

Laboratorio

Creando Bots con el Microsoft Bot Framework

Versión: 1.0.0

Enero de 2018



[Miguel Muñoz Serafín](#)

@msmdotnet



CONTENIDO

Introducción	3
Ejercicio 1: Creando la aplicación bot con Bot Builder SDK for .NET	6
Tarea 1. Crear un proyecto ASP.NET Web API.	6
Tarea 2. Agregar el controlador de la aplicación.	9
Ejercicio 2: Implementando seguridad en el bot	11
Tarea 1. Agregar las credenciales de autenticación del bot en el archivo web.config.	11
Tarea 2. Agregar el atributo BotAuthentication al controlador.....	12
Ejercicio 3: Creando diálogos con el Bot Builder SDK for .NET	14
Tarea 1. Definir un Diálogo.....	14
Tarea 2. Crear el punto de entrada del diálogo.	18
Ejercicio 4: Probando el funcionamiento del bot.....	23
Tarea 1. Descargar e instalar el Bot Framework Emulator.	23
Tarea 2. Iniciar el bot.....	23
Tarea 3. Iniciar el emulador y conectar al bot.	24
Tarea 4. Probar el bot.....	25
Ejercicio 5: Creando un bot con Bot Service (opcional)	27
Tarea 1. Crear un nuevo servicio bot.	27
Tarea 2. Probar el bot en el Web Chat de Azure.....	34
Tarea 3. Probar el bot con el emulador.	36
Tarea 4. Editar el bot con el editor en línea.	41

Introducción

¿Qué es un Bot? Bot es una abreviación de Robot. En términos de software, podemos decir que un Bot es un programa de computadora que es capaz de llevar a cabo tareas concretas e imitar el comportamiento humano. Tradicionalmente un bot efectúa sus tareas a través de Internet.

¿Qué es un Chatbot? En términos de software, podemos decir que un Chatbot es un programa de computadora que imita una conversación con personas y que puede utilizar inteligencia artificial. Un Chatbot puede transformar la manera en que un usuario interactúa con Internet, convirtiendo una serie de tareas en una simple conversación.

En una conversación, los bots pueden comunicarse mediante texto, voz o tarjetas enriquecidas conteniendo texto, imágenes y botones de acción. Un bot puede ser tan simple como una aplicación que busca un texto en una fuente de datos y envía una respuesta, o puede ser una aplicación compleja que utilice técnicas de inteligencia artificial con un complejo seguimiento del estado de la conversación y la integración con servicios de negocios existentes.

Los desarrolladores de aplicaciones web pueden pensar de un bot como un Servicio Web RESTful que intercambia información en formato JSON y que puede ser desarrollado en el lenguaje de su preferencia. Por ejemplo, los desarrolladores .NET podrían crear un bot utilizando ASP.NET Web API.

Microsoft Bot Framework

Para facilitar el desarrollo de bots, Microsoft lanzó Bot Framework, una plataforma en la cual podemos desarrollar un código único y desplegarlo en varias plataformas de conversación.

Microsoft Bot Framework es una plataforma que proporciona todos los elementos necesarios para construir y conectar bots inteligentes que interactúen de forma natural, en cualquier parte que se encuentren conversando los usuarios, desde texto/sms, Facebook Messenger, Skype, Slack, correo de Office 365 y otros servicios populares. A través del Bot Framework podemos utilizar recursos de Inteligencia Artificial, recursos de Machine Learning, realizar una comunicación con datos REST, reconocimiento del lenguaje natural, procesamiento de imágenes, etc.

Bot Builder

Uno de los componentes del *Bot Framework* es el **Bot Builder**. El *Bot Builder* es un Framework que proporciona SDKs, bibliotecas, ejemplos y herramientas para ayudarnos a construir y depurar bots. Los desarrolladores podemos crear bots utilizando los SDKs proporcionados por *Bot Builder*:

- **Bot Builder SDK for .NET** para desarrolladores de C# y
- **Bot Builder SDK for Node.js** para desarrolladores de JavaScript.

Bot Builder también incluye el **Bot Framework Emulator** para probar nuestros bots y el **Channel Inspector** para obtener una vista previa de la experiencia de usuario de nuestro bot en diferentes canales.

Bot Framework también proporciona **APIs REST** que nos permiten crear bots utilizando cualquier lenguaje de programación. En otras palabras, no necesitamos trabajar con C# o Node.js para poder crear Bots con el *Microsoft Bot Framework*.

Bot Framework Connector

Otro de los componentes del *Microsoft Bot Framework* es el **Bot Connector**. El *Bot Connector* es un servicio online que permite conectar nuestro bot con una serie de servicios como Skype, Slack, SMS, Facebook Messenger o Telegram entre otros a través de una interfaz de API REST implementada en el código del bot. A parte del envío y recepción de mensajes, el *Bot Connector* puede conectarse con los servicios de inteligencia e implementar otras funcionalidades como:

- Capacidad de almacenar el estado. El *Bot Connector* ofrece almacenamiento para guardar el estado de la conversación.
- Servicios de traducción. Opcionalmente, *Bot Connector* al conectarse con los servicios de inteligencia puede permitir la comunicación entre un bot y un usuario a pesar de que su idioma no sea el mismo.
- Telemetría. Se recoge información sobre el servicio, como por ejemplo el número de peticiones, los mensajes que han resultado en fallo etc.

Azure Bot Service

Bot Service es un producto de Microsoft Azure que nos permite desarrollar bots inteligentes con rapidez ya que combina el potencial de **Microsoft Bot Framework** y **Azure Functions**.

Bot Service nos proporciona todo lo necesario para construir, conectar, probar, implementar, supervisar y administrar bots. **Bot Service** proporciona los componentes principales para la creación de bots, incluido el **Bot Builder SDK** para desarrollar bots y el **Bot Framework** para conectar bots a canales.

Bot Service proporciona un entorno integrado diseñado específicamente para el desarrollo de bots. Podemos crear un bot, conectarlo, probarlo, implementarlo y administrarlo desde un navegador web sin necesidad de un editor o control de código fuente por separado. Para bots simples, es probable que no tengamos la necesidad de escribir una sola línea de código.

