Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS | San Diego

Building Business Applications Using Bots

Michael Washington

Programmer

AiHelpWebsite.com

Level: Introductory/Intermediate

Code Again for the First Time!

About Michael Washington

- Microsoft Reconnect MVP
- Microsoft Certified Professional

Agenda

- Background
  - What is the Microsoft Bot Framework V4?
  - How to get started with the Microsoft Bot Framework V4 (and how much it costs)
- Creating a Hello World! Bot
- Using Dialogs
- Using LUIS-Language Understanding Intelligent Service
- Using QnA Maker

What is the Microsoft Bot Framework V4?

The **Microsoft Bot Framework V4** allows you to create intelligent bots that interact naturally wherever your users are (text/SMS to Skype, Slack, Office 365 mail and other popular services).

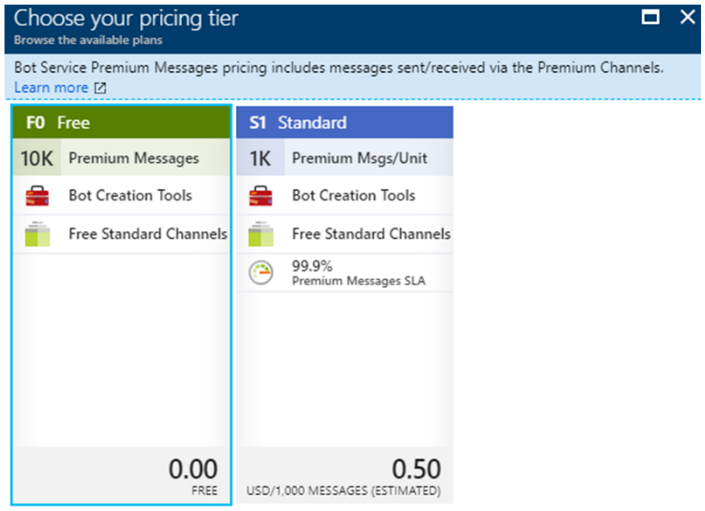
In preview now, the **Bot Builder V4 Preview SDK** offers new features and is extensible with a pluggable middleware model.

How To Get Started

Requirements

- Visual Studio 2017 (or higher) with the following workloads:
  - ASP.NET and web development
  - Azure development
  - .NET Core cross-platform development
- Bot Builder V4 SDK Template for Visual Studio
- Bot Framework Emulator (download the latest version even if it is "Alpha")
- A Microsoft Azure Subscription

Pricing



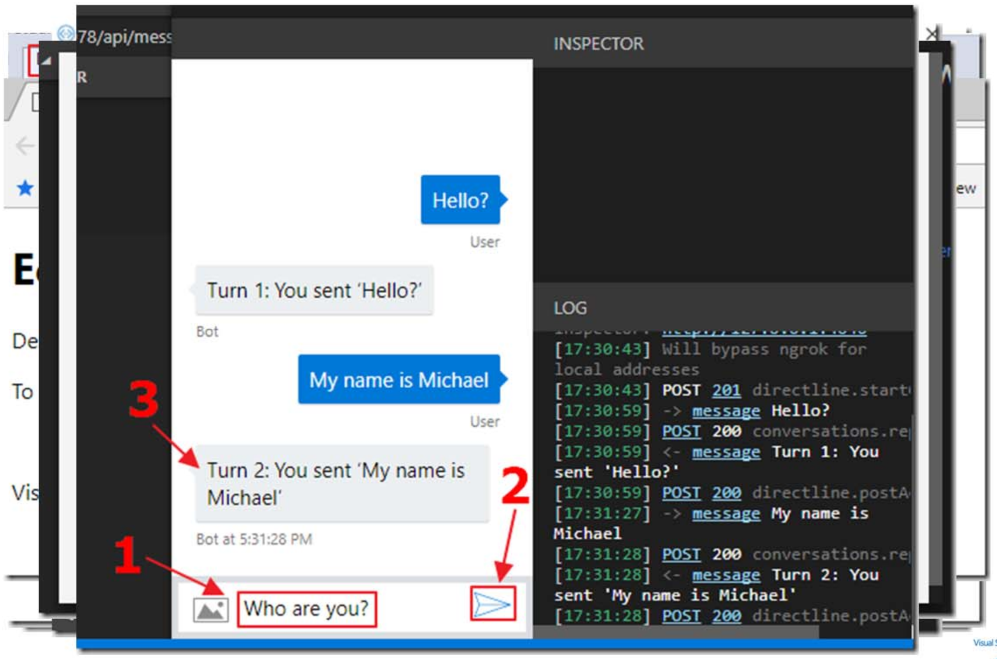
Choose your pricing tier
Browse the available plans

