning if the requested From or To cities are
// recognized as entities but they are not in the Airport entity list.
// In some cases LUIS will recognize the From and To composite
// entities as a valid cities but the From and To Airport values
// will be empty if those entity values can't be mapped to a
// canonical item in the Airport.
private static async Task
ShowWarningForUnsupportedCities(ITurnContext context, FlightBooking
luisResult, CancellationToken cancellationToken)
{
    var unsupportedCities = new List<string>();

    var fromEntities = luisResult.FromEntities;
    if (!string.IsNullOrEmpty(fromEntities.From) &&
string.IsNullOrEmpty(fromEntities.Airport))
    {
        unsupportedCities.Add(fromEntities.From);
    }

    var toEntities = luisResult.ToEntities;

```

```

        if (!string.IsNullOrEmpty(toEntities.To) &&
string.IsNullOrEmpty(toEntities.Airport))
        {
            unsupportedCities.Add(toEntities.To);
        }

        if (unsupportedCities.Any())
        {
            var messageText = $"Sorry but the following airports
are not supported: {string.Join(',', unsupportedCities)}";
            var message = MessageFactory.Text(messageText,
messageText, InputHints.IgnoringInput);
            await context.SendActivityAsync(message,
cancellationTokens);
        }
    }

    private async Task<DialogTurnResult>
FinalStepAsync(WaterfallStepContext stepContext, CancellationToken
cancellationTokens)
    {
        // If the child dialog ("BookingDialog") was cancelled, the
user failed to confirm or if the intent wasn't BookFlight
// the Result here will be null.
        if (stepContext.Result is BookingDetails result)
        {
            // Now we have all the booking details call the booking
service.

            // If the call to the booking service was successful
tell the user.

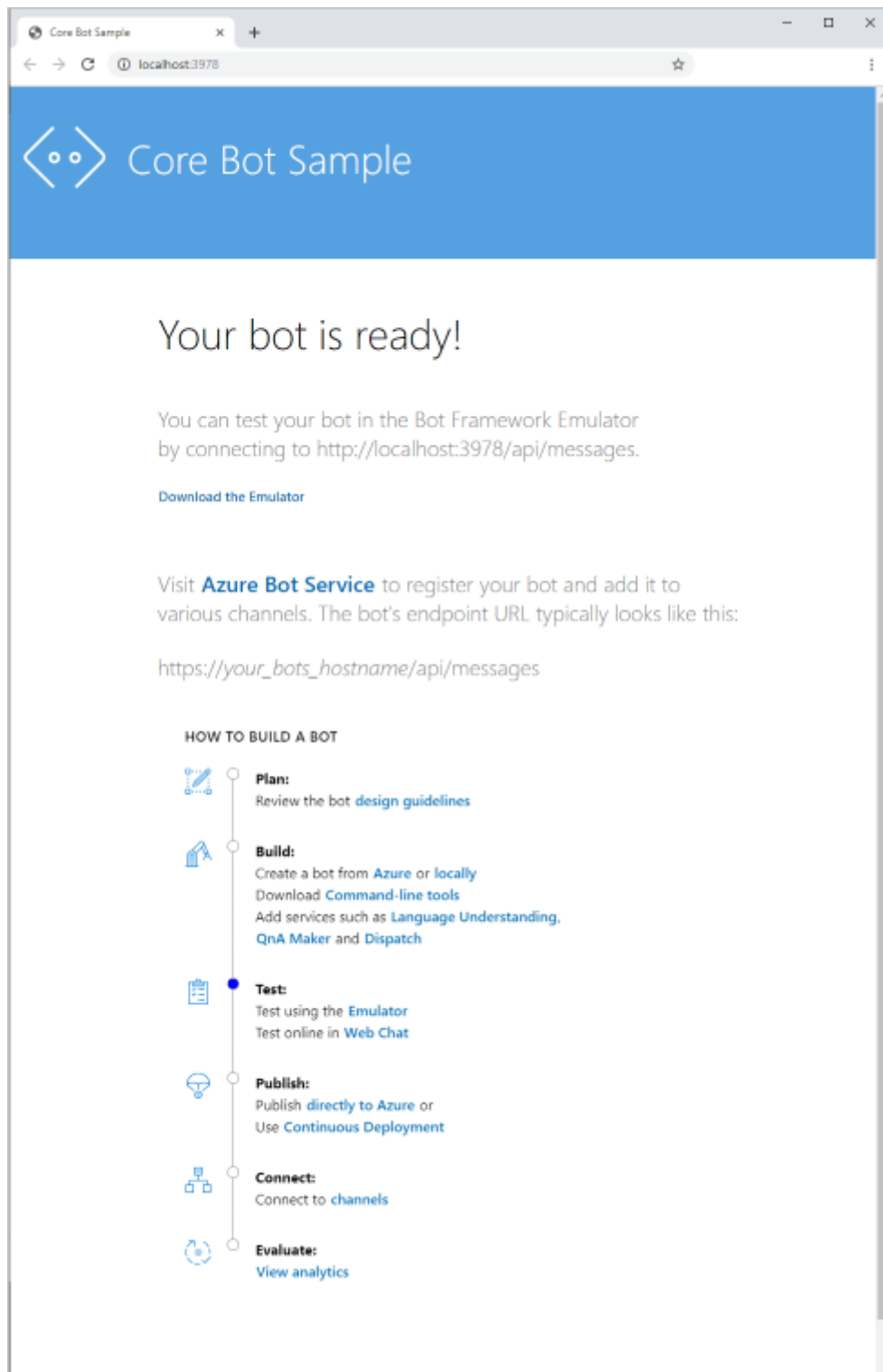
            var timeProperty = new
TimexProperty(result.TravelDate);
            var travelDateMsg =
timeProperty.ToNaturalLanguage(DateTime.Now);
            var messageText = $"I have you booked to
{result.Destination} from {result.Origin} on {travelDateMsg}";
            var message = MessageFactory.Text(messageText,
messageText, InputHints.IgnoringInput);
            await stepContext.Context.SendActivityAsync(message,
cancellationTokens);
        }