Bot Service acelera el desarrollo de bots poniendo a nuestra disposición varias plantillas de bots que podemos elegir cuando creamos un bot desde el portal de Azure. Después de haber creado el bot, es

posible modificarlo directamente en el navegador utilizando el editor de Azure o en un entorno de desarrollo integrado (IDE) como **Visual Studio** y **Visual Studio Code**.

En este laboratorio crearemos un Chatbot utilizando **Bot Builder SDK for .NET** y las herramientas que nos proporciona **Microsoft Bot Framework**. Opcionalmente, si cuentas con una suscripción a Microsoft Azure, tendrás la oportunidad de crear un bot utilizando **Azure Bot Service**.

Objetivos

Al finalizar este laboratorio, los participantes serán capaces de:

- Crear un bot utilizando el **Bot Builder SDK for .NET**.
- Implementar seguridad en el bot.
- Describir la forma de crear diálogos con el **Bot Builder SDK for .NET**.
- Probar el funcionamiento del bot utilizando el **Bot Framework Emulator**.
- Crear un bot con Azure Bot Service.

Requisitos

Para la realización de este laboratorio es necesario contar con lo siguiente:

- **Un equipo de desarrollo con Visual Studio 2017.** Puedes utilizar la versión gratuita [Visual Studio Community](#). El **Bot Builder SDK for .NET** actualmente soporta C#. Visual Studio for Mac no es soportado. Los pasos descritos en este laboratorio fueron diseñados con Visual Studio Enterprise 2017 sobre una máquina con Windows 10 Pro.
- **Una suscripción a Microsoft Azure (Opcional).** Puede adquirirse una suscripción de prueba en el siguiente enlace: <https://azure.microsoft.com/free/>

Tiempo estimado para completar este laboratorio: **5 horas**.

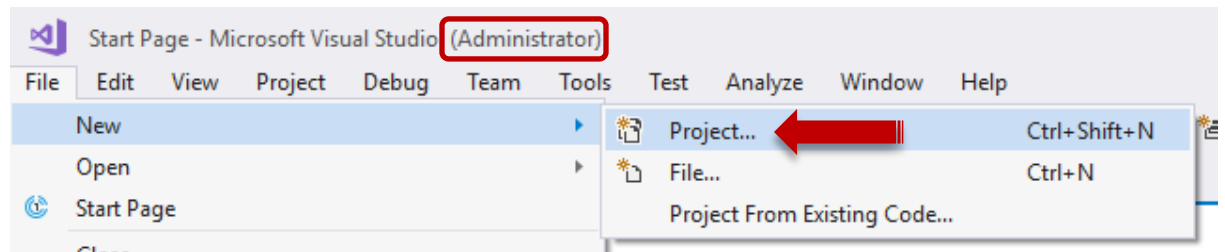
Ejercicio 1: Creando la aplicación bot con Bot Builder SDK for .NET

Una aplicación bot es simplemente una aplicación RESTful que intercambia información en formato JSON. El *Bot Builder SDK for .NET* proporciona la plantilla "[Bot Application.zip](#)" que nos facilita la creación de un bot, sin embargo, con el fin de explorar a detalle los componentes de una aplicación bot, en este ejercicio utilizaremos la plantilla tradicional para crear una aplicación ASP.NET Web API e instalaremos el **Bot Builder SDK for .NET** a través de un paquete NuGet.

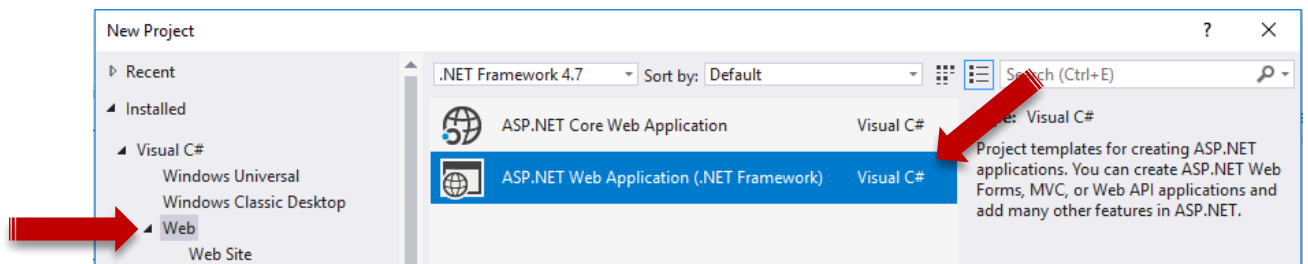
Tarea 1. Crear un proyecto ASP.NET Web API.

Realiza los siguientes pasos para crear una aplicación *ASP.NET Web API* que utilizaremos para el desarrollo de un chatbot.

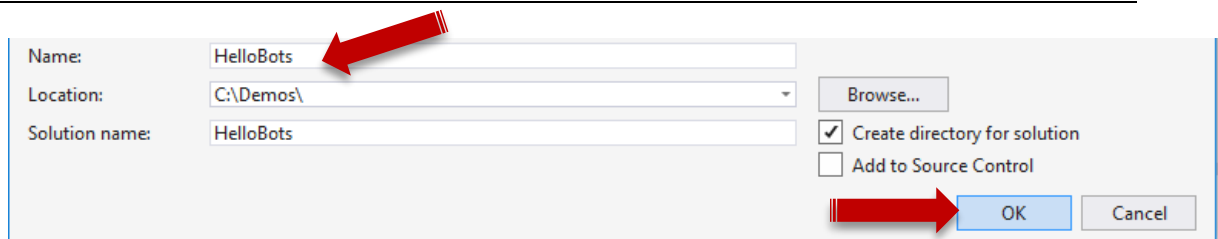
1. Abre Visual Studio bajo el contexto de Administrador.
2. En Visual Studio selecciona **File > New > Project...**