Bot Service Premium Messages pricing includes messages sent/received via the Premium Channels.
Learn more

F0 Free	S1 Standard
10K Premium Messages	1K Premium Msgs/Unit
Bot Creation Tools	Bot Creation Tools
Free Standard Channels	Free Standard Channels
	99.9% Premium Messages SLA
0.00 FREE	0.50 USD/1,000 MESSAGES (ESTIMATED)

+Plus hosting for website and storage

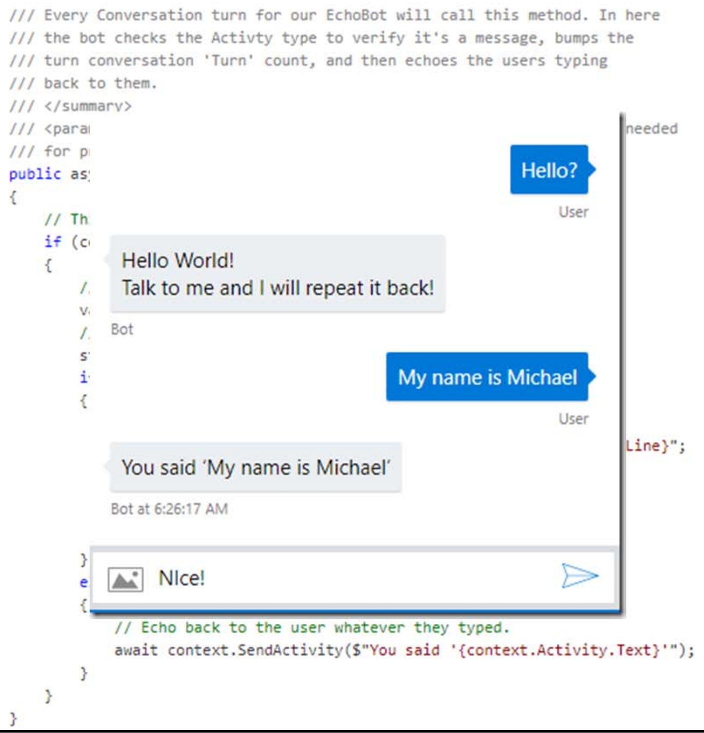
Creating A Hello World! Bot



```
[17:30:43] Will bypass ngrok for
local addresses
[17:30:43] POST 201 directline.start
[17:30:59] -> message Hello?
[17:30:59] POST 200 conversations.re
[17:30:59] <- message Turn 1: You
sent 'Hello?'
[17:30:59] POST 200 directline.postA
[17:31:27] -> message My name is
Michael
[17:31:28] POST 200 conversations.re
[17:31:28] <- message Turn 2: You
sent 'My name is Michael'
[17:31:28] POST 200 directline.postA
```

Demonstration

Update The Code



```
/// Every Conversation turn for our EchoBot will call this method. In here
/// the bot checks the Activty type to verify it's a message, bumps the
/// turn conversation 'Turn' count, and then echoes the users typing
/// back to them.
/// </summary>
/// <para                                                      needed
/// for p
public as
{
    // Th
    if (c
    {
        /
        v
        /
        s
        i
        {
                                                               Line}";
        }
        e
        {
            // Echo back to the user whatever they typed.
            await context.SendActivity($"You said '{context.Activity.Text}'");
        }
    }
}
```

Hello?

User

Hello World!
Talk to me and I will repeat it back!

Bot

My name is Michael

User

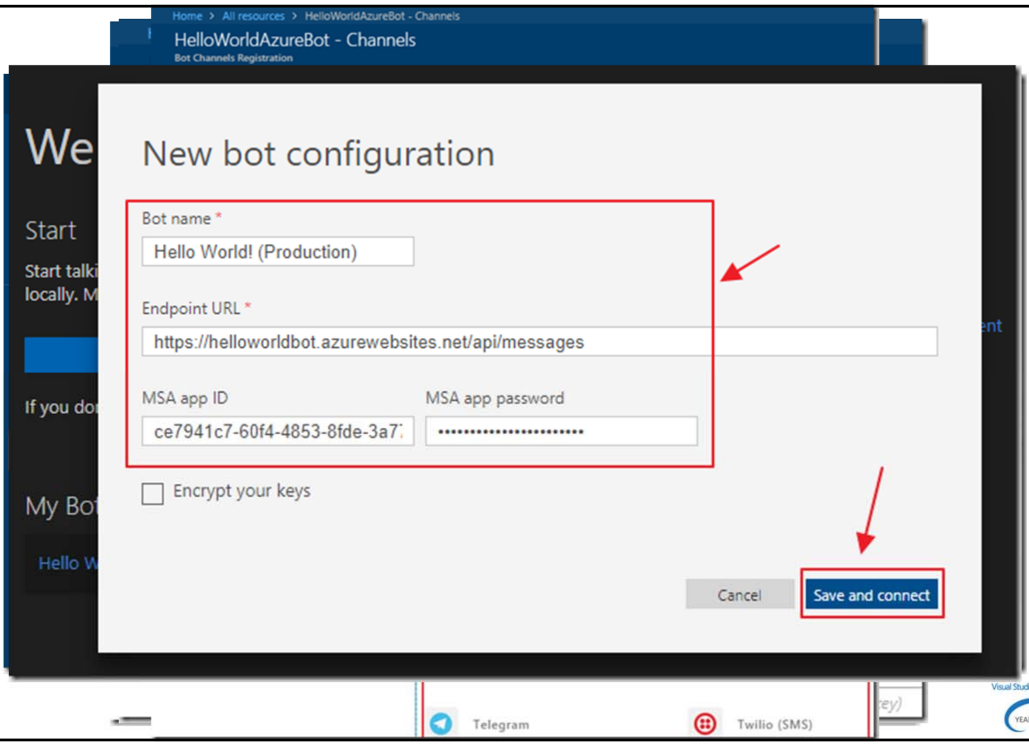
You said 'My name is Michael'

