



Lab 9 Azure Bot Service

By: Maarten Struys

IoT Solution Architect

December, 2017

If you have any issues or concerns, please email: virtualbootcamphelp@microsoft.com.

Execution Time: 45 minutes.

Required Hardware:

- Windows 10 PC
- Access to a WiFi network (without a captive portal aka web page login)

Required Operating System:

- Windows 10

Other Requirements:

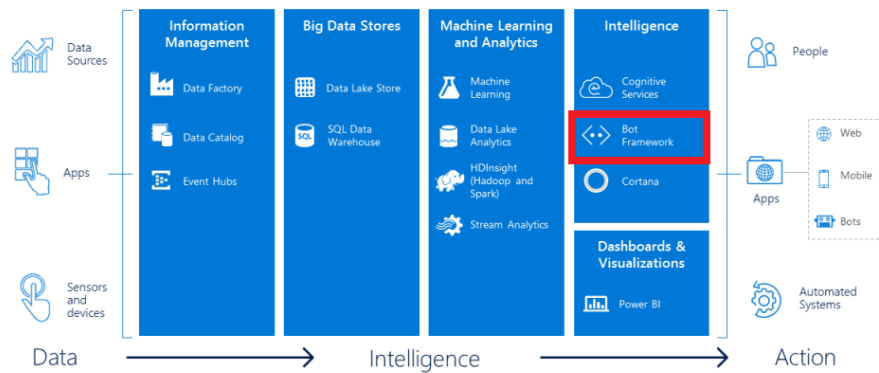
- Azure Subscription

Required Software:

Software	Size	Installation URL
Visual Studio 2017	150 MB	https://aka.ms/vs/15/release/vs_community.exe

Introduction

This lab explores how to build and host a bot using the Azure Bot Service



In this lab, you will create a bot to interact with users. Several bot features will be used including dialogs, form flows, prompt dialogs, and bot state.

Business Case

An example of a business case would be to provide support for internal or external users. This is key to maintaining their satisfaction. Ideally, this support is provided in a timely manner and without using more resources than required.

Learning Objectives

Upon completing this lab, you will have hands-on experience with the following functions and concepts related to Microsoft's Bot Framework:

- Creating a bot using the Azure Bot Service
- Testing a bot inside the Azure Bot Service
- Updating the bot using Visual Studio 2017
- Using form flow to create a bot form
- Creating a bot dialog
- Interacting with a user through the dialog

All source code that you are going to need is provided in the appendices of this document to save you typing. If you find it easier to have all source code needed in separate text files, you can download those here:

<https://1drv.ms/f/s!AmHxRmFJqwlzpqZszKlyjSPCcJ6oig>.

Create a bot with the Azure Bot Service

The Azure Bot Service accelerates the process of developing a bot by provisioning a web host with one of four bot templates you can modify in an integrated environment that is purpose-built for bot development.

Prerequisites

You must have a Microsoft Azure subscription before you can use the Azure Bot Service.

Create your bot

- Step 1. On the Azure portal click [Create a resource](#) in the menu blade.
- Step 2. In the [New](#) blade, navigate to the **AI + Cognitive Services** category, and select [Web App Bot](#).
- Step 3. In the Bot Service blade, provide the requested information, and click [Create](#) to create the bot service and deploy it to the cloud.
- Step 4. Set Bot name to your bot's name. Use the same name for **App name** (automatically populated in the textbox). The name is used as the subdomain when your bot is deployed to the cloud (for example, *mybasicbot.azurewebsites.net*). The name must be unique. You will see a green checkmark if you have entered a correct name for your bot.
- Step 5. Select the subscription to use (your subscription should be visible in the drop-down box).
- Step 6. Select the [resource group](#), [hosting plan](#), and [location](#). For this lab, **create a new** resource group and use a new **App Service Plan**. You can use a Free App Service Plan (F1) for this lab.
- Step 7. Switch off **Application Insights**. This useful feature gives you all kinds of insight in the use of your bot, but we are not going to explore application insights in this lab.
- Step 8. Click on **Create** to deploy the bot service

Microsoft Azure

<<

+

Create a resource

☰

All services

★

FAVORITES

Dashboard

All resources

Resource groups

App Services

Function Apps

SQL databases

Azure Cosmos DB

Virtual machines

Load balancers

Storage accounts

Virtual networks

Azure Active Directory

Monitor

Advisor

Security Center

Cost Management + Billing

Help + support

Home > New > Bot Service

Bot Service

Bot Service

* Bot name ⓘ
MyBasicBotMST ✓

* Subscription
Visual Studio Enterprise ▾

* Resource group
☐ Create new ☒ Use existing
MyBasicBotMST-RG ✓

* Location
West Europe ▾

Pricing tier ([View full pricing details](#))
F0 (10K Premium Messages) ▾

* App name ⓘ
MyBasicBotMST ✓
.azurewebsites.net

* Bot template
Basic (C#) >

* App service plan/Location
MyDialogBot-MST-AppService/... >

* Azure Storage ⓘ
☐ Create New ☒ Select Existing
mybasicbotmstb289 ✓

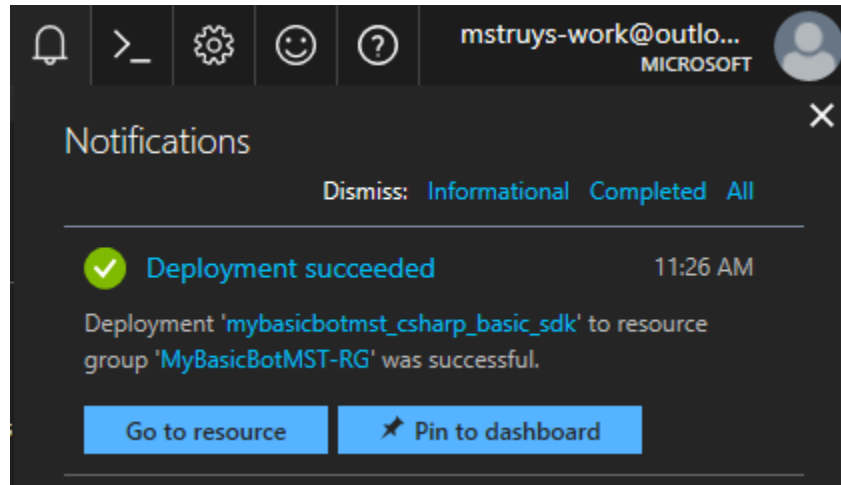
Application Insights ⓘ

☐ Pin to dashboard

[Automation options](#)

5 | Page

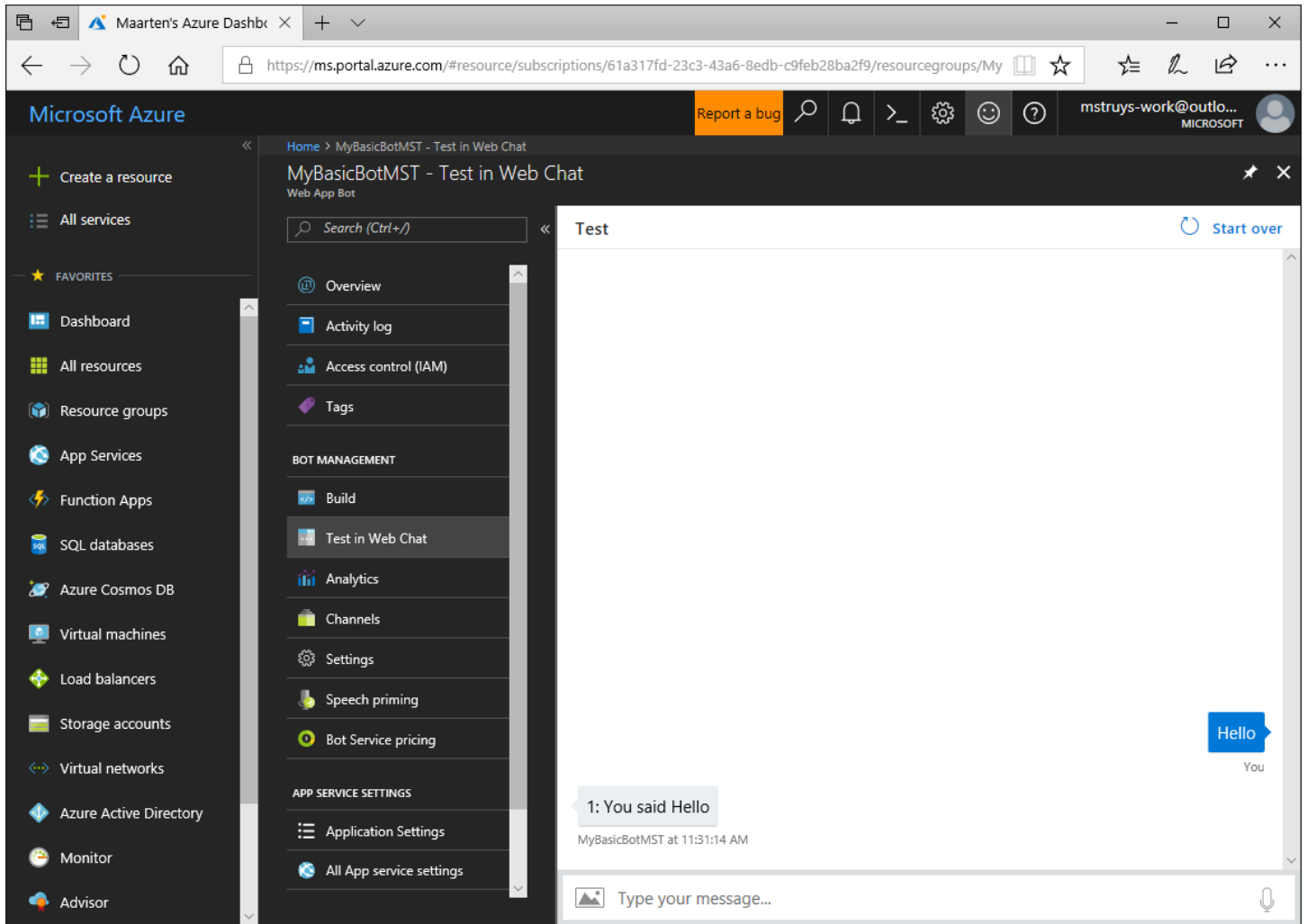
- Step 9. Confirm that the bot service has been deployed. To do so, click **Notifications** (the bell icon that is located along the top edge of the Azure portal). The notification will change from **Deployment started** to **Deployment succeeded**.
- Step 10. After the notification changes to **Deployment succeeded**, click **Go to resource** on the notification message.



Your bot has now been created. Note that we don't change anything to the settings but immediately start testing the bot.

Test the bot you have just created.