3. En la Ventana **New Project** selecciona la plantilla **ASP.NET Web Application (.NET Framework)**.

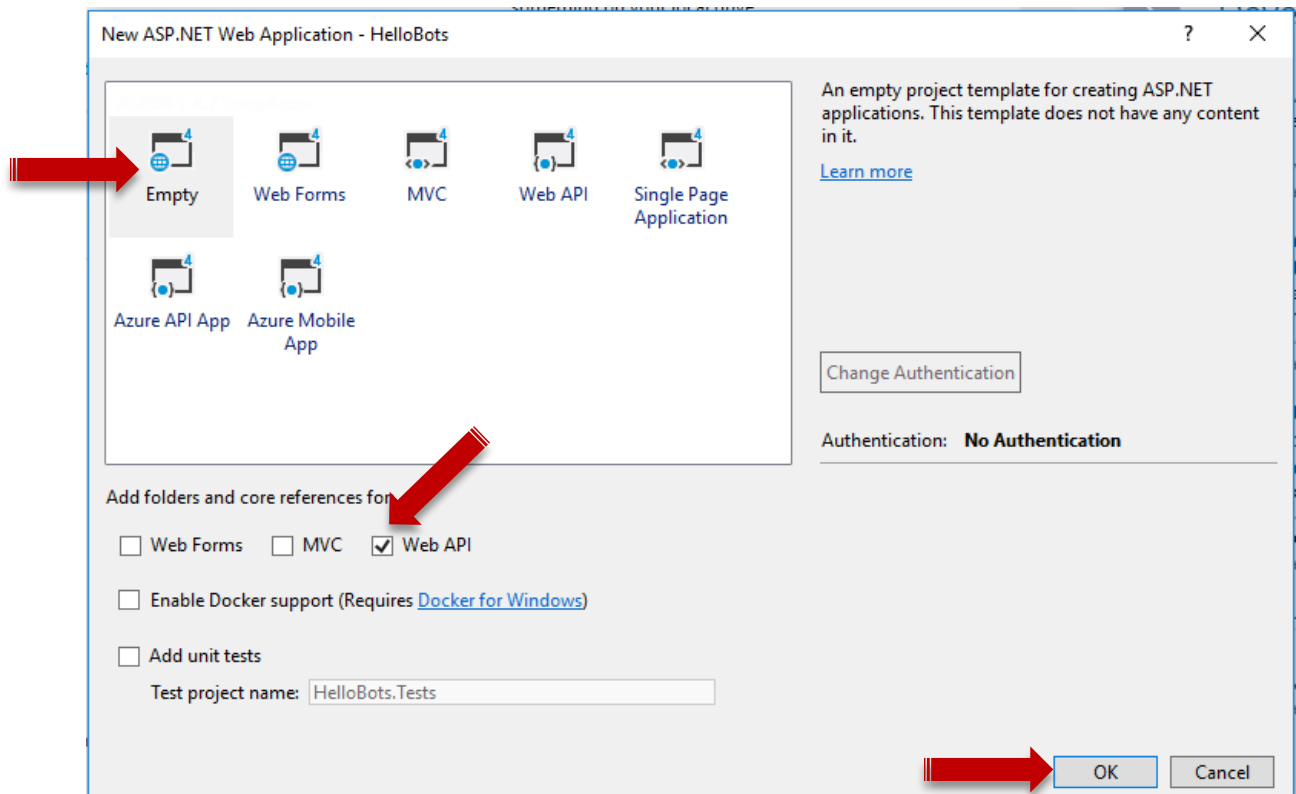


4. Proporciona el nombre, ubicación del Proyecto y haz clic en **OK** para continuar.



Name: HelloBots
Location: C:\Demos\
Solution name: HelloBots
☒ Create directory for solution
☐ Add to Source Control
OK Cancel

5. En la ventana **New ASP.NET Web Application** selecciona la plantilla **Empty**, activa la casilla **Web API** y haz clic en **OK** para aceptar la creación del proyecto.



New ASP.NET Web Application - HelloBots

An empty project template for creating ASP.NET applications. This template does not have any content in it.
[Learn more](#)

Change Authentication

Authentication: No Authentication

Add folders and core references for:

☐ Web Forms ☐ MVC ☒ Web API

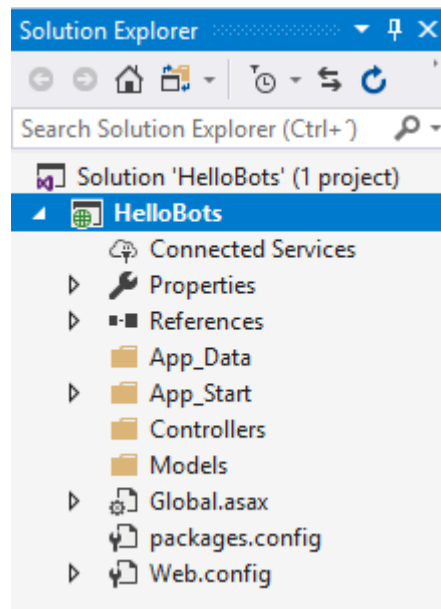
☐ Enable Docker support (Requires [Docker for Windows](#))