Bot at 6:26:17 AM

NIce!

Demonstration

Deploy The Bot


Home > All resources > HelloWorldAzureBot - Channels

HelloWorldAzureBot - Channels

Bot Channels Registration

New bot configuration

Bot name *: Hello World! (Production)

Endpoint URL *: https://helloworldbot.azurewebsites.net/api/messages

MSA app ID: ce7941c7-60f4-4853-8fde-3a7

MSA app password: ••••••••••••••••••••••

☐ Encrypt your keys

Demonstration

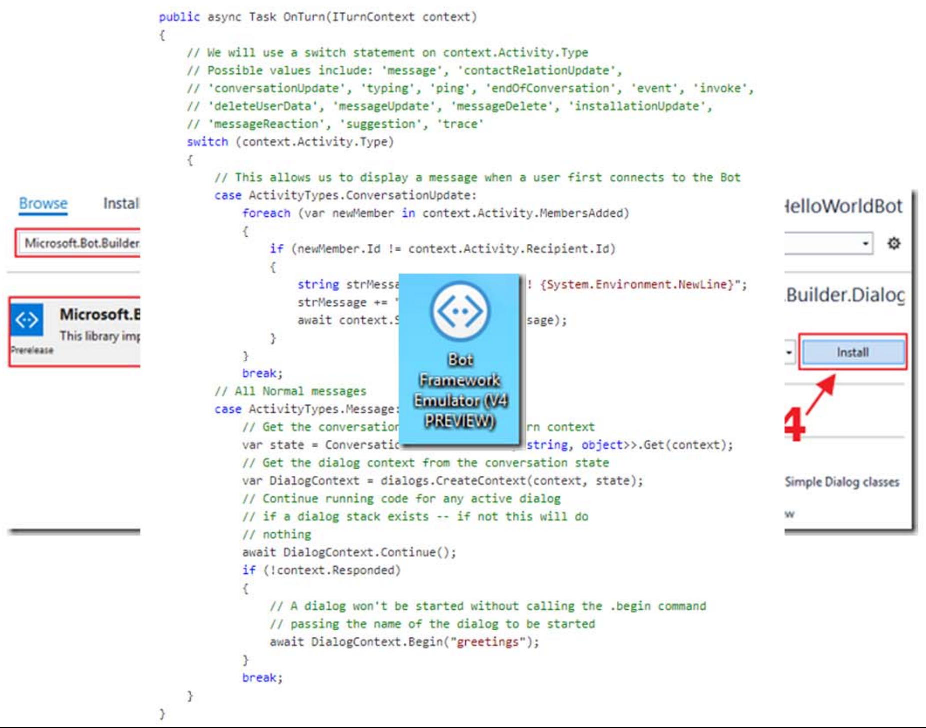
Using Dialogs

A **dialog** encapsulates application logic much like a function does in a standard program. It allows you to perform a specific task, such as gathering the details of a user’s profile, and then possibly reuse the code as needed. **Dialogs** can also be chained together in **DialogSets**.