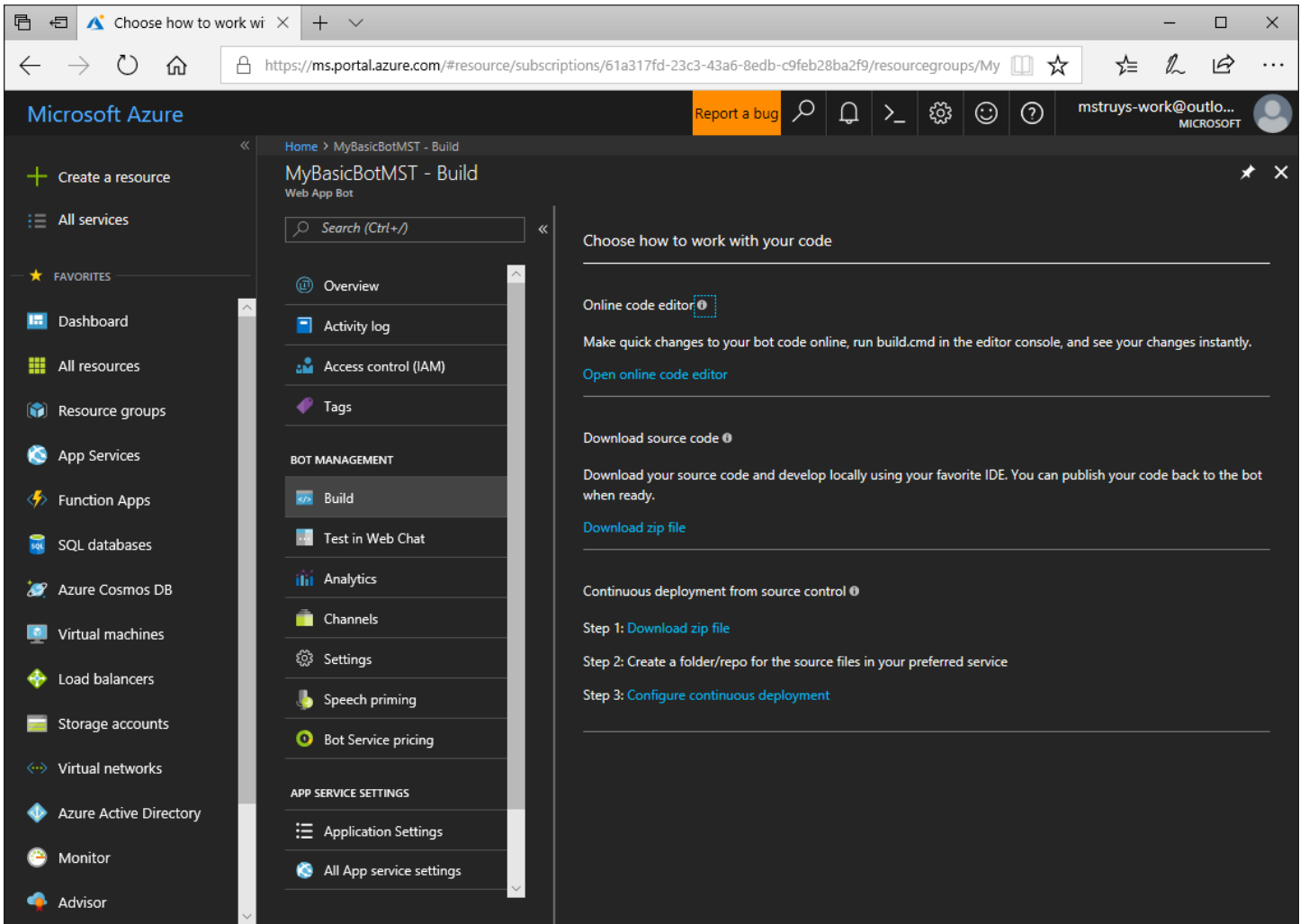
- Step 11. Select **Test in Web Chat** on your bot blade and type **Hello** as the first message to your bot. The bot will reply simply by echoing your message.



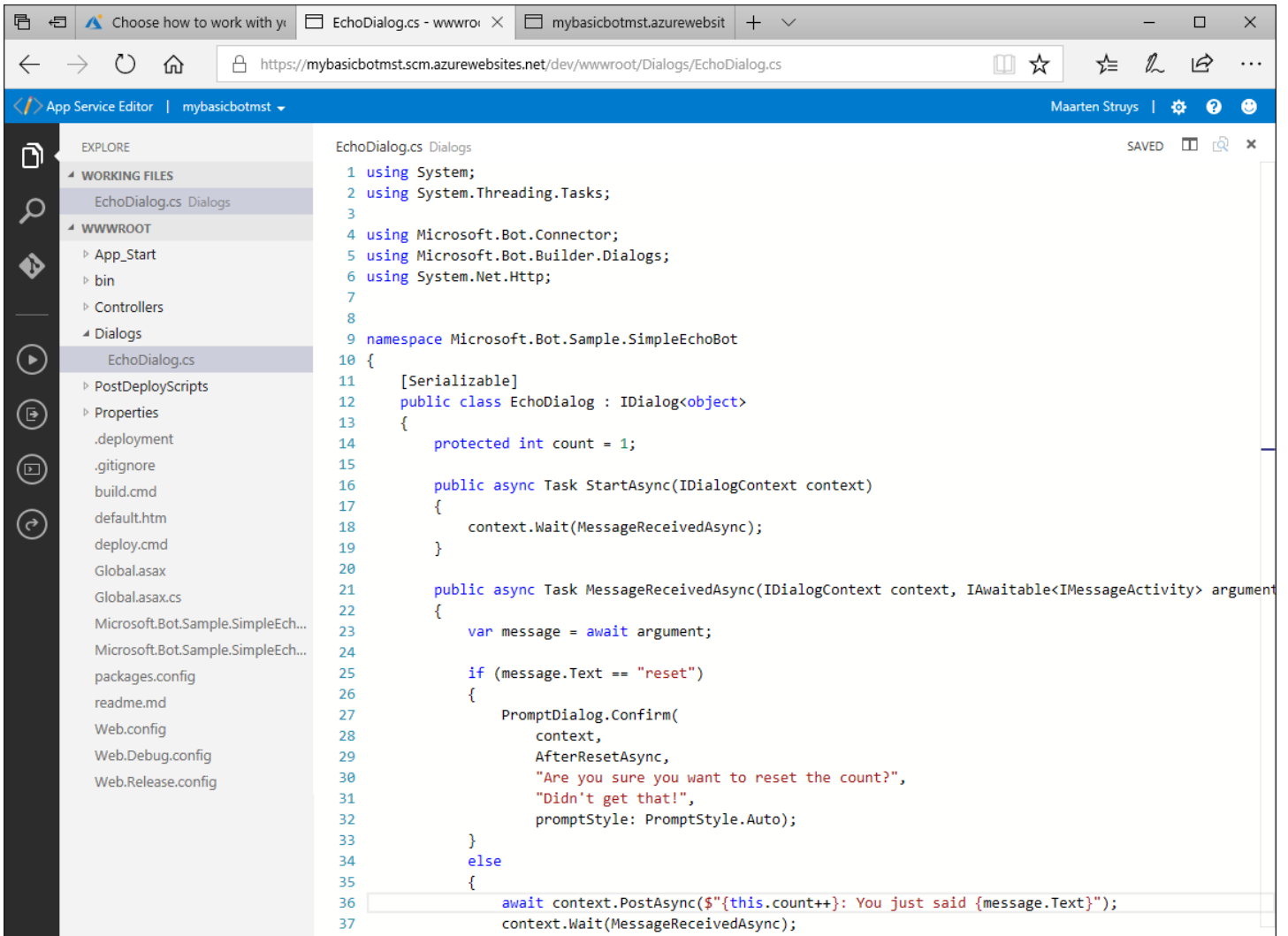
Deploy changes to your web app bot

Using the App Service plan, it is very easy to modify your bot source and re-deploy your changes.

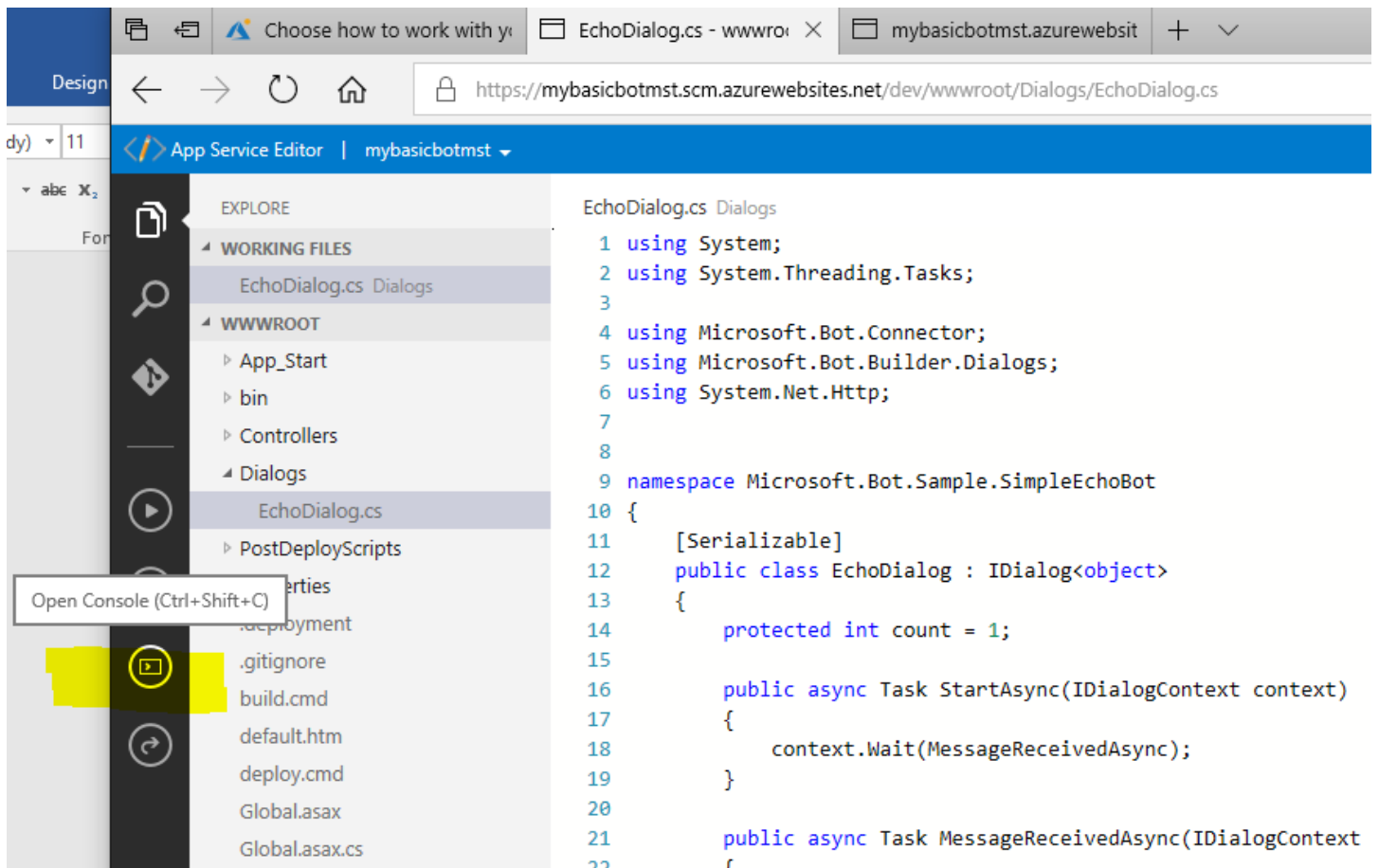
Step 12. In the bot blade, click on the **Build** tab and in the Build blade, click on **Open online code editor**.