☐ Add unit tests

Test project name: HelloBots.Tests

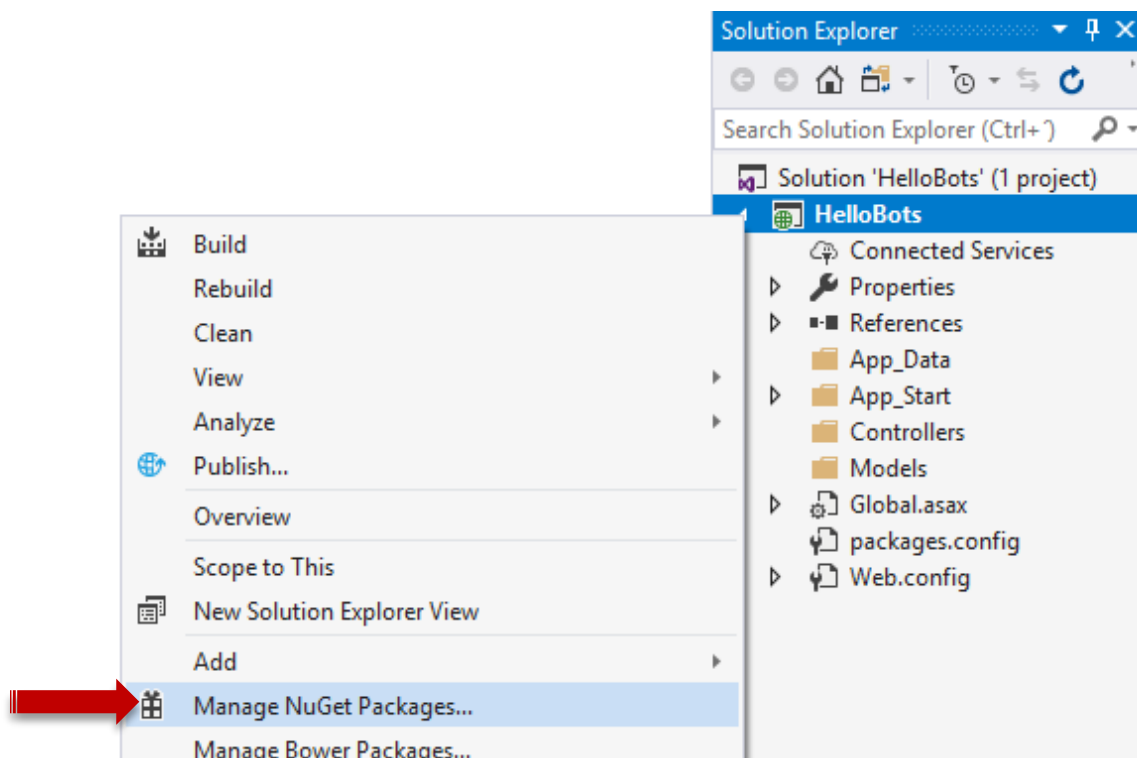
OK Cancel

Al finalizar la creación del proyecto, la ventana **Solution Explorer** mostrará la estructura de la aplicación similar a la siguiente.

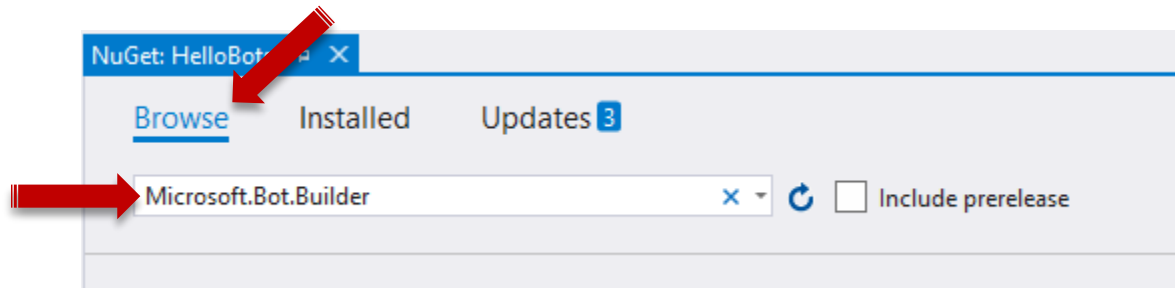


El siguiente paso será instalar el **Microsoft Bot Builder SDK for .NET** en el proyecto.

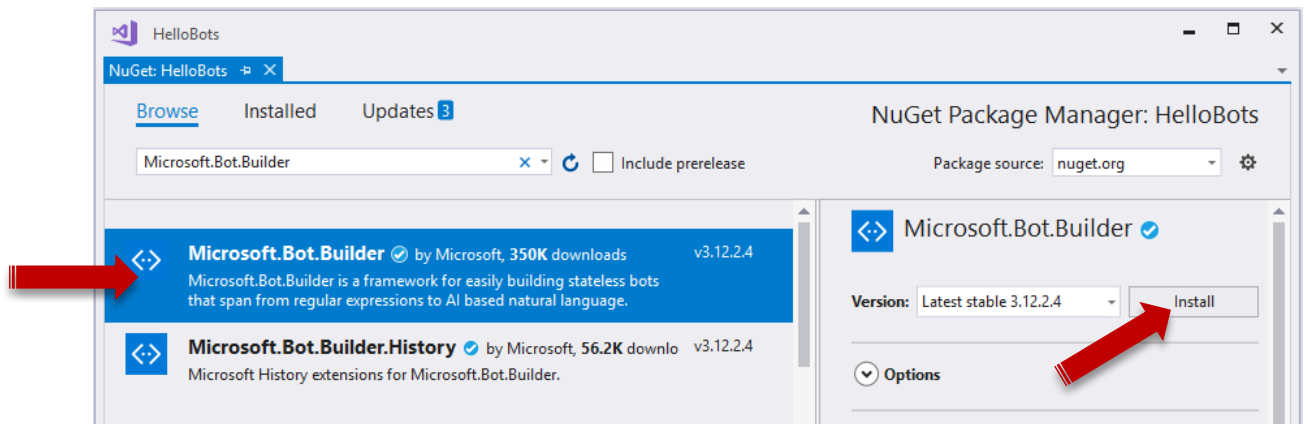
6. Selecciona la opción **Manage NuGet Packages...** del menú contextual del proyecto.



7. En la ventana **NuGet** selecciona la pestaña **Browse** y escribe **Microsoft.Bot.Builder** dentro del cuadro de búsqueda.



8. Selecciona el paquete **Microsoft.Bot.Builder** y haz clic en **Install** para instalar el paquete en el proyecto.

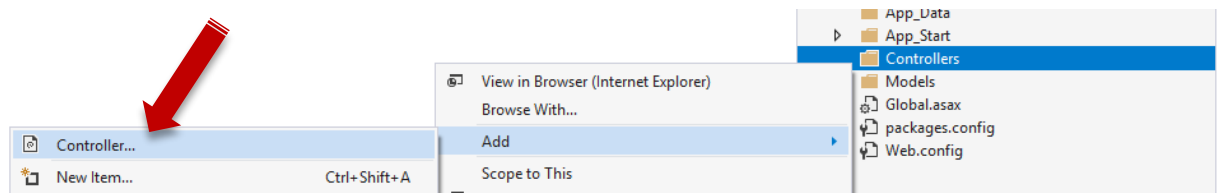


Acepta la realización de cambios en el proyecto y el acuerdo de licencias cuando te sea requerido.