The **Microsoft Bot Builder SDK** includes two built-in features to help you manage conversations using **dialogs**:

•DialogSets - This is a collection of **Dialogs**. To use **dialogs**, you must first create a **dialog set** to add the **dialog** to. A **dialog** can contain only a single *waterfall step*, or it can consist of multiple *waterfall steps.*

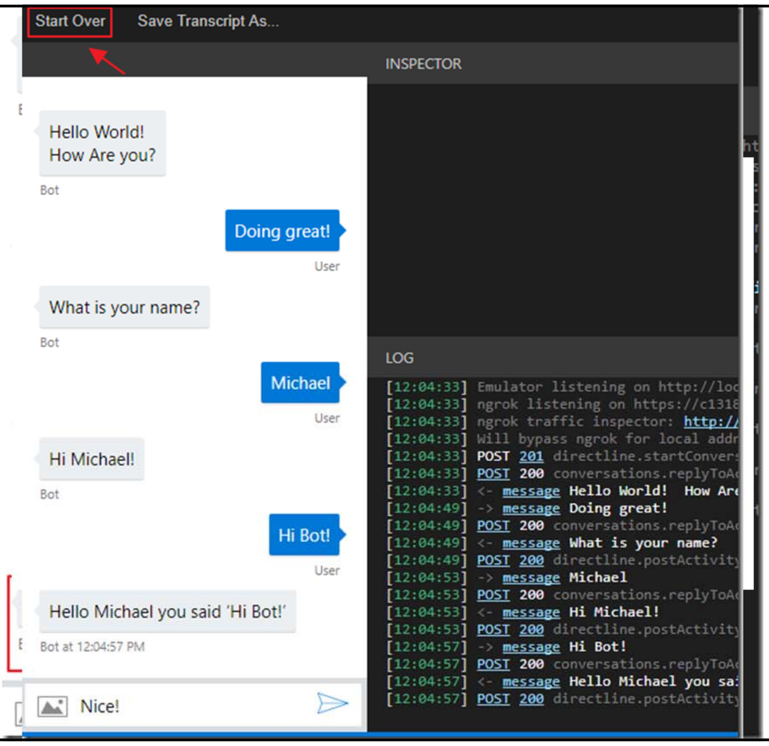
•Prompts - This provides the methods you can use to ask users for different types of information. For example, a text input, a multiple choice, or a date or number input. A prompt dialog uses at least two functions, one to prompt the user to input data, and another function to process and respond to the data.



```
public async Task OnTurn(ITurnContext context)
{
    // We will use a switch statement on context.Activity.Type
    // Possible values include: 'message', 'contactRelationUpdate',
    // 'conversationUpdate', 'typing', 'ping', 'endOfConversation', 'event', 'invoke',
    // 'deleteUserData', 'messageUpdate', 'messageDelete', 'installationUpdate',
    // 'messageReaction', 'suggestion', 'trace'
    switch (context.Activity.Type)
    {
        // This allows us to display a message when a user first connects to the Bot
        case ActivityTypes.ConversationUpdate:
            foreach (var newMember in context.Activity.MembersAdded)
            {
                if (newMember.Id != context.Activity.Recipient.Id)
                {
                    string strMessa                ! {System.Environment.NewLine}";
                    strMessage += '
                    await context.S                sage);
                }
            }
            break;
        // All Normal messages
        case ActivityTypes.Message:
            // Get the conversatior               rn context
            var state = Conversatic               string, object>>.Get(context);
            // Get the dialog context from the conversation state
            var DialogContext = dialogs.CreateContext(context, state);
            // Continue running code for any active dialog
            // if a dialog stack exists -- if not this will do
            // nothing
            await DialogContext.Continue();
            if (!context.Responded)
            {
                // A dialog won't be started without calling the .begin command
                // passing the name of the dialog to be started
                await DialogContext.Begin("greetings");
            }
            break;
    }
}
```

Demonstration

Adding User State



Hello World!
How Are you?

Bot

Doing great!

User

What is your name?

Bot

Michael

User

Hi Michael!

Bot

Hi Bot!

User

Hello Michael you said 'Hi Bot!'

Bot at 12:04:57 PM

Nice!

```
[12:04:33] Emulator listening on http://loc
[12:04:33] ngrok listening on https://c1318
[12:04:33] ngrok traffic inspector: http://
[12:04:33] Will bypass ngrok for local addr
[12:04:33] POST 201 directline.startConvers
[12:04:33] POST 200 conversations.replyToAc
[12:04:33] <- message Hello World!  How Are
[12:04:49] -> message Doing great!
[12:04:49] POST 200 conversations.replyToAc
[12:04:49] <- message What is your name?
[12:04:49] POST 200 directline.postActivity
[12:04:53] -> message Michael
[12:04:53] POST 200 conversations.replyToAc
[12:04:53] <- message Hi Michael!
[12:04:53] POST 200 directline.postActivity
[12:04:57] -> message Hi Bot!
[12:04:57] POST 200 conversations.replyToAc
[12:04:57] <- message Hello Michael you sai
[12:04:57] POST 200 directline.postActivity
```