        // Restart the main dialog with a different message the
second time around
        var promptMessage = "What else can I do for you?";
        return await
stepContext.ReplaceDialogAsync(InitialDialogId, promptMessage,
cancellationTokens);
    }
}

```

Start the bot code in Visual Studio

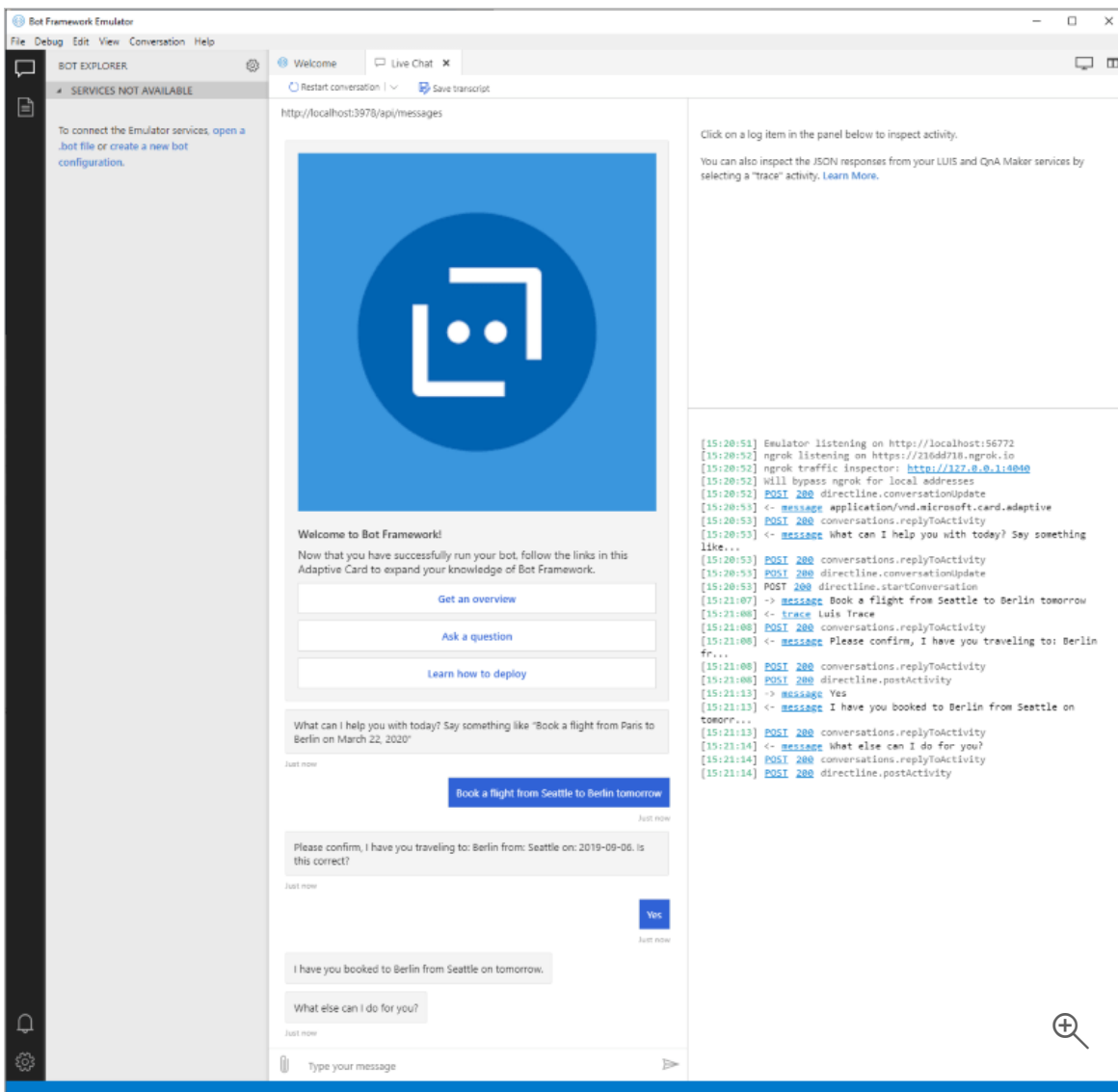
In Visual Studio 2019, start the bot. A browser window opens with the web app bot's web site at `http://localhost:3978/`. A home page displays with information about your bot.



Use the bot emulator to test the bot

1. Begin the Bot Emulator and select **Open Bot**.

2. In the **Open a bot** pop-up dialog, enter your bot URL, such as `http://localhost:3978/api/messages`. The `/api/messages` route is the web address for the bot.
3. Enter the **Microsoft App ID** and **Microsoft App password**, found in the **appsettings.json** file in the root of the bot code you downloaded, then select **Connect**.
4. In the bot emulator, enter `Book a flight from Seattle to Berlin tomorrow` and get the same response for the basic bot as you received in the **Test in Web Chat** in a previous section.



5. Select **Yes**. The bot responds with a summary of its actions.
6. From the log of the bot emulator, select the line that includes `<- trace LuisV3 Trace`. This displays the JSON response from LUIS for the intent and entities of the utterance.

The screenshot shows the Bot Framework Emulator interface. On the left, a chat window displays a conversation where the user asks to book a flight from Seattle to Berlin for tomorrow. The bot responds with a confirmation and a button to book the flight. The right pane shows the LUIS recognizer result, which includes the intent 'BookFlight', entities like 'From', 'To', and 'datetime', and the recognizer result JSON. The bottom pane shows the emulator logs, which include the bot's response and the user's confirmation.

Recognizer Result