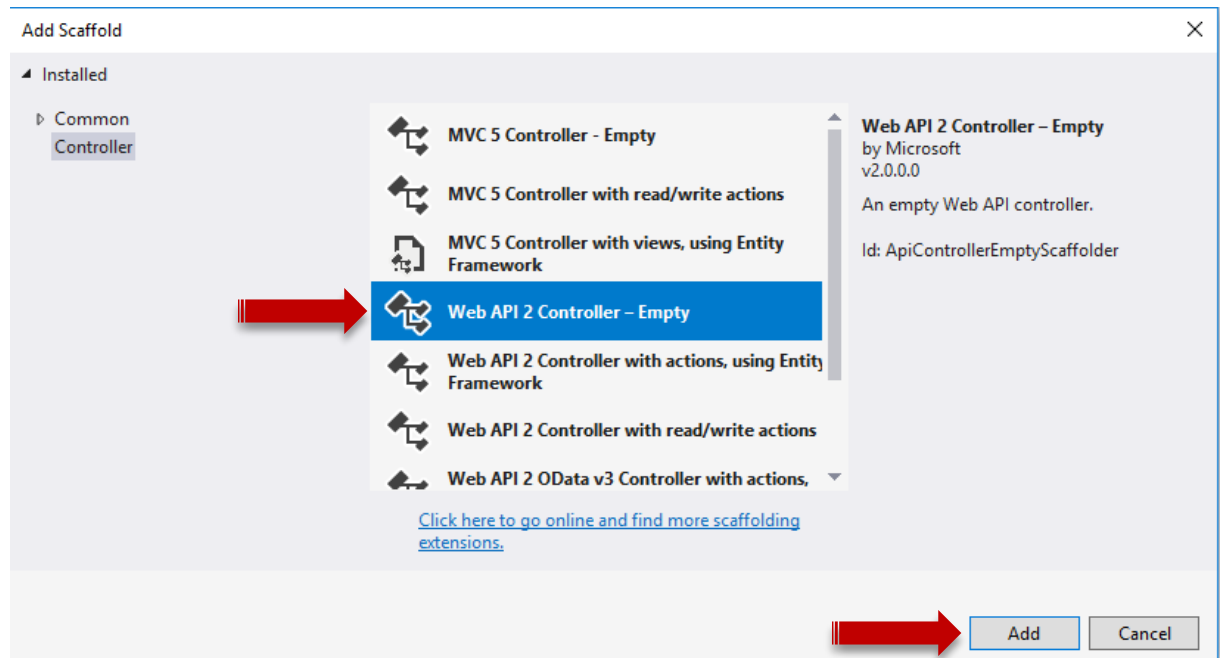
Tarea 2. Agregar el controlador de la aplicación.

En esta tarea agregarás el controlador de la aplicación Web API encargado de atender las peticiones de los usuarios.

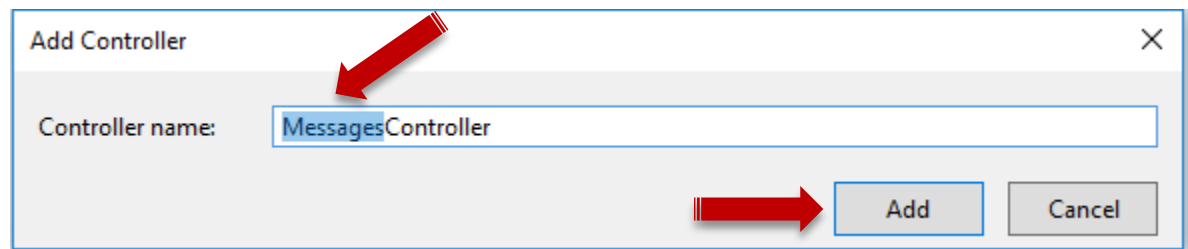
1. Selecciona la opción **Add > Controller...** del menú contextual de la carpeta **Controllers**.



2. En la ventana **Add Scaffold** selecciona la plantilla **Web API 2 Controller – Empty** y haz clic en **Add**.



3. En la ventana **Add Controller** escribe el nombre del controlador y haz clic en **Add**.



El código del controlador será similar al siguiente.

```
public class MessagesController : ApiController
{
}
```

Ejercicio 2: Implementando seguridad en el bot

Una estrategia que debemos seguir al desarrollar un bot es asegurar que el extremo de comunicación del bot (*Endpoint*) únicamente pueda ser accedido por el servicio **Bot Framework Connector**.

Para asegurar que el *endpoint* del bot sólo pueda ser accedido por el *Bot Framework Connector* debemos configurar el endpoint del bot para que utilice únicamente HTTPS y habilitar la autenticación *Bot Framework* registrando el bot en **Azure Bot Service**. Al registrar el bot, podremos obtener desde el portal de Azure el identificador de la aplicación y su contraseña.

Con el fin de enfocarnos en el desarrollo del bot, dejaremos el proceso de registro del bot con **Azure Bot Service** para otro laboratorio y sólo agregaremos a la aplicación los elementos necesarios para que pueda efectuarse la autenticación y autorización del bot.

Tarea 1. Agregar las credenciales de autenticación del bot en el archivo *web.config*.