Demonstration

Using LUIS Language Understanding Intelligent Service

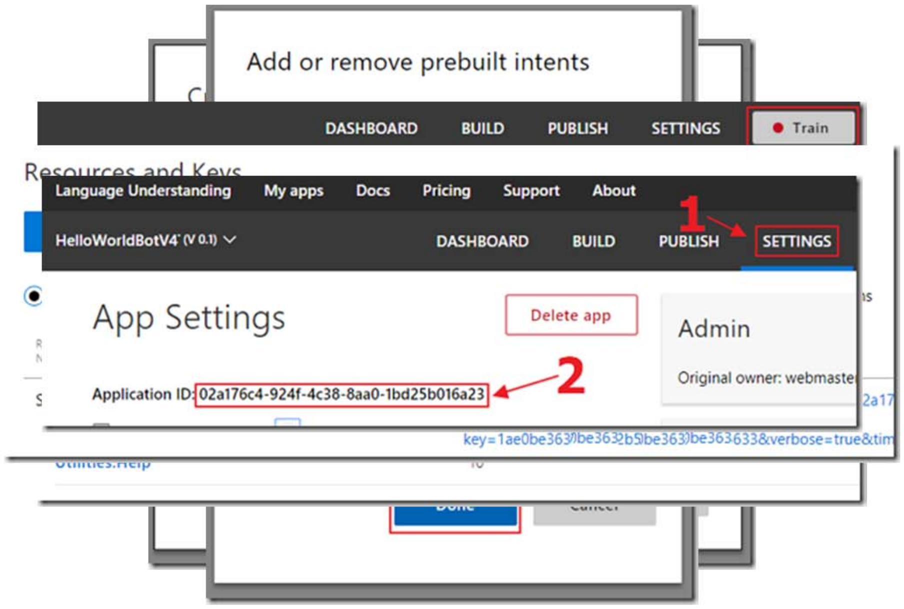
The Microsoft Language Understanding Intelligent Service (LUIS) allows you to create intelligent applications that allow your end-users to use natural language to perform actions.

The **Language Understanding Intelligent Service** (**LUIS**), allows developers to create language understanding models that are specific to their problem domain.

To use **LUIS**, developers log into the LUIS website, enter a few ***utterances*** and their ***labels***, and then deploy a model to an HTTP endpoint on Azure.

The utterances sent to the endpoint are logged and labeled. The website allows the developer (or any application administrator) to train the application by identifying issues, which can be resolved by either adding more labels or by giving hints to the machine learner in the form of features.
A developer can create and deploy a model in minutes, and then maintain and train it as the usage of the application grows.