- Step 13. Azure, click your bot's **BUILD** tab, and click **Open online code editor**. This opens a new tab in your browser that shows an online development environment. If you see an error (Unable to open 'readme.md') you can just ignore it.
- Step 14. In the **Explore** blade, expand the **Dialogs** entry and click on the file **EchoDialog.cs**.
- Step 15. On line 36 in the code editor, change the string **You said** into **You just said**. Your changes will automatically be saved, but to have the changes appear in your bot, you must build to bot.



Step 16. In the left hand menu (showing a number of icons), select the **Open Console** Icon or press **Ctrl – Shift – C** simultaneously.

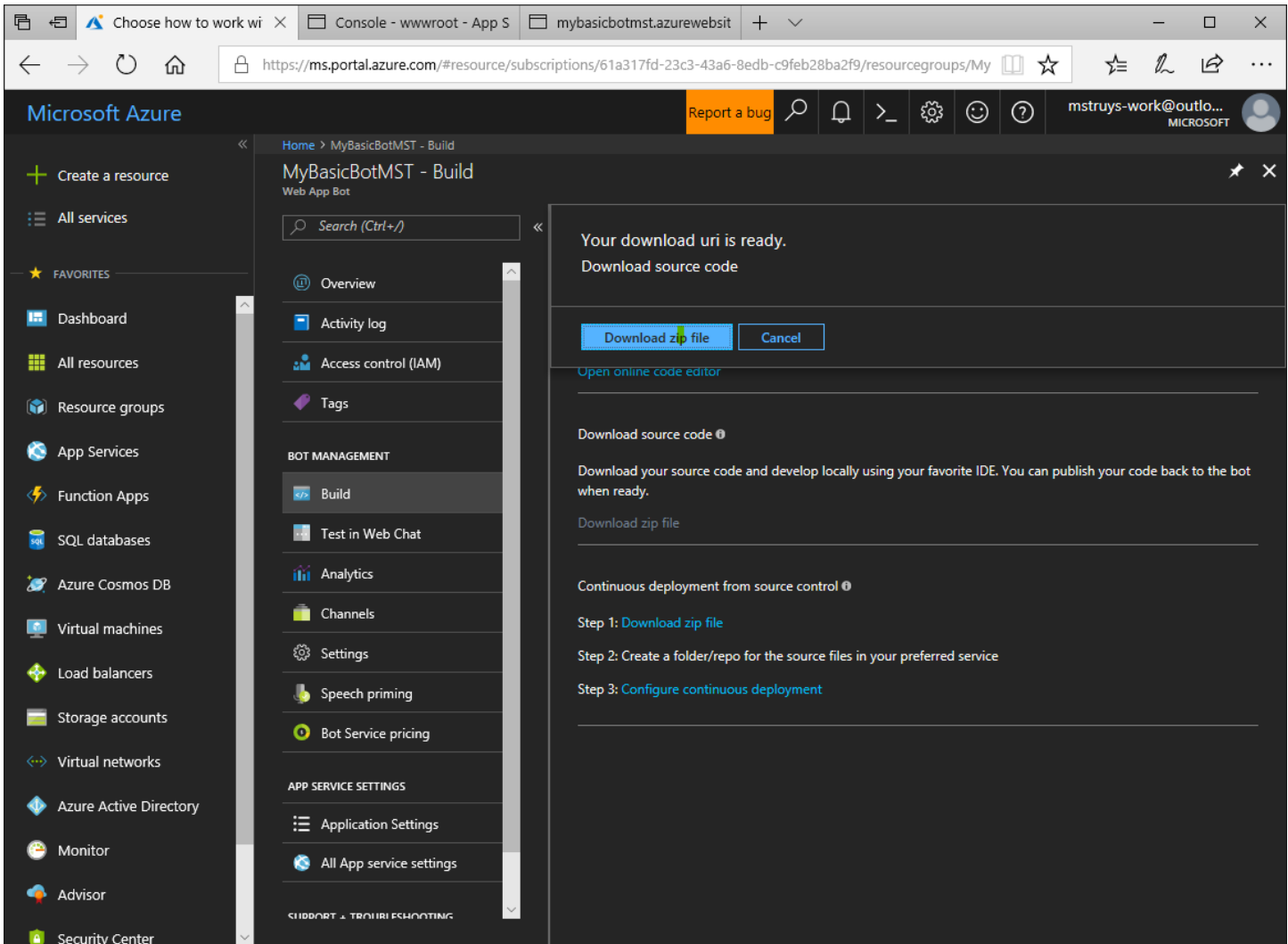


- Step 17. In the Console window, type **build.cmd** and press the enter key. The console window shows the deployment's progress until it's complete. Note: the first time it might take some time to build your bot.
- Step 18. Test your bot again by typing a few messages into the built-in chat control. Make sure to click the **Start over** button in the chat control before testing the bot again.

Adding more functionality to your bot

For small changes to your bot, it is adequate to use the online editor as you have seen in the previous section. However, it is also possible to make changes to your bot using Visual Studio 2017. In this section we are taking the existing bot and add a more dialog based interface to it. You will download the bot's solution to your development machine and open it in Visual Studio 2017. After that, you will add additional code, test the bot locally and finally upload the modified bot to your App Service Plan in Azure.

- Step 19. In Azure, click your bot's **BUILD** tab, and click **Download zip file**. At this moment, a download URI is created for you. Once the zip file is available, you see a small dialog with a download button (**Download zip file**).

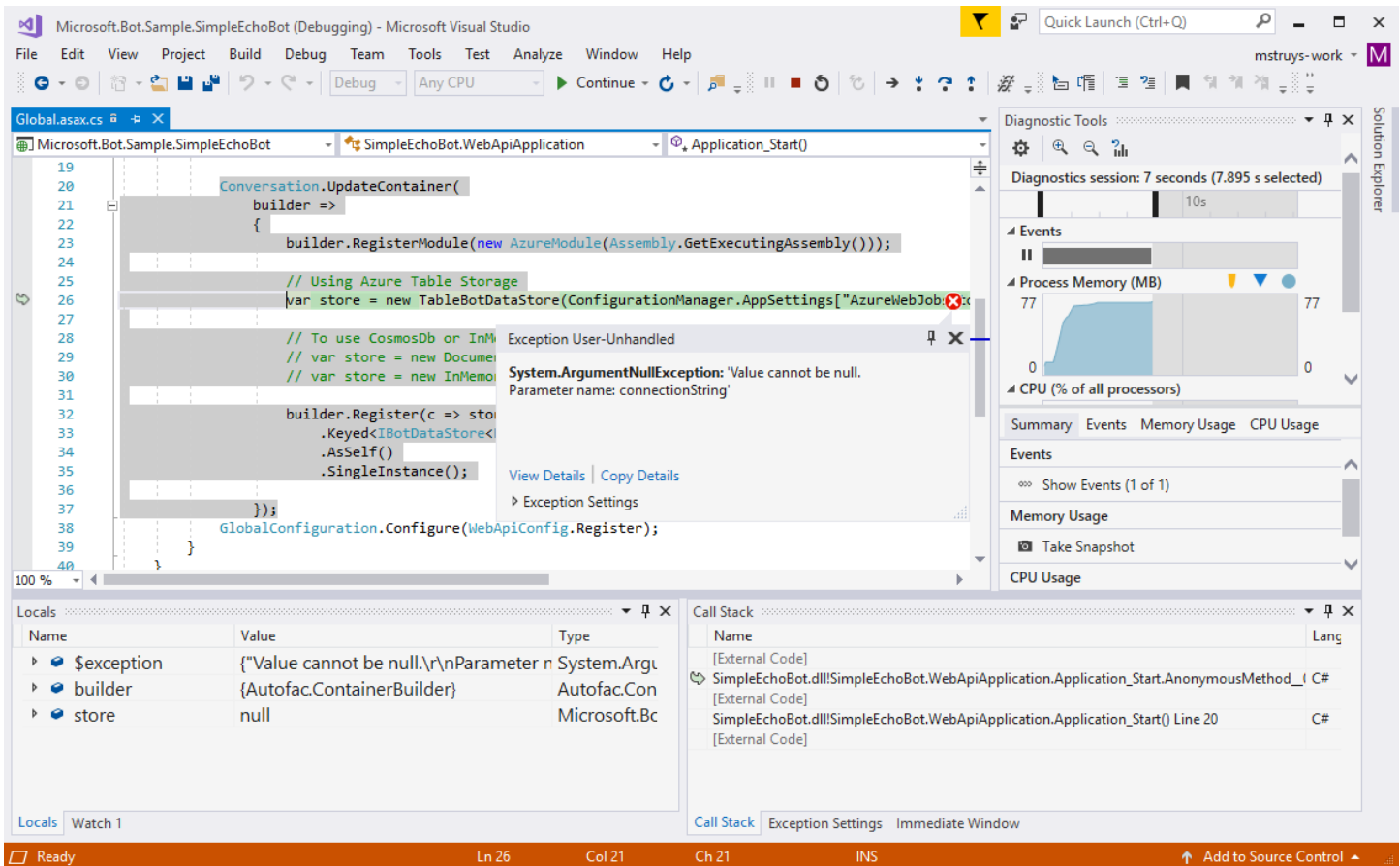


- Step 20. Store the zip file to a location on your local hard drive.
- Step 21. Unzip the file and navigate to the unzipped folder that was just created.
- Step 22. Open the solution that contains all your bot's source code in Visual Studio 2017 by clicking **File, Open** and navigating to the solution file that you just downloaded and unzipped.
- Step 23. Click **OK** on the Security Warning window.
- Step 24. Build the entire solution to make sure that all NuGet packages are downloaded to your development machine by selecting **Build – Rebuild Solution** from the Visual Studio Menu.