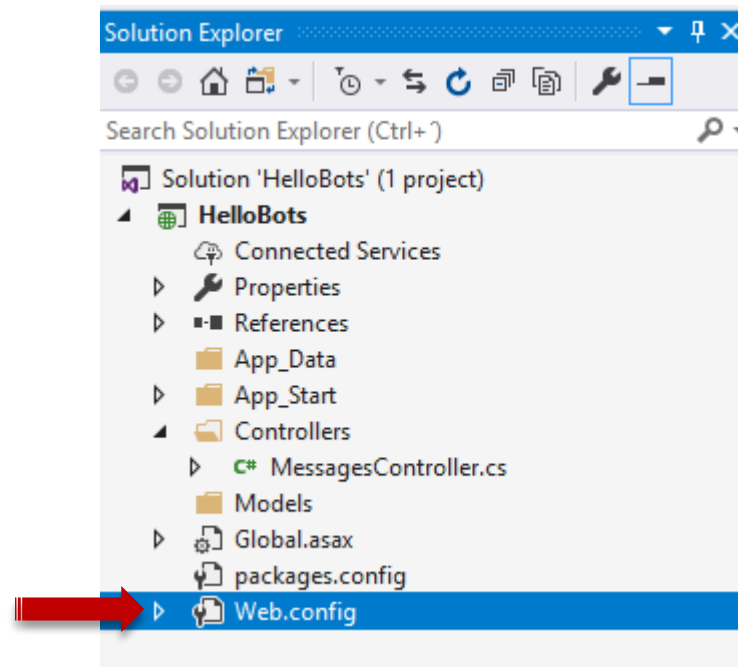
Una vez que el bot haya sido registrado en el portal de Azure podremos obtener ahí mismo los valores **MicrosoftAppId** y **MicrosoftAppPassword** necesarios para el proceso de autenticación del Bot con el *Bot Framework Connector*.

Los valores de autenticación podemos especificarlos como parámetros del atributo **BotAuthentication** o bien almacenarlos en el archivo *web.config*.

En caso de que los valores de autenticación no sean proporcionados, el proceso de autenticación no podrá llevarse a cabo. Vale la pena mencionar que en este laboratorio no contaremos con esos valores, sin embargo, eso no evitará que podamos probar la funcionalidad de nuestro bot utilizando el **Bot Framework Emulator**. El proceso de autenticación será necesario cuando el bot esté en un ambiente de producción.

Realiza los siguientes pasos para agregar los valores de autenticación en el archivo *web.config*.

1. Abre el archivo *web.config* de la raíz del proyecto.



2. Agrega el siguiente código dentro del elemento **appSettings**.

```
<configuration>
  <appSettings>
    <add key="MicrosoftAppId" value="" />
    <add key="MicrosoftAppPassword" value="" />
  </appSettings>
</system.web>
```

Posteriormente, cuando hayas registrado el bot con el **Bot Service** de Azure podrás reemplazar los valores correspondientes.

3. Guarda los cambios y cierra el archivo **web.config**.

Tarea 2. Agregar el atributo BotAuthentication al controlador.

Para implementar el proceso de autenticación del bot podemos decorar la clase controlador con el atributo **BotAuthentication**. En el atributo **BotAuthentication** podemos especificar como argumentos los valores **MicrosoftAppId** y **MicrosoftAppPassword** o bien no proporcionar argumentos en el atributo para que los valores sean obtenidos del archivo **web.config**.

El **Bot Framework Connector** proporciona una API REST que permite a un bot comunicarse con diversos canales tales como Skype, Email, Slack, etc. **Bot Framework Connector** facilita la comunicación entre el bot y el usuario transmitiendo mensajes del bot al canal y del canal hacia el bot.

En el *Bot Builder SDK for .NET*, la biblioteca *Microsoft.Bot.Connector* habilita el acceso al *Connector*.

1. Agrega el siguiente código a la clase del controlador para importar el espacio de nombres ***Microsoft.Bot.Connector***.

```
using Microsoft.Bot.Connector;
```

2. Agrega ahora el atributo ***BotAuthentication*** a la clase controlador.

```
[BotAuthentication]  
public class MessagesController : ApiController  
{  
}
```

Al especificar el atributo ***BotAuthentication*** sin parámetros, las credenciales serán tomadas del archivo *web.config*.

3. Guarda los cambios realizados.

Ejercicio 3: Creando diálogos con el Bot Builder SDK for .NET

Cuando creamos un bot utilizando el *Bot Builder SDK for .NET*, podemos utilizar diálogos para modelar una conversación y administrar el flujo de la conversación. Un diálogo puede estar compuesto por otros diálogos para maximizar su reutilización. En cualquier instante de tiempo, un contexto de diálogo mantiene una pila con los diálogos que se encuentran activos en la conversación.

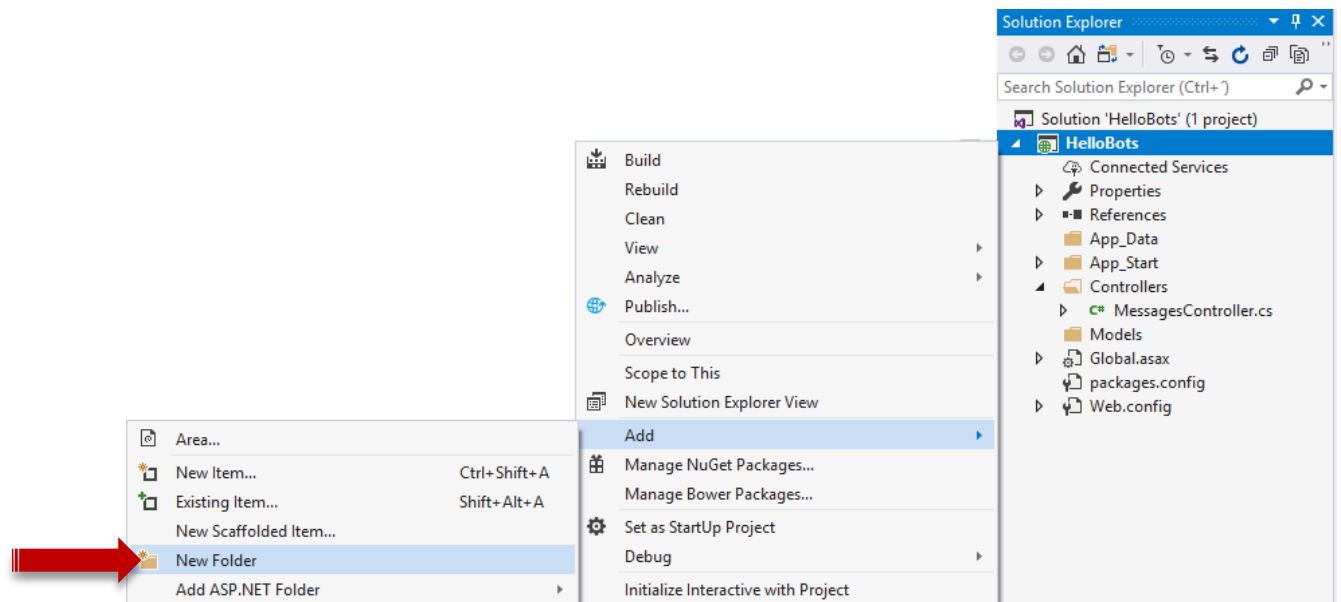
En este ejercicio agregaremos los elementos necesarios para implementar un dialogo en el chatbot.