Create The LUIS Application

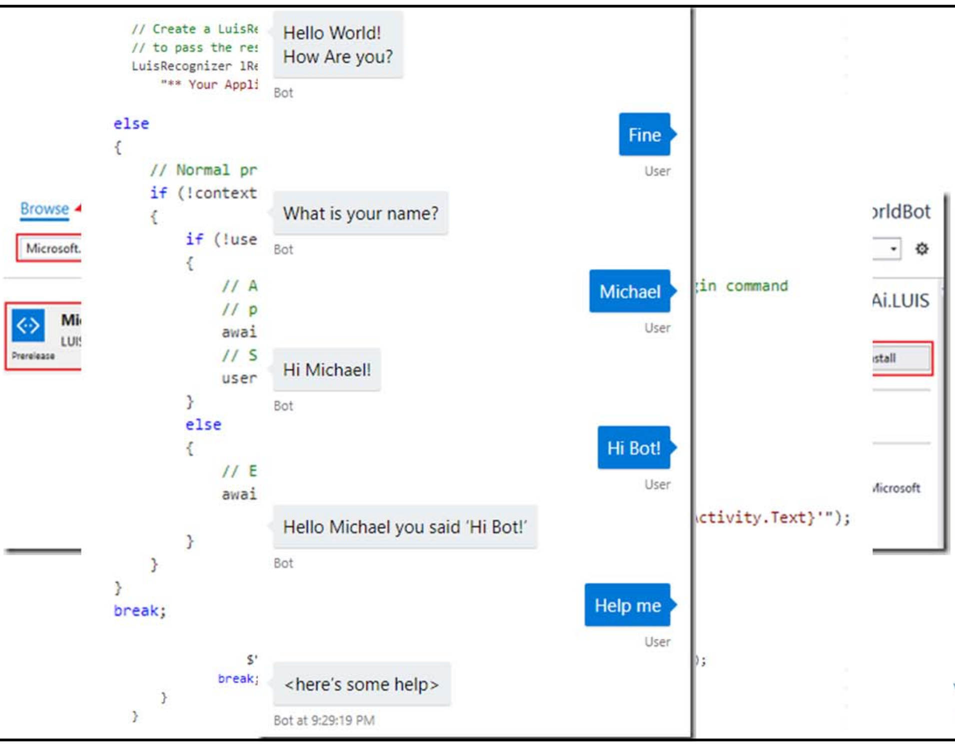


App Settings

Application ID: 02a176c4-924f-4c38-8aa0-1bd25b016a23

key=1ae0be363?be363?b5?be363?be363633&verbose=true&tim

Update The Bot Code



```
// Create a LuisRe
// to pass the re:
LuisRecognizer lRe
    "** Your Appli

else
{
    // Normal pr
    if (!context
    {
        if (!use
        {
            // A
            // p
            awai
            // S
            user
        }
        else
        {
            // E
            awai

        }
    }
}
break;

                  $'
              break;
          }
      }
```