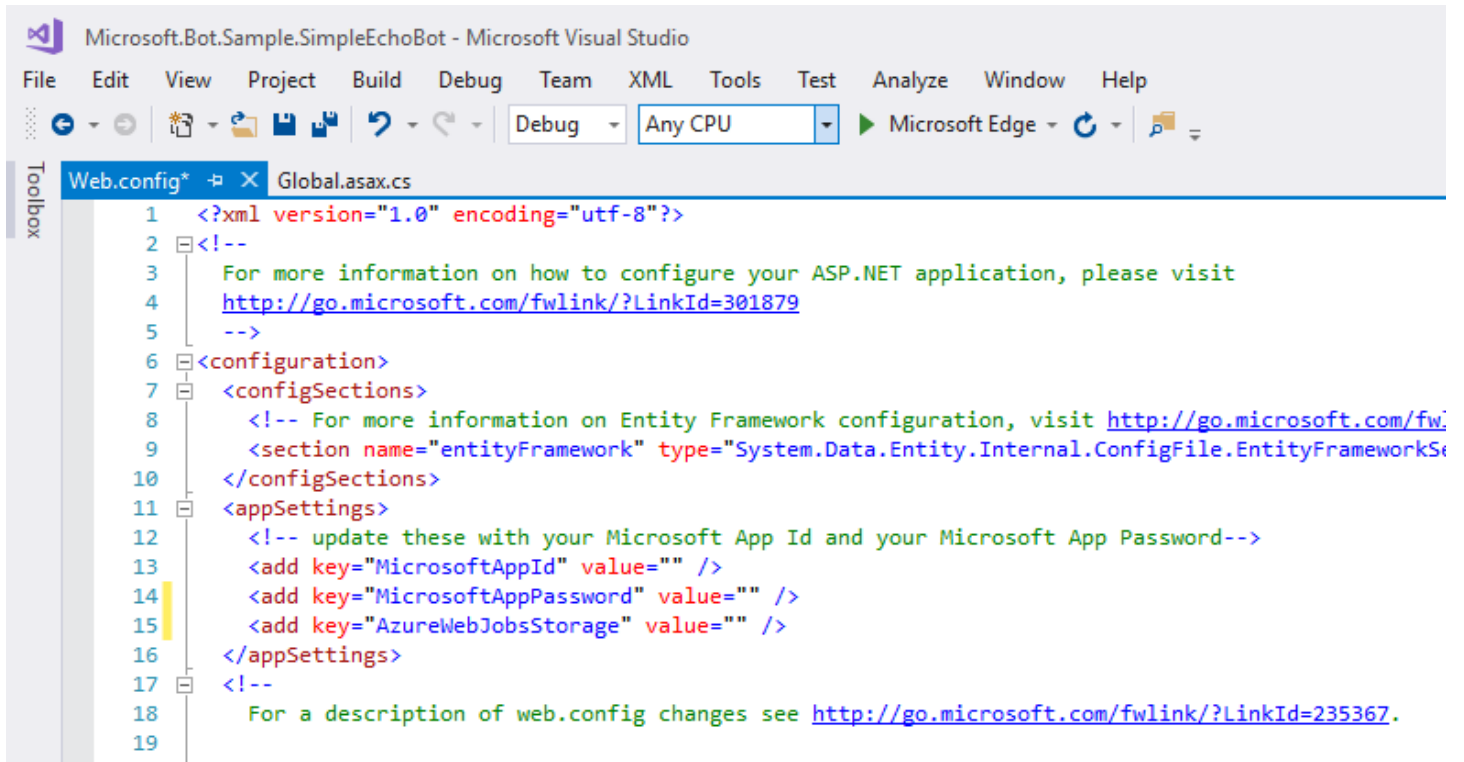
Test your bot locally using the Bot Framework Channel Emulator

The Bot Framework Emulator is a desktop application that allows bot developers to test and debug their bots, either locally or remotely. Using the emulator, you can chat with your bot and inspect the messages that your bot sends and receives. The emulator displays messages as they would appear in a web chat UI and logs JSON requests and responses as you exchange messages with your bot.

- Step 25. If you haven't done so yet, you need to download the Bot Framework Emulator. Download packages for Mac, Windows, and Linux are available via the [GitHub releases page](#). The latest Windows installer is available via the [emulator download page](#) (download starts immediately).
- Step 26. Start your bot from inside Visual Studio 2017 by clicking on the start button (you will automatically deploy the bot locally and a welcome page is shown in the browser if everything works immediately).
- Step 27. At this moment you might run into a **System.ArgumentNullException**. If this is the case, take the following steps to fix the error. If you don't see this exception in Visual Studio, you can continue to step 35.

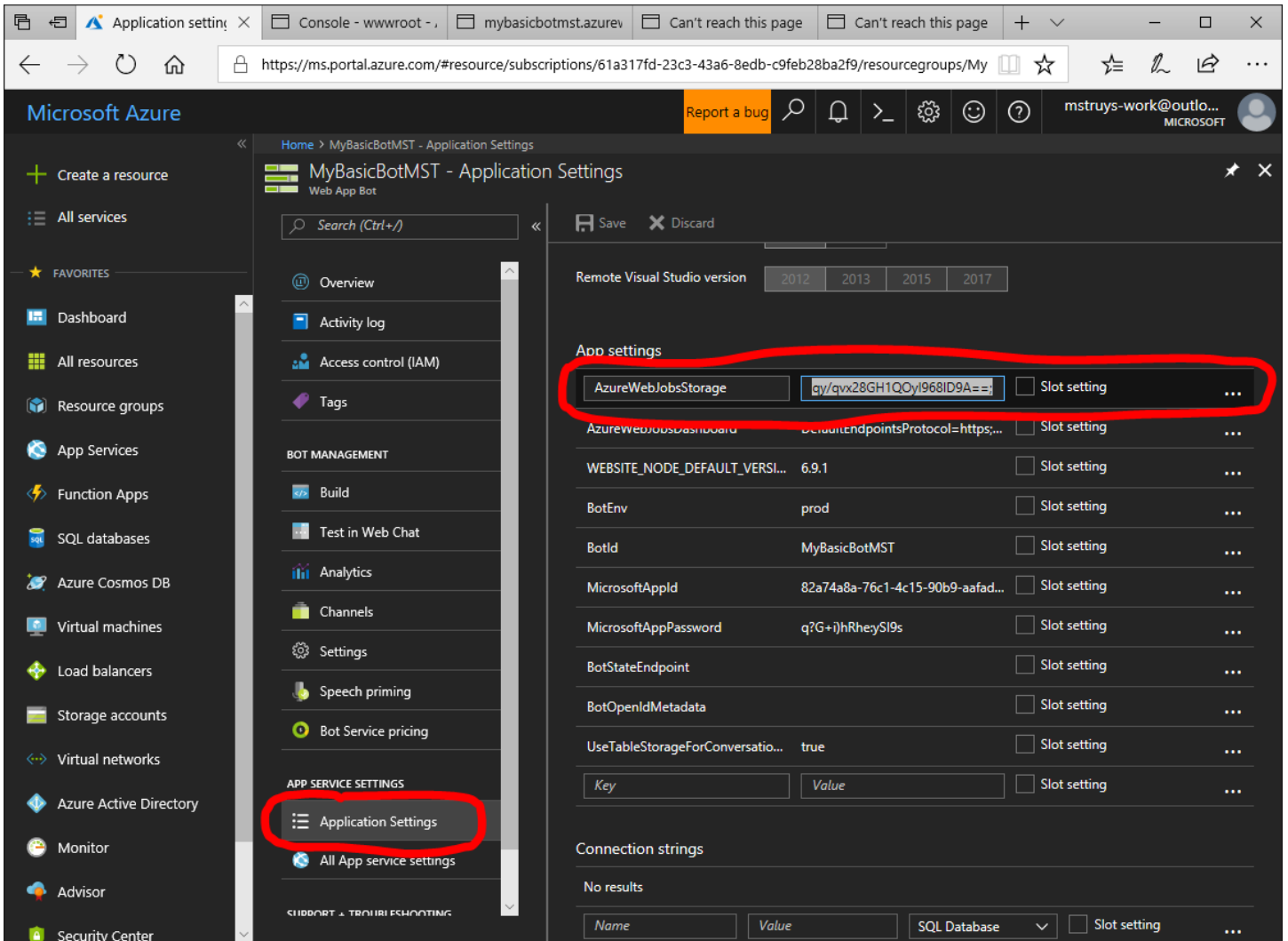


- Step 28. To solve this problem, you need to stop debugging by clicking on the red square (stop button) in the Visual Studio debug menu or simultaneously click **Shift – F5**.
- Step 29. In **Solution Explorer**, click the **Web.config** file to open it in a text editor.
- Step 30. Under line 14, insert the following line:
`<add key="AzureWebJobsStorage" value="" />`



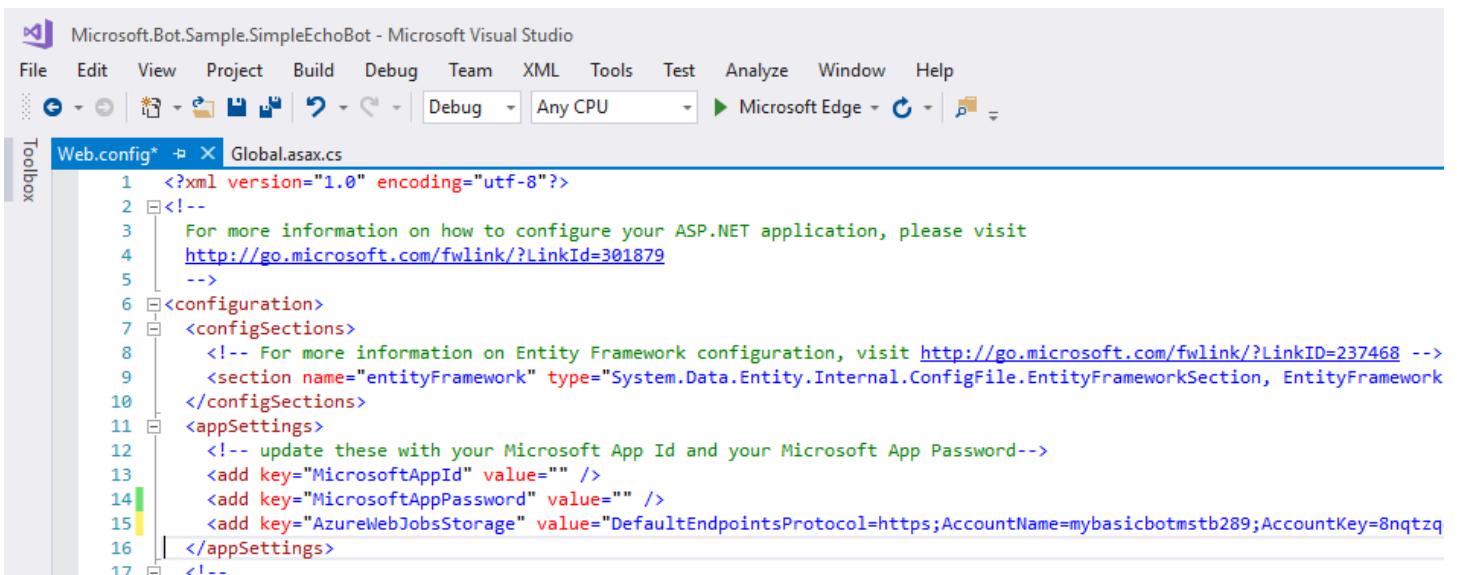
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!--
3 For more information on how to configure your ASP.NET application, please visit
4 http://go.microsoft.com/fwlink/?LinkId=301879
5 -->
6 <configuration>
7 <configSections>
8 <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=236758 -->
9 <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection" />
10 </configSections>
11 <appSettings>
12 <!-- update these with your Microsoft App Id and your Microsoft App Password-->
13 <add key="MicrosoftAppId" value="" />
14 <add key="MicrosoftAppPassword" value="" />
15 <add key="AzureWebJobsStorage" value="" />
16 </appSettings>
17 <!--
18 For a description of web.config changes see http://go.microsoft.com/fwlink/?LinkId=235367.
19 -->
```

Step 31. You will also need to provide a value for AzureWebJobsStorage, that you can find under **Application Settings** in the bot blade inside your Azure portal. Copy the value of AzureWebJobsStorage. Make sure you have selected the entire string.

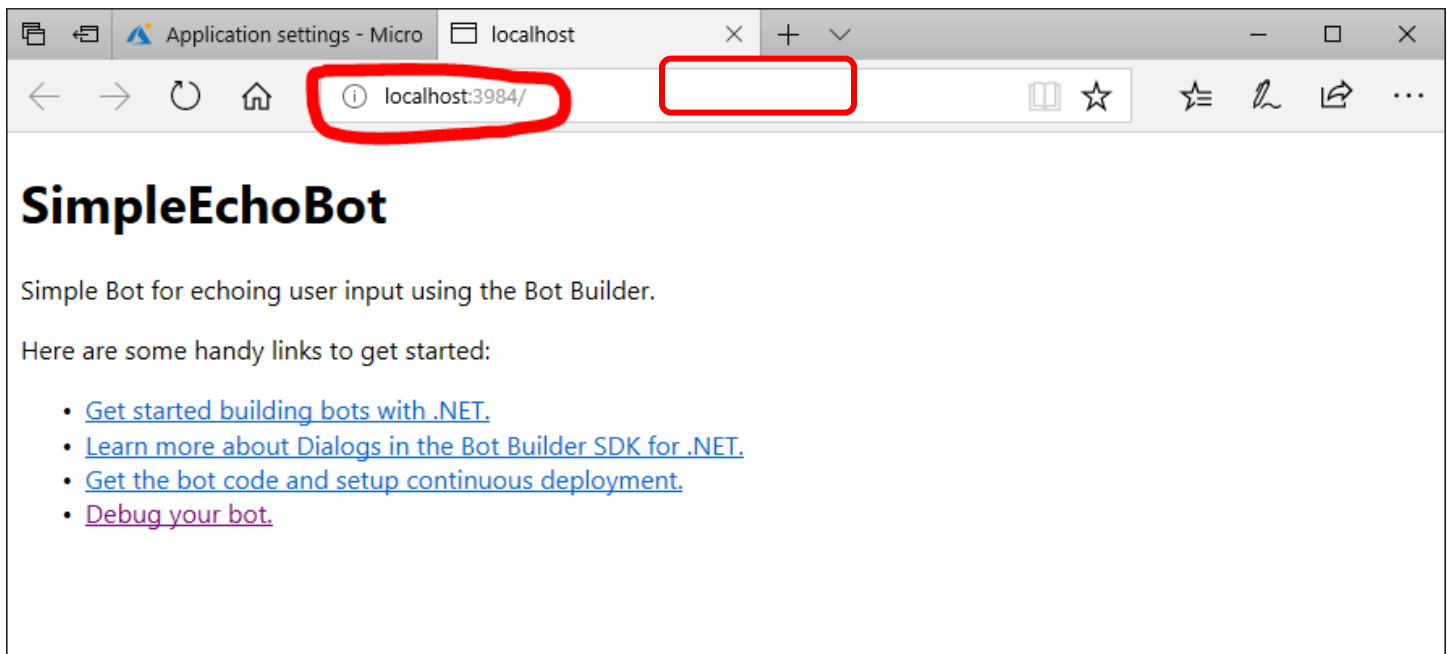


Step 32. Paste the copied AzureWebJobsStorage value into the **value** field of the **Web.config** file in Visual Studio (on Line 15).

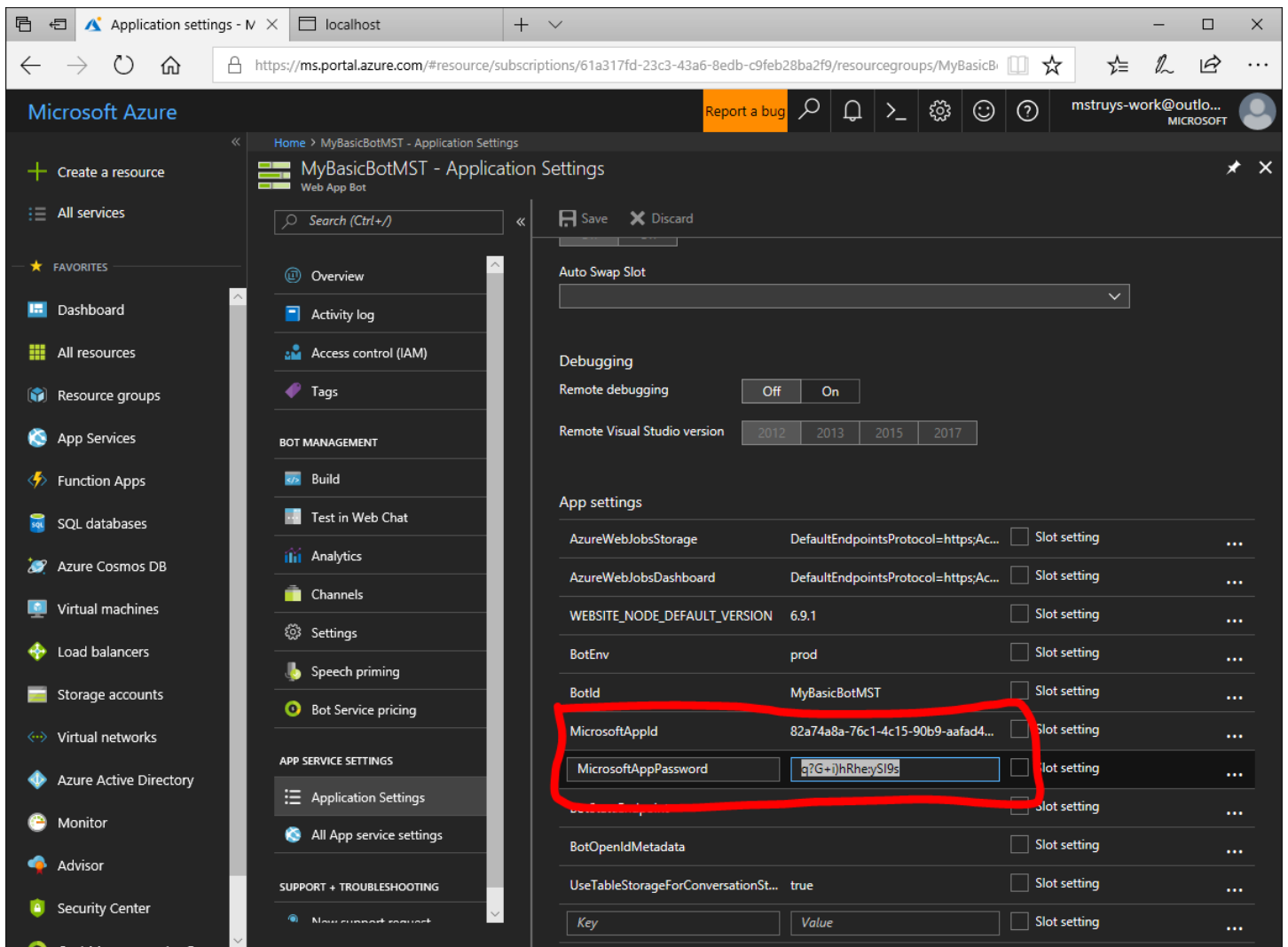
Step 33.



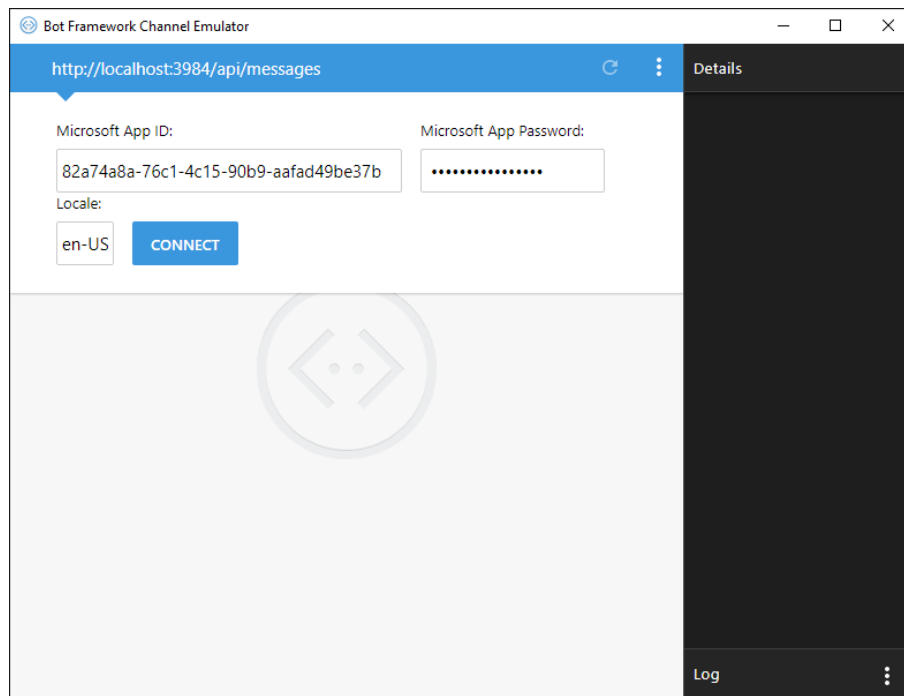
- Step 34. Save the Web.config file and start your bot from inside Visual Studio 2017 by clicking on the start button (you will automatically deploy the bot locally and a welcome page is shown in the browser if everything works correctly).
- Step 35. NOTE: If you get different run time errors than described in Step 27, most likely you are using an outdated version of IISExpress 10.0. The easiest way to fix this is by reinstalling IISExpress.
- Step 36. After all errors are resolved, your bot should start and open a page in your browser. Use the endpoint that you will see in the address bar of your browser to connect to the Bot Framework Emulator.



- Step 37. Launch the Bot Framework Emulator and enter your bot's endpoint into the emulator's address bar (in this example: <http://localhost:3984/api/messages>).
- Step 38. Get your bot's App Service credentials. You can find the App Service credentials under **Application Settings** in the bot blade inside your Azure portal. Copy the values for App as shown in the following figure. NOTE: Since you need those values later again, it is best to store them temporarily in an empty notepad file and copy them from there.



Step 39. Copy the values for Microsoft App ID and Microsoft App Password in the Bot Framework Channel Emulator



Step 40. Stop debugging your bot and add the same Microsoft App ID and Microsoft App Password in the **Web.config** file on lines 13 and 14.

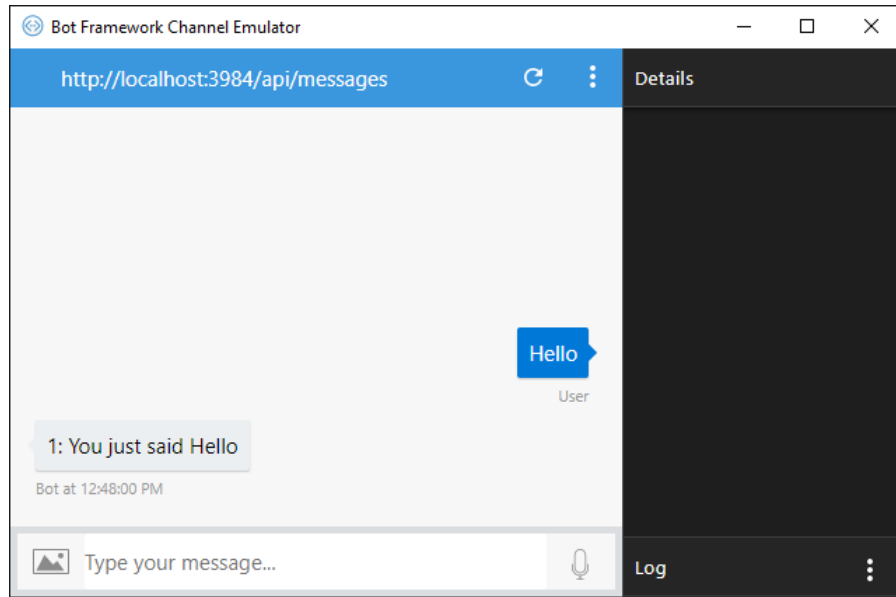
```

1 <?xml version="1.0" encoding="utf-8"?>
2 <!--
3 For more information on how to configure your ASP.NET application, please visit
4 http://go.microsoft.com/fwlink/?LinkId=301879
5 -->
6 <configuration>
7 <configSections>
8 <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkId=301879 -->
9 <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.Entity
10 </configSections>
11 <appSettings>
12 <!-- update these with your Microsoft App Id and your Microsoft App Password-->
13 <add key="MicrosoftAppId" value="82a74a8a-76c1-4c15-90b9-aafad49be37b" />
14 <add key="MicrosoftAppPassword" value="q?G+i)hRhe:ySI9s" />
15 <add key="AzureWebJobsStorage" value="DefaultEndpointsProtocol=https;AccountName=my
16 </appSettings>
17 <!--

```

Step 41. Start debugging again inside Visual Studio 2017 and wait until the bot has started (appears in the browser).

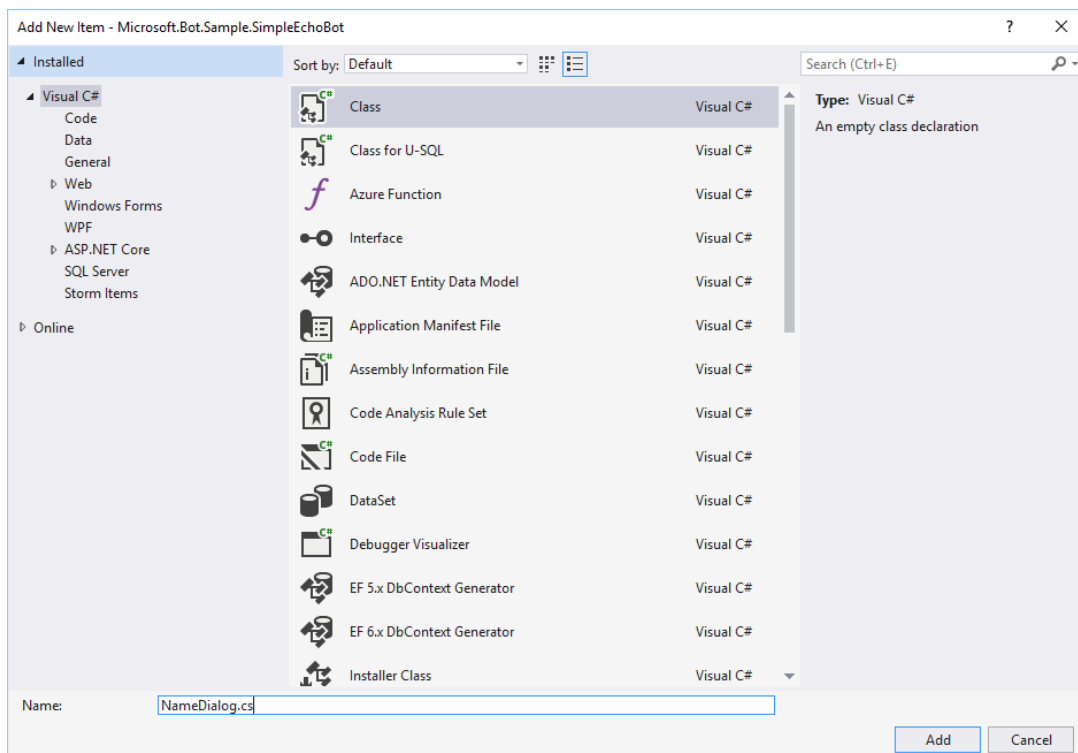
Step 42. In the Bot Framework Emulator, now click on the Connect button to debug your bot locally.



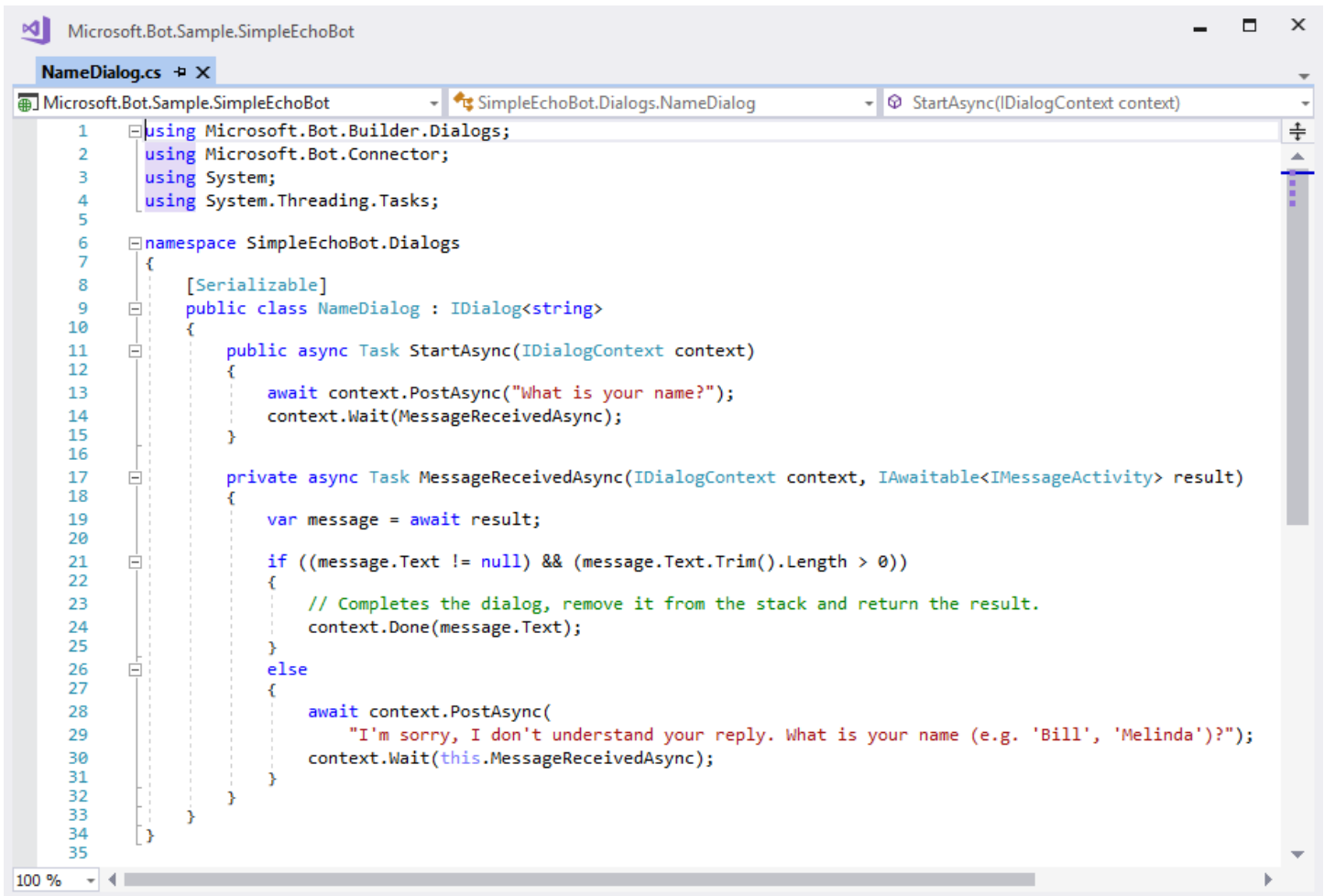
Add a dialog based conversation to your bot

Now we will add a dialog style conversation where the bot will ask for your (first) name and age.

- Step 43. Stop debugging your bot inside Visual Studio 2017.
- Step 44. Add a new class to your solution by navigating to the **Dialog** folder in your solution, **right-clicking** in that folder and select **Add New Item – Class**.
- Step 45. Call your new class **NameDialog.cs**.



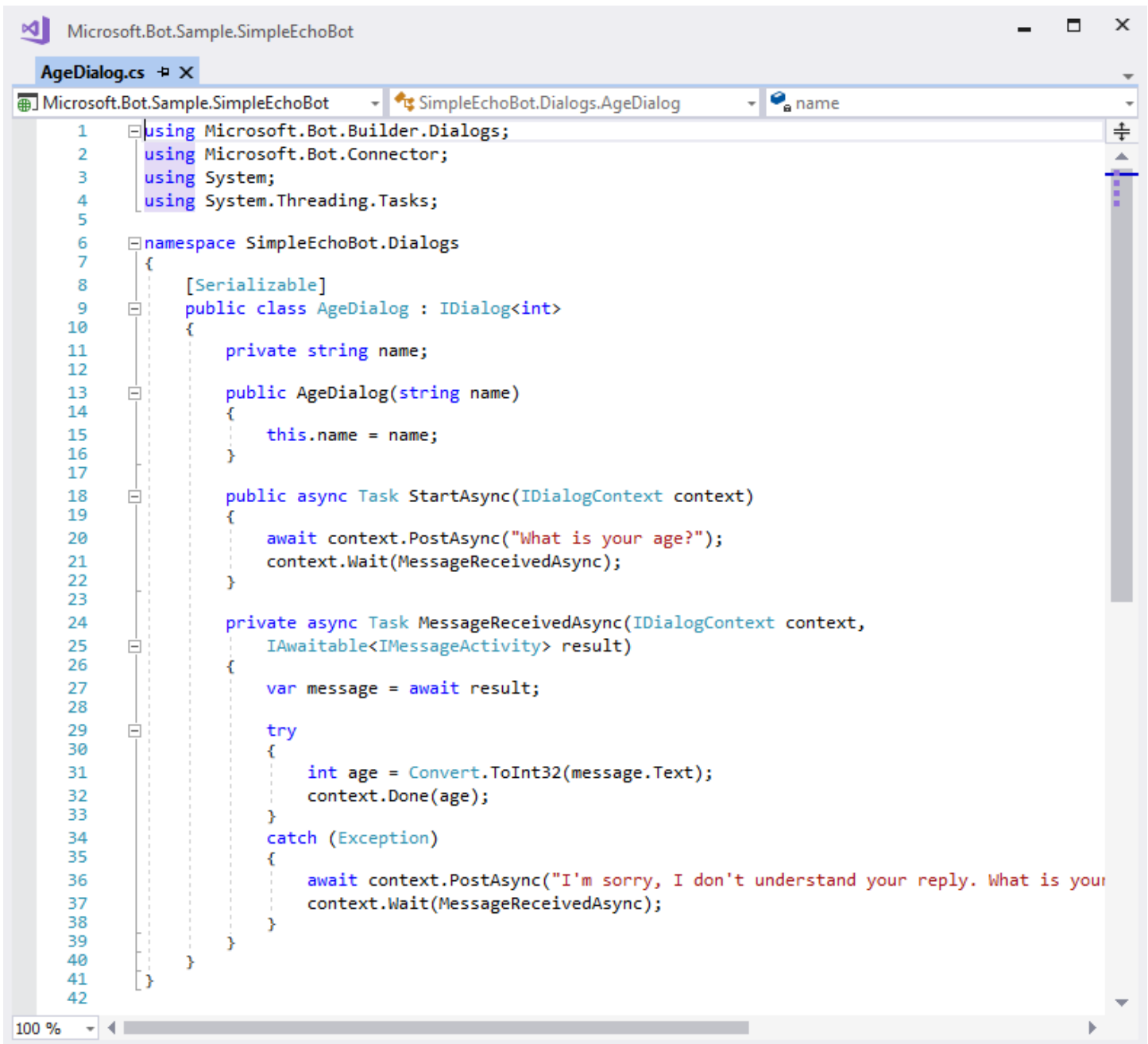
- Step 46. Replace the contents of the newly created **NameDialog** class with the following code. To save time, you can copy and paste the code for this class from Appendix 1 of this document. Just double click on the source code in [Appendix 1](#), copy all the contents and replace the contents of NameDialog.cs with the copied source code.



```
1 using Microsoft.Bot.Builder.Dialogs;
2 using Microsoft.Bot.Connector;
3 using System;
4 using System.Threading.Tasks;
5
6 namespace SimpleEchoBot.Dialogs
7 {
8     [Serializable]
9     public class NameDialog : IDialog<string>
10     {
11         public async Task StartAsync(IDialogContext context)
12         {
13             await context.PostAsync("What is your name?");
14             context.Wait(MessageReceivedAsync);
15         }
16
17         private async Task MessageReceivedAsync(IDialogContext context, IAwaitable<IMessageActivity> result)
18         {
19             var message = await result;
20
21             if ((message.Text != null) && (message.Text.Trim().Length > 0))
22             {
23                 // Completes the dialog, remove it from the stack and return the result.
24                 context.Done(message.Text);
25             }
26             else
27             {
28                 await context.PostAsync(
29                     "I'm sorry, I don't understand your reply. What is your name (e.g. 'Bill', 'Melinda')?");
30                 context.Wait(this.MessageReceivedAsync);
31             }
32         }
33     }
34 }
35
```

- Step 47. Add another new class to your solution by navigating to the Dialog folder in your solution, right-clicking in that folder and select Add New Item – Class.
- Step 48. Call your new class AgeDialog.cs.

Step 49. Replace the contents of the newly created AgeDialog class with the following code. To save time, you can copy and paste the code for this class from [Appendix 2](#).

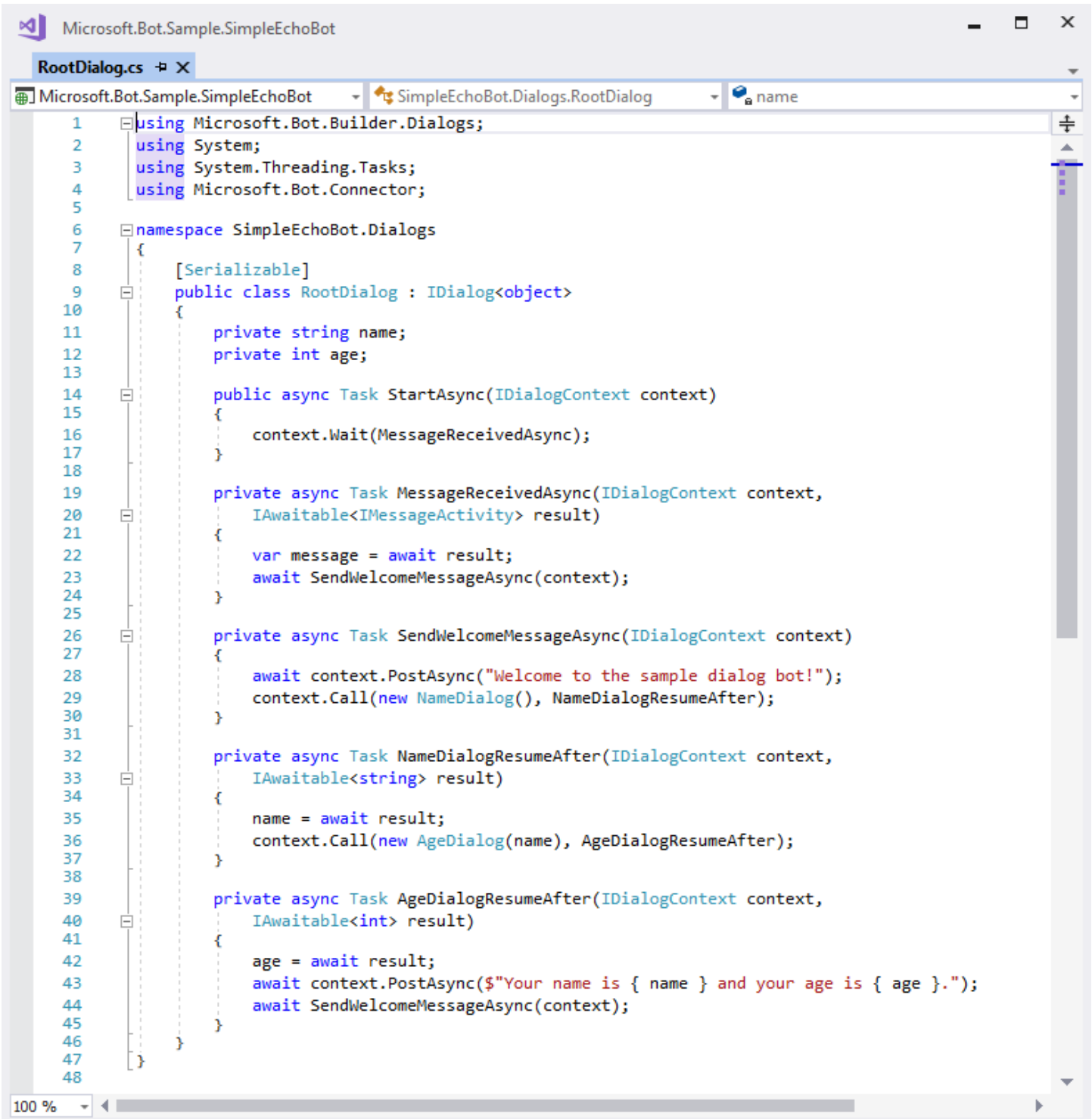


```
1 using Microsoft.Bot.Builder.Dialogs;
2 using Microsoft.Bot.Connector;
3 using System;
4 using System.Threading.Tasks;
5
6 namespace SimpleEchoBot.Dialogs
7 {
8     [Serializable]
9     public class AgeDialog : IDialog<int>
10     {
11         private string name;
12
13         public AgeDialog(string name)
14         {
15             this.name = name;
16         }
17
18         public async Task StartAsync(IDialogContext context)
19         {
20             await context.PostAsync("What is your age?");
21             context.Wait(MessageReceivedAsync);
22         }
23
24         private async Task MessageReceivedAsync(IDialogContext context,
25             IAwaitable<IMessageActivity> result)
26         {
27             var message = await result;
28
29             try
30             {
31                 int age = Convert.ToInt32(message.Text);
32                 context.Done(age);
33             }
34             catch (Exception)
35             {
36                 await context.PostAsync("I'm sorry, I don't understand your reply. What is your");
37                 context.Wait(MessageReceivedAsync);
38             }
39         }
40     }
41 }
42
```

Step 50. Add another new class to your solution by navigating to the **Dialog** folder in your solution, **right-clicking** in that folder and select **Add New Item – Class**.

Step 51. Call your new class **RootDialog.cs**. This class will determine the navigation between the different dialogs in your bot/

Step 52. Replace the contents of the newly created RootDialog class with the following code. To save time, you can copy and paste the code for this class from [Appendix 3](#).