Tarea 1. Definir un Diálogo.

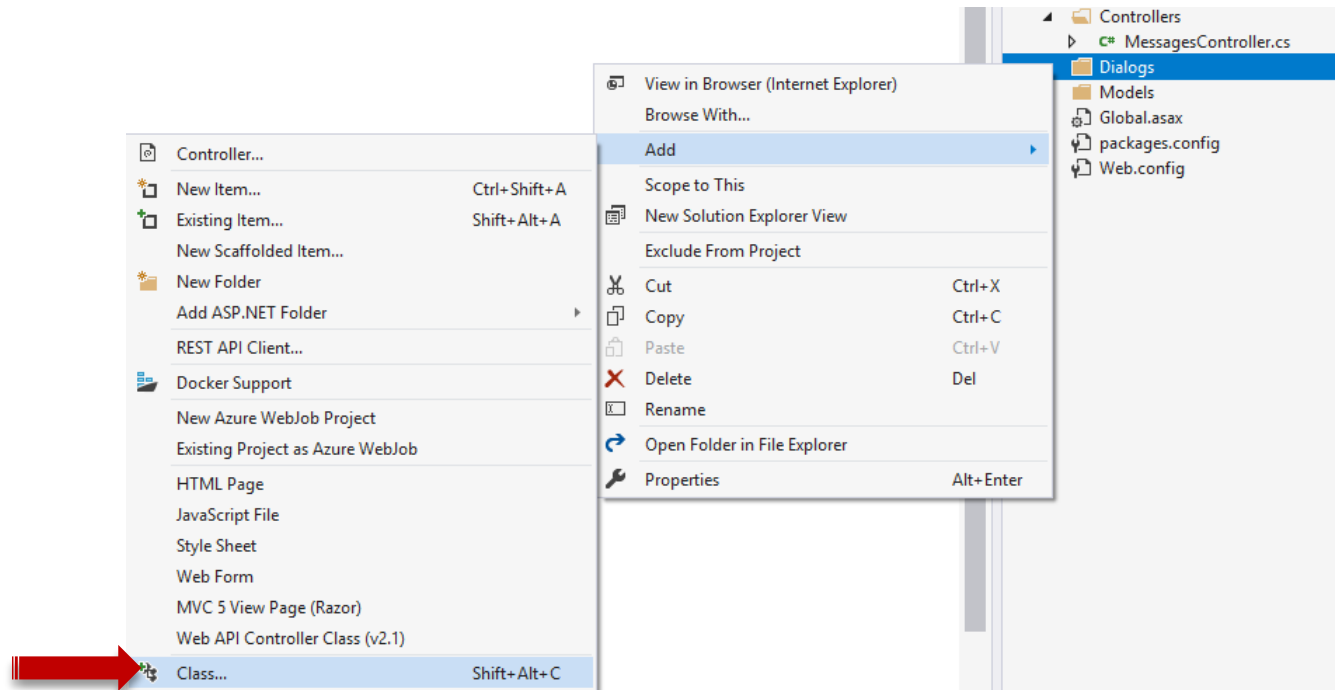
Cada dialogo en una conversación es una abstracción que encapsula su propio estado en una clase C# que implementa la interface *IDialog*.

En esta tarea agregaremos una clase al proyecto para definir un nuevo dialogo.

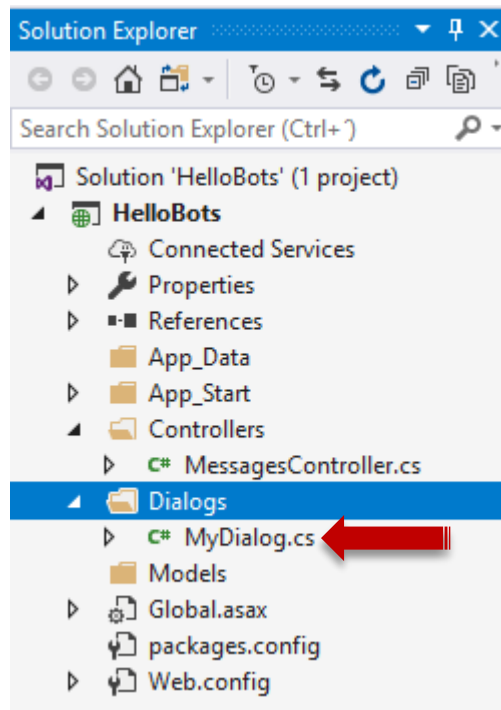
1. Selecciona la opción **Add > New Folder** del menú contextual del proyecto y crea una carpeta llamada **Dialogs**. En esta carpeta podremos organizar todos los diálogos que requiera nuestra aplicación bot.



2. Selecciona ahora la opción **Add > Class...** del menú contextual de la carpeta **Dialogs** para agregar una nueva clase llamada **MyDialog.cs**.



La estructura del proyecto será similar a la siguiente.



En el *Bot Builder SDK*, la biblioteca **Microsoft.Bot.Builder** nos permite implementar diálogos.

3. Agrega el siguiente código al inicio del archivo **MyDialog.cs**. Esto nos permitirá el acceso a todas las clases e interfaces del *Bot Builder Framework* que nos ayudarán a crear y utilizar diálogos.

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

4. Modifica ahora la definición de la clase **MyDialog** para indicar que implementa la interface **IDialog<string>**.

```
public class MyDialog:IDialog<string>  
{  
}
```

Existen 2 implementaciones de la interface *IDialog*, una con genéricos y otra sin genéricos. Cada bot puede devolver un valor del tipo especificado como parámetro del tipo genérico. Para nuestro ejemplo, el diálogo **MyDialog** trabajará con tipos **string**.