Hello World!
How Are you?
Bot

Fine
User

What is your name?
Bot

Michael
User

Hi Michael!
Bot

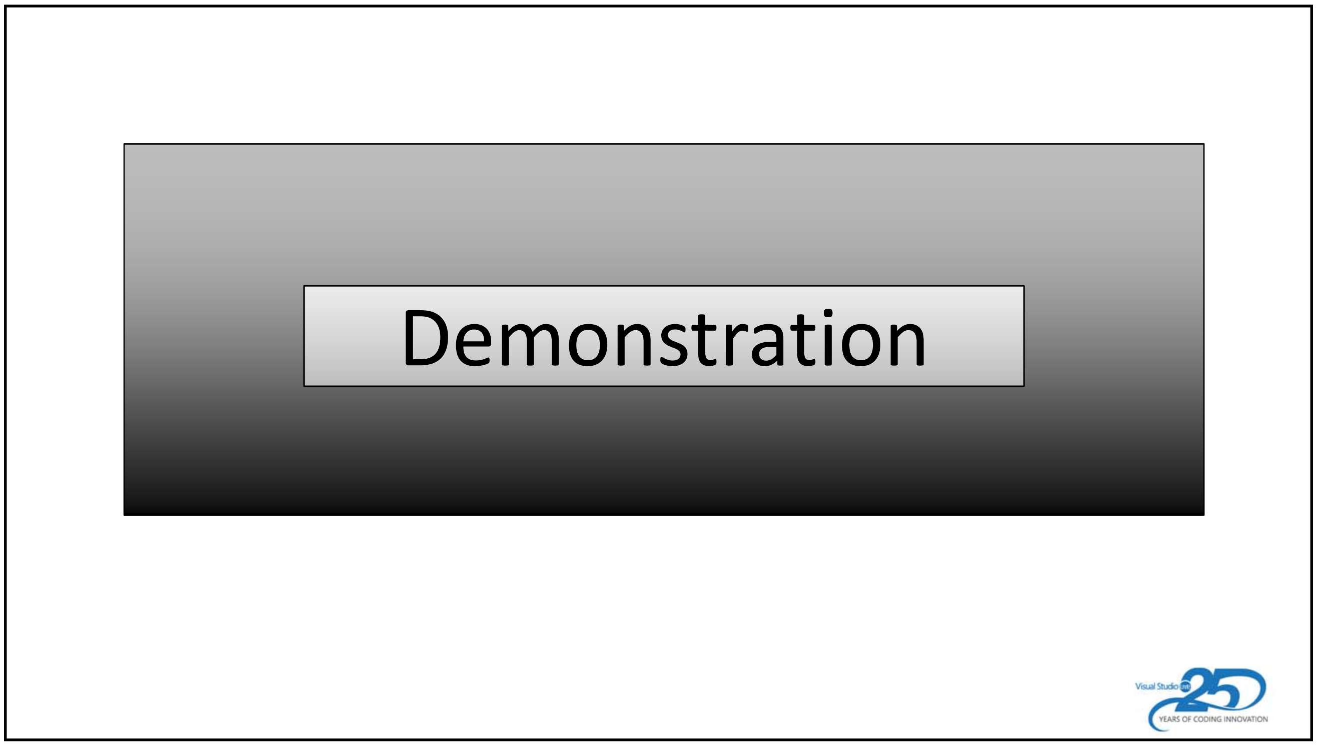
Hi Bot!
User

Hello Michael you said 'Hi Bot!'
Bot

Help me
User

<here's some help>
Bot at 9:29:19 PM

```
;in command
ctivity.Text}'");
);
```



Demonstration

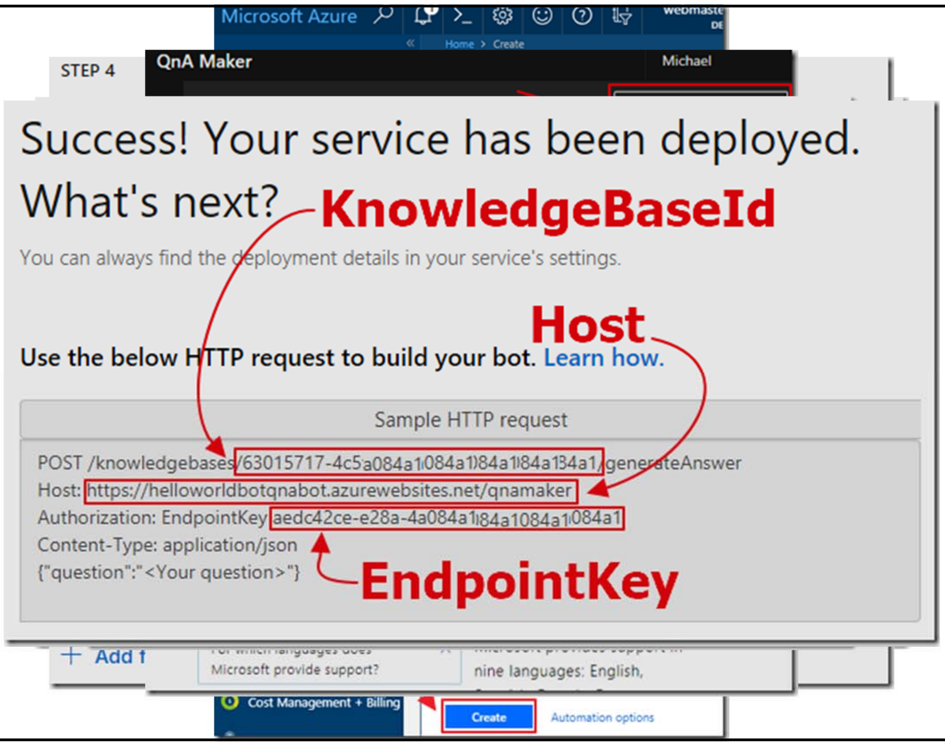
Using QnA Maker

According to the Microsoft QnA Maker website, the **QnA Maker** allows you to: "*Build, train and publish a simple question and answer bot based on FAQ URLs, structured documents, product manuals or editorial content in minutes*".

This really speeds your **Bot** development because it leverages the abilities of LUIS (Language Understanding Intelligent Service) to understand human conversation, and combines it with a service that allows you to provide a list of questions and answers, or simply upload a product manual or provide a link to a website, and the questions and answers will be parsed for you automatically.

The end result is an incredible amount of functionality that you can add to your **Bot** easily.

Create The QnA Maker Service



Success! Your service has been deployed.

What's next?

You can always find the deployment details in your service's settings.

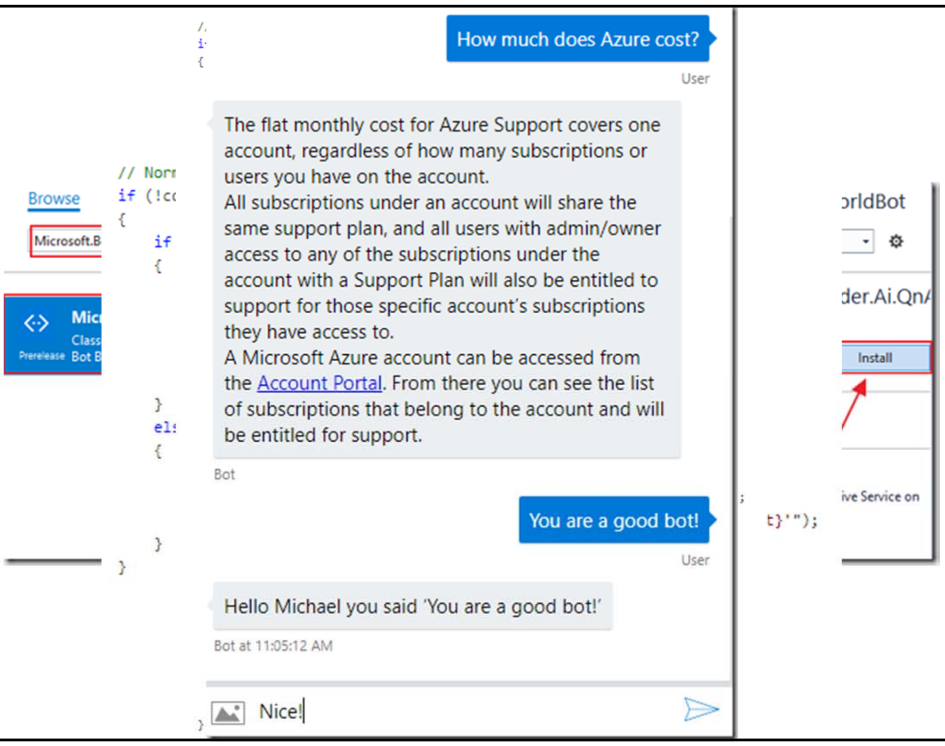
Use the below HTTP request to build your bot. Learn how.

Sample HTTP request

```
POST /knowledgebases/63015717-4c5a084a1084a1084a184a1084a1/generateAnswer
Host: https://helloworldbotqnabot.azurewebsites.net/qnamaker
Authorization: EndpointKey aedc42ce-e28a-4a084a184a1084a1084a1
Content-Type: application/json
{"question":"<Your question>"}
```

Demonstration

Update The Bot Code



How much does Azure cost?

User

The flat monthly cost for Azure Support covers one account, regardless of how many subscriptions or users you have on the account.
All subscriptions under an account will share the same support plan, and all users with admin/owner access to any of the subscriptions under the account with a Support Plan will also be entitled to support for those specific account's subscriptions they have access to.
A Microsoft Azure account can be accessed from the Account Portal. From there you can see the list of subscriptions that belong to the account and will be entitled for support.

Bot

You are a good bot!

User

Hello Michael you said 'You are a good bot!'

Bot at 11:05:12 AM

Nice!

Demonstration

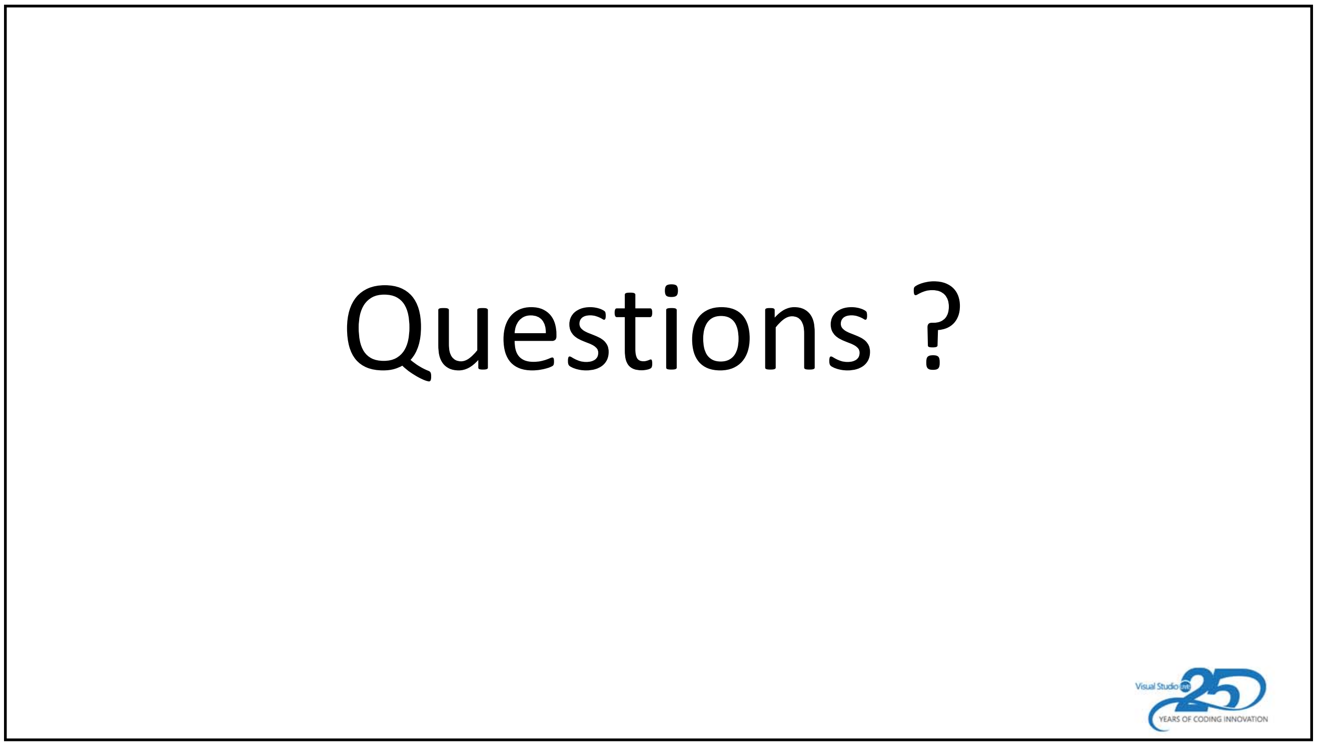
How to deploy to Skype, Cortana and Facebook

How to communicate with users through Amazon Alexa

Resources

AI Help Website
http://AIHelpWebsite.com

ADefWebserver
http://ADefWebserver.com



Questions ?

Thank You!