```

{
  "startIndex": 19,
  "text": "Seattle",
  "type": "From",
  "endIndex": 36,
  "modelType": "Composite",
  "recognitionSources": [
    {
      "startIndex": 30,
      "text": "Berlin",
      "type": "To",
      "endIndex": 45,
      "modelType": "Prebuilt",
      "recognitionSources": [
        {
          "startIndex": 37,
          "text": "tomorrow",
          "type": "builtin.datetimeV2.date"
        }
      ]
    }
  ],
  "From": [
    {
      "$instance": {
        "Airport": [
          {
            "endIndex": 26,
            "modelType": "List"
          }
        ],
        "recognitionSources": [
          {
            "startIndex": 19,
            "text": "Seattle",
            "type": "Airport"
          }
        ]
      }
    }
  ]
}

```

Emulator Logs

```

[14:05:57] Emulator listening on http://[::]:53798
[14:05:57] ngrok not configured (only needed when connecting to remotely hosted bots)
[14:05:57] Connecting to bots hosted remotely
[14:05:57] Edit ngrok settings
[14:05:58] -> conversationUpdate
[14:06:01] -> message application/vnd.microsoft.card.adaptive
[14:06:01] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:01] -> message What can I help you with today? Say something like...
[14:06:01] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:01] POST 200 directline/conversations/<conversationId>/activities
[14:06:13] -> message Book a flight from Seattle to Berlin tomorrow
[14:06:14] -> trace LuisV3 Trace
[14:06:14] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:14] -> message Please confirm, I have you traveling to: Berlin fr...
[14:06:14] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:14] POST 200 directline/conversations/<conversationId>/activities
[14:06:17] -> message Yes
[14:06:17] -> message I have you booked to Berlin from Seattle on tomorr...
[14:06:17] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:17] -> message What else can I do for you?
[14:06:17] POST 200 conversations/<conversationId>/activities/<activityId>
[14:06:17] POST 200 directline/conversations/<conversationId>/activities

```

More information about bots

For more information about using this service with bots, begin with the following resources:

Resource	Purpose
Azure Bot service	The Azure Bot service provides a complete cloud-hosted web service with a bot endpoint. The services uses Bot framework , which is available in several languages.
Bot Framework	The Microsoft Bot Framework is a comprehensive platform for building enterprise-grade conversational AI experiences.

Resource	Purpose
Bot Framework Emulator	The Bot Framework Emulator is a cross-platform desktop application that allows bot developers to test and debug bots built using the Bot Framework SDK. You can use the Bot Framework Emulator to test bots running locally on your machine or to connect to bots running remotely.
Bot tools	The Bot Framework tools are a collection of cross-platform command line tools designed to cover end-to-end bot development workflow.
Bot builder samples	Full-developed bot samples are designed to illustrate scenarios you'll need to implement to build great bots.

Next steps

See more [samples](#) with conversational bots.

Build a Language Understanding app with a custom subject domain

Is this page helpful?

 Yes  No