```
1 using Microsoft.Bot.Builder.Dialogs;
2 using System;
3 using System.Threading.Tasks;
4 using Microsoft.Bot.Connector;
5
6 namespace SimpleEchoBot.Dialogs
7 {
8     [Serializable]
9     public class RootDialog : IDialog<object>
10     {
11         private string name;
12         private int age;
13
14         public async Task StartAsync(IDialogContext context)
15         {
16             context.Wait(MessageReceivedAsync);
17         }
18
19         private async Task MessageReceivedAsync(IDialogContext context,
20             IAwaitable<IMessageActivity> result)
21         {
22             var message = await result;
23             await SendWelcomeMessageAsync(context);
24         }
25
26         private async Task SendWelcomeMessageAsync(IDialogContext context)
27         {
28             await context.PostAsync("Welcome to the sample dialog bot!");
29             context.Call(new NameDialog(), NameDialogResumeAfter);
30         }
31
32         private async Task NameDialogResumeAfter(IDialogContext context,
33             IAwaitable<string> result)
34         {
35             name = await result;
36             context.Call(new AgeDialog(name), AgeDialogResumeAfter);
37         }
38
39         private async Task AgeDialogResumeAfter(IDialogContext context,
40             IAwaitable<int> result)
41         {
42             age = await result;
43             await context.PostAsync($"Your name is { name } and your age is { age }.");
44             await SendWelcomeMessageAsync(context);
45         }
46     }
47 }
48
```

Step 53. Finally, open the existing class MessageController.cs that you can find in the folder **Controllers** of your Visual Studio solution.

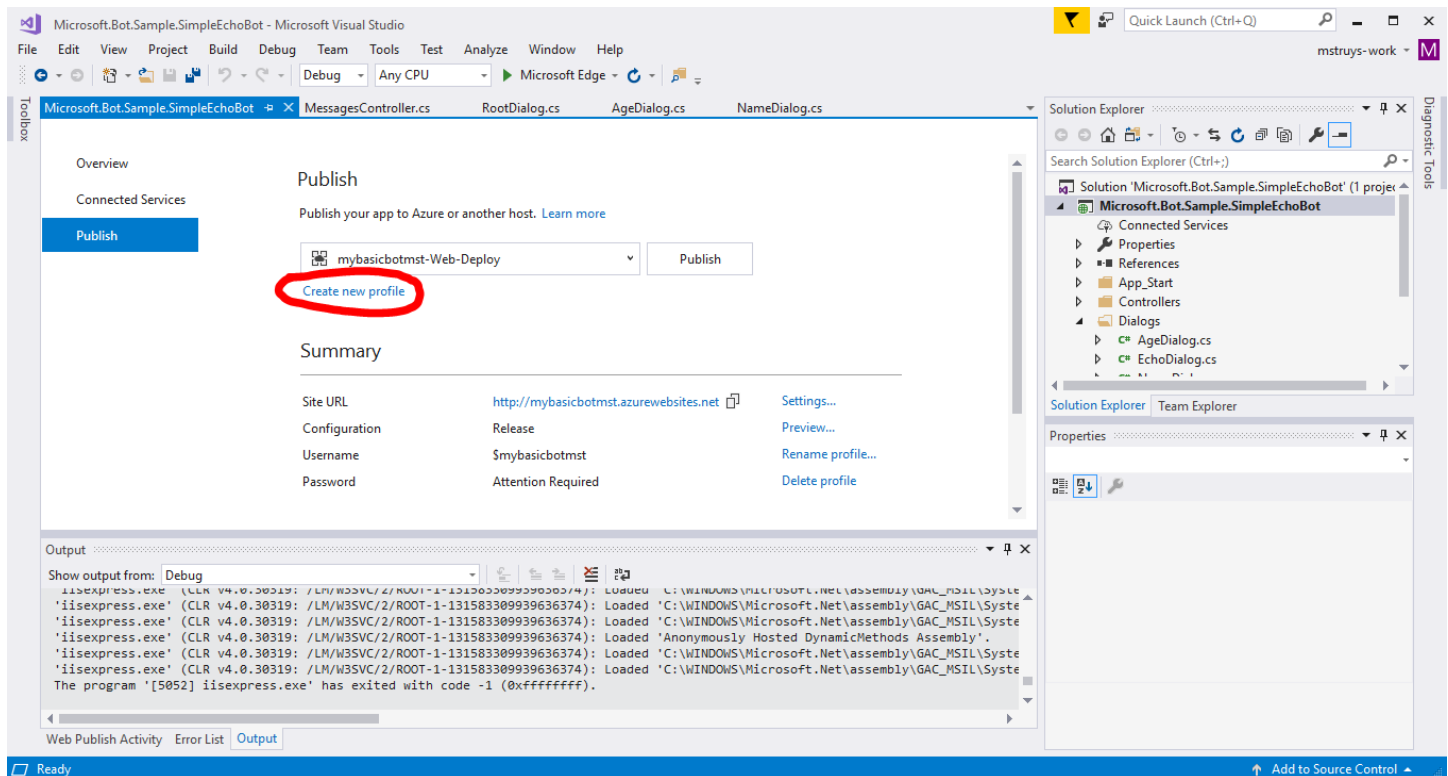
Step 54. Replace the following line of code (should be at line 25):
`await Conversation.SendAsync(activity, () => new EchoDialog());`
with the following
`await Conversation.SendAsync(activity, () => new RootDialog());`

- Step 55. Also make sure to add a **using SimpleEchoBot.Dialogs** statement at the beginning of the file **MessageController.cs**.
- Step 56. Start debugging the bot again inside Visual Studio 2017 and use the Bot Framework Channel Emulator to debug your bot locally. You should now be able to have a simple conversation with your bot. It will ask you for your name and age. After entering, it will show you a message and starts all over from the beginning.

Publish your modified bot back to Azure

To set up publishing from Visual Studio using the `.PublishSettings` file, perform the following steps:

- Step 57. In the Solution Explorer pane, right-click your project, and click **Publish...** The Publish window opens.
- Step 58. In the Publish window, click **Create new profile**, click **Import profile**, and click **Publish**.



- Step 59. Navigate to your project folder, navigate to the **PostDeployScripts** folder, select the file that ends in `.PublishSettings`, and click **Open**.

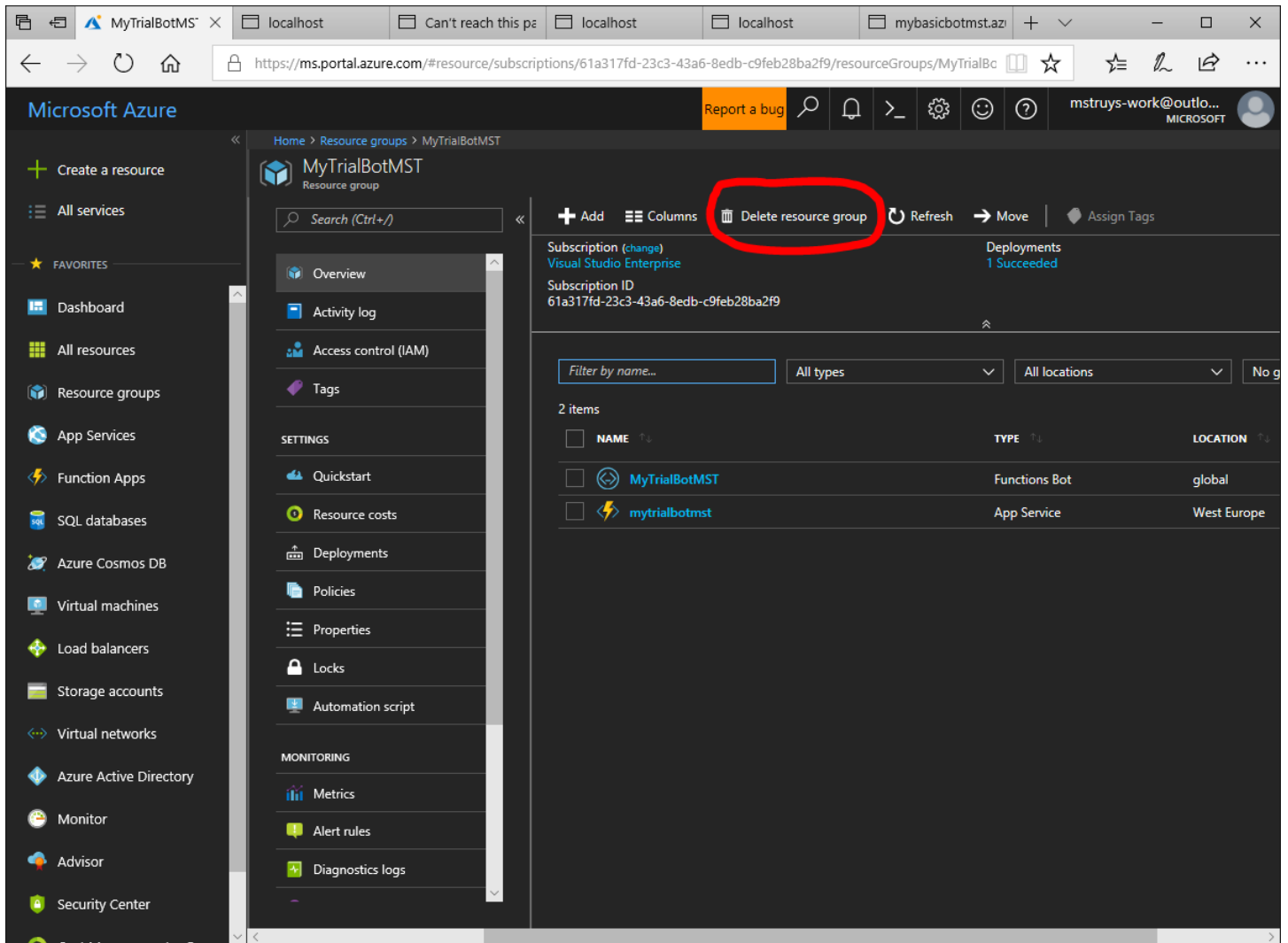
You have now configured publishing for this project. Your bot will be build one more time and deployed to Azure. It also will be started automatically.

Now you can go back to your Azure portal. Now that your bot is running in the cloud, try it out by typing a few messages into the built-in chat control that's located to the right of the code editor in Azure. You can now have the same conversation with your bot, but this time it is running inside Azure.

Finally, you can remove the entire bot (and thus no longer use Azure resources) by deleting the entire Resource Group you have created for this lab.

- Step 60. Go to the **Resource Groups** blade in the Azure portal and select the Resource Group that you have created in Step 6.

Step 61. Click Delete resource group to delete the bot and the attached storage.



Step 62. Type the name of the resource group in the edit box and click **Delete** to delete all resources associated with your bot.

This completes the lab on bots. Of course, this was just an initial exploration of the Microsoft Bot Framework. We encourage you to continue experimenting with the Microsoft Bot Framework and to look at integrating Microsoft Cognitive Services into your bot.

Appendix 1 – Source code for NameDialog class

```
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Connector;
using System;
using System.Threading.Tasks;

namespace SimpleEchoBot.Dialogs
{
    [Serializable]
    public class NameDialog : IDialog<string>
    {
        public async Task StartAsync(IDialogContext context)
        {
            await context.PostAsync("What is your name?");
            context.Wait(MessageReceivedAsync);
        }

        private async Task MessageReceivedAsync(IDialogContext context,
            IAwaitable<IMessageActivity> result)
        {
            var message = await result;

            if ((message.Text != null) && (message.Text.Trim().Length > 0))
            {
                // Completes the dialog, remove it from the stack and return the result.
                context.Done(message.Text);
            }
            else
            {
                await context.PostAsync(
                    "I'm sorry, I don't understand your reply. What is your name (e.g. 'Bill', 'Melinda')?");
                context.Wait(this.MessageReceivedAsync);
            }
        }
    }
}
```


Appendix 2 – Source code for AgeDialog class

```
using Microsoft.Bot.Builder.Dialogs;
using Microsoft.Bot.Connector;
using System;
using System.Threading.Tasks;

namespace SimpleEchoBot.Dialogs
{
    [Serializable]
    public class AgeDialog : IDialog<int>
    {
        private string name;

        public AgeDialog(string name)
        {
            this.name = name;
        }

        public async Task StartAsync(IDialogContext context)
        {
            await context.PostAsync("What is your age?");
            context.Wait(MessageReceivedAsync);
        }

        private async Task MessageReceivedAsync(IDialogContext context,
            IAwaitable<IMessageActivity> result)
        {
            var message = await result;

            try
            {
                int age = Convert.ToInt32(message.Text);
                context.Done(age);
            }
            catch (Exception)
            {
                await context.PostAsync("I'm sorry, I don't understand your reply. What  
is your age (e.g. '42')?");
                context.Wait(MessageReceivedAsync);
            }
        }
    }
}
```

Appendix 3 – Source code for the RootDialog class

```
using Microsoft.Bot.Builder.Dialogs;
using System;
using System.Threading.Tasks;
using Microsoft.Bot.Connector;

namespace SimpleEchoBot.Dialogs
{
    [Serializable]
    public class RootDialog : IDialog<object>
    {
        private string name;
        private int age;

        public async Task StartAsync(IDialogContext context)
        {
            context.Wait(MessageReceivedAsync);
        }

        private async Task MessageReceivedAsync(IDialogContext context,
            IAwaitable<IMessageActivity> result)
        {
            var message = await result;
            await SendWelcomeMessageAsync(context);
        }

        private async Task SendWelcomeMessageAsync(IDialogContext context)
        {
            await context.PostAsync("Welcome to the sample dialog bot!");
            context.Call(new NameDialog(), NameDialogResumeAfter);
        }

        private async Task NameDialogResumeAfter(IDialogContext context,
            IAwaitable<string> result)
        {
            name = await result;
            context.Call(new AgeDialog(name), AgeDialogResumeAfter);
        }

        private async Task AgeDialogResumeAfter(IDialogContext context,
            IAwaitable<int> result)
        {
            age = await result;
            await context.PostAsync($"Your name is { name } and your age is { age }.");
            await SendWelcomeMessageAsync(context);
        }
    }
}
```