5. Decora la clase **MyDialog** con el atributo **[Serializable]**. El estado del diálogo es salvado automáticamente por el *Bot Framework* y, en consecuencia, el Framework indica que los objetos *IDialog* deben ser serializables. Sin este atributo, el Framework no podría guardar el estado del diálogo en el almacén de estado.

```
[Serializable]  
public class MyDialog:IDialog<string>  
{  
}
```

6. Agrega el siguiente código dentro de la clase **MyDialog** para implementar el método **StartAsync**. Este es el único método definido por la interface **IDialog**. El método recibe un objeto **IDialogContext** que será utilizado para enviar mensajes al usuario, suspender el diálogo actual, navegar a otro diálogo y, almacenar y recuperar el estado.

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

El método **Wait** del objeto **IDialogContext** suspende la ejecución actual, guarda el estado y espera entradas del usuario. Tan pronto recibe una entrada del usuario, este ejecuta el método que le es proporcionado como parámetro de tipo **ResumeAfter<T> Delegate** suspendiendo la ejecución actual. El método **ResumeAfter<T> Delegate** actúa como un delegado de continuación que es ejecutado cuando se reciben nuevos mensajes.

La implementación del método **MessageReceivedAsync** es el corazón de la clase **MyDialog**. Ese es el lugar donde podemos implementar la lógica del bot. La siguiente es la definición del delegado *ResumeAfter* <t>.

```
public delegate Task ResumeAfter<in T>(IDialogContext context, IAwaitable<in T> result);
```

Podemos notar que el delegado *ResumeAfter*<T> recibe un objeto *IDialogContext* y un objeto *IAwaitable*<T> conteniendo el texto enviado por el usuario.

7. Agrega el siguiente código para implementar el método **MessageReceivedAsync**.

```
private async Task MessageReceivedAsync(IDialogContext context,
    IAwaitable<object> result)
{
}
}
```

El *Bot Framework Connector* utiliza un objeto **Activity** para pasar información de ida y vuelta entre el bot y el canal (usuario). El tipo *Activity* más común es **message** pero existen otros tipos de *Activity* que pueden ser utilizados para comunicar varios tipos de información al bot o al canal. El parámetro **result** del método **MessageReceivedAsync** es el objeto **Activity** que contiene el mensaje proporcionado por el usuario.

8. Agrega el siguiente código dentro del método **MessageReceivedAsync** para obtener el objeto *Activity*.

```
var ActivityMessage = await result as Activity;
```

9. Agrega el siguiente código después de la línea anterior para obtener el mensaje enviado por el usuario.

```
var Message = ActivityMessage.Text;
```

En este punto podemos implementar una lógica para procesar el mensaje recibido y obtener la respuesta que le enviaremos al usuario. Por simplicidad, en este ejemplo únicamente enviaremos un eco del mensaje recibido.

10. Agrega ahora el siguiente código para enviar la respuesta al usuario utilizando el método **PostAsync** del objeto *IDialogContext*.

```
await context.PostAsync($"Escribiste: {Message}");
```

11. Finalmente, agrega el siguiente código para esperar el siguiente mensaje.

```
context.Wait(MessageReceivedAsync);
```

El siguiente es el código completo de la clase **MyDialog**.

```
[Serializable]
public class MyDialog:IDialog<string>
{
    public async Task StartAsync(IDialogContext context)
    {
        context.Wait(MessageReceivedAsync);
    }

    private async Task MessageReceivedAsync(IDialogContext context,
        IAwaitable<object> result)
    {
        var ActivityMessage = await result as Activity;
        var Message = ActivityMessage.Text;
        await context.PostAsync($"Escribiste: {Message}");
        context.Wait(MessageReceivedAsync);
    }
}
```

Tarea 2. Crear el punto de entrada del diálogo.

Un método **HttpPost** dentro del controlador de la aplicación es el punto de entrada del bot y el método **Conversation.SendAsync** es el punto de inicio del diálogo.

En esta tarea exploraremos la forma de administrar el flujo de un diálogo.

1. Abre el archivo **MessagesController.cs**.
2. Agrega el siguiente código al inicio del archivo para importar el espacio de nombres que nos permitirá el acceso a todas las clases e interfaces del *Bot Builder Framework* y que nos ayudarán a crear y utilizar un diálogo.

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

3. Agrega el siguiente código dentro de la clase **MessagesController** para definir el método **Post** que funcionará como el punto de entrada del bot.

```
public async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    // ...
}
```

El método **Post** es marcado como **async** debido a que el *Bot Builder* utiliza las facilidades de C# para manejar comunicaciones asíncronas. El método **Post** devuelve un objeto **Task** que representa la tarea que es responsable de enviar respuestas del mensaje recibido. En caso de existir una excepción, la tarea que es devuelta por el método contendrá la información de la excepción.

El *Bot Framework Connector* utiliza un objeto **Activity** para pasar información de ida y vuelta entre el bot y el canal (usuario). El valor del parámetro **activity** del método **Post** que es obtenido del cuerpo de la petición representa el objeto **Activity** proporcionado por el *Bot Connector*.

El tipo *Activity* más común es **message** pero existen otros tipos de *Activity* que pueden ser utilizados para comunicar varios tipos de información al bot o al canal.

4. Agrega el siguiente código dentro del método **Post** para invocar al método **Conversation.SendAsync** en caso de que el tipo **Activity** recibido sea un mensaje.

```
public async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    if(activity.Type == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new Dialogs.MyDialog());
    }
}
```

El método **Conversation.SendAsync** es clave para implementar diálogos con el *Bot Builder SDK for .NET*. Este método es una máquina de estado que sigue el principio de [Inversión de dependencias](#) y realiza los siguientes pasos:

1. Crea instancias de los componentes requeridos.
2. Deserializa el estado de la conversación (la pila de diálogos y el estado de cada dialogo en la pila) desde el **IBotDataStore**.
3. Reanuda el proceso de conversación donde fue suspendido y espera un mensaje.
4. Envía las respuestas.
5. Serializa el estado actualizado de la conversación y lo guarda de regreso en el **IBotDataStