AI Live 2018

Build a Microsoft Bot Framework bot with the Bot Builder SDK v4

Using Visual Studio 2017, and ASP.NET Core

Estimated time to complete: **30 Minutes**

This walkthrough will show you how you can create, build, and test a simple bot using the Azure Portal first, followed by Visual Studio 2017 using C#, ASP.NET, and the Bot Builder v4 SDK.

Contents

Create Your First Bot

Create Your Bot via the Azure Portal

Create Your Bot in Visual Studio 2017

Test Your Bot Locally Using the Emulator

Update Your Bot and Debug

Create Your First Bot

You'll create your first bot using the Azure Bot Service via the Azure Portal, and then you'll use Visual Studio 2017 to build a bot. When using Visual Studio 2017, you'll access the Bot Builder v4 SDK runtime components via NuGet.org.

Create Your Bot via the Azure Portal

You'll use the Azure Portal to create your first bot.

1. Launch a modern browser like Microsoft Edge, Google Chrome, etc. But not Internet Explorer.
2. Access https://portal.azure.com
3. Log in using an account that has full control of an active Azure Subscription.
4. Click the **Create a resource** link in the upper left-hand portion of the portal home page.
5. In the list of items on the **Azure Marketplace** blade, click **AI + Machine Learning**.
6. Then, select **Web App Bot**.
   The Azure Portal will open a Web App Bot blade that you need to fill in with the requested information to create your bot.
7. Use the data in the following table to configure most of the information:

Item	Value	Notes
Bot name	Your bot's display name. Consider **az360botabc99** where **abc** are your initials and **99** is a couple of numbers like the year you graduated, etc.	The display name for your bot that appears in channels and directories. Your bot's name can only have the following characters: **-**, **a-z**, **A-Z**, **0-9**, and **_**
Subscription	Your subscription	If necessary, select the Azure subscription you want to use if you have more than one.
Resource Group	Click Create New and provide a name like **az360bot**.	Create a new resource group. It will be easier to clean up when you're done.

Location	**East US** or **East US 2**	Select the geographic location for your resource group. Your location choice can be any location listed, though it's often best to choose a location closest to your customers. The location cannot be changed once the bot is created. For the lab at *Live! 360*, consider **East US** or **East US 2**.
Pricing tier	**F0**	Select a pricing tier. You may update the pricing tier at any time.
App name	A unique name. Consider **az360botabc99** where **abc** are your initials and **99** is a couple of numbers like the year you graduated, etc.	The unique URL name of the bot, no more than 35 characters in length. For example, if you name your bot *live360botabc99*, then your bot's URL will be http://live360botabc99.azurewebsites.net. The name must use alphanumeric and underscore characters only. The App name cannot be changed once the bot is created.
Bot template	**Echo Bot**	Choose **SDK v4** and **C#**
App service plan/Location	Select your Bot App Service plan created earlier.	
Azure Storage	A new Azure storage account	Create a new data storage account.
Application Insights	**Off**	You'll do this later.
Microsoft App ID and password	Auto create App ID and password	You'll see more of this later.

8. Once you've filled out the blade and checked your entries, click **Create**.
   Wait for Azure to create your bot. Use the **Notifications** pane to monitor the status.

9. When your bot is ready, click the **Go to resource** button or navigate via the **Resource Groups** blade.
10. Feel free to explore a bit and when you're ready, click **Test in Web Chat** under the **Bot Management** heading.
11. Once the page loads, type a message like **Hello, world!**
    You will receive a debug status message and then a **Turn 1** reply to your message.
12. Send a few more messages and notice the counter goes up.
13. Minimize your browser and move on to the next section.

Create Your Bot in Visual Studio 2017

In the first section you built a simple bot via the Azure Portal. Now you'll start from Visual Studio.

1. Start **Visual Studio 2017**.
2. Choose **File | New | Project**.
3. In the *New Project* dialog, under **Visual C#**, choose **Bot Framework**.
4. Expand the **Bot Framework** node so you can see and select **Multi-project**.
5. From the middle pane select **Bot Builder Echo Bot V4**.
6. Provide a **Name** like **Vs360bot**.
7. Set the **Location** to **C:\botlab.**
8. Ensure **Create directory for solution** is checked.
9. Ensure **Add to Source Control** is **NOT** checked.
10. Click **OK**.
11. Select **Build | Rebuild Solution**.
12. In the *Solution Explorer* window, double-click on the **Properties** node.
13. Change the **Target Framework** to **.NET Core 2.1**.
14. Save your work via **File | Save All**.
15. Select **Build | Rebuild Solution**.
16. In the *Solution Explorer*, right-click on the **Solution** node and choose **Manage NuGet Packages for Solution.**
17. In the *NuGet - Solution* window, if **Updates** is showing a number next to it, select the **Updates** link.
18. Click **Select all packages** followed by the **Update** button and complete the NuGet package update process.
19. Save your work via **File | Save All**.
20. Select **Windows | Close All Documents** to tidy up Visual Studio.
21. Leave Visual Studio open and continue to the next section.

Test Your Bot Locally Using the Emulator

Now that you've created and built your bot, you'll test it.

1. In Visual Studio, select **Debug | Start Debugging**. Visual Studio starts your default browser and points to localhost on a custom port for your bot.
2. Minimize your browser and Visual Studio 2017.
3. Start the **Bot Framework Emulator (V4)**.
4. Click the **Open Bot** button on the *Welcome* page.
5. Navigate to into **C:\botlab\** and drill down until you find the **BotConfiguration.bot** file.
6. Select it and click the **Choose file** button in the dialog.
7. When you do, you should see a bunch of status messages in the LOG window on right side of the emulator.
8. In the **Type your message field**, enter a message like **Hello, World Local!** and press Enter.
   You should see what you typed followed by the bot's reply like earlier up in Azure.
9. Select your message first and notice the JSON message in the INSPECTOR – JSON window.
10. Select the reply and review its data.
11. When ready, switch to your web browser and close it (or at least the tab used by your bot).
12. In Visual Studio, select **Debug | Stop Debugging**.
13. Leave things open and continue to the next section.

Update Your Bot and Debug

You'll now make a change to your bot and debug it running

1. In the *Solution Explorer* window, open the **CounterState.cs** file.
2. Add two new members as follows:

```
public bool GreetingDone { get; set; } = false;
public bool HadEnough { get; set; } = false;
```

3. Save your changes.
4. Next from the *Solution Explorer* window, open the **EchoWithCounterBot.cs** file.
5. Find an existing block of code around line 64 as follows:

```
if (turnContext.Activity.Type == ActivityTypes.Message)
{
    // Get the conversation state from the turn context.
    var state = await _accessors.CounterState.GetAsync(turnContext, () =>
        new CounterState());

    // Bump the turn count for this conversation.
    state.TurnCount++;
```

6. Replace the rest of the code *within* the if block with the following block of code after **state.TurnCount++;**.

```
var responseMessage =
   $"Turn {state.TurnCount}: You sent '{turnContext.Activity.Text}'\n";

if (!state.GreetingDone)
{
    responseMessage = "Hello, welcome to the Echo Bot!";
    state.GreetingDone = true;
    state.HadEnough = false;
}
else if (!state.HadEnough && state.GreetingDone)
{
    if (turnContext.Activity.Text.ToLower().Contains("goodbye"))
    {
        responseMessage = "Sorry to see you go. Bye!";
        state.TurnCount = 0;
        state.GreetingDone = false;
        state.HadEnough = true;
    }
}

await _accessors.CounterState.SetAsync(turnContext, state);

await _accessors.ConversationState.SaveChangesAsync(turnContext);

await turnContext.SendActivityAsync(responseMessage);
```

7. Save your changes.
8. Select **Debug | Start Debugging**.
9. Once the browser shows your bot's home page, switch back to the emulator click the **Restart conversation** button.
10. Type a greeting message. What do you see?
11. Type some other message. What do you see?

12. Type **Goodbye**. What do you see?
13. Switch back to Visual Studio, and place a breakpoint on the **if** statement.
14. Switch back to the emulator and repeat the process (do NOT restart the conversation), examining the state variables and the flow as you go.
15. When ready, switch to your web browser and close it (or at least the tab used by your bot).
16. In Visual Studio, select **Debug | Stop Debugging**.
17. Take a quick break and move on to the next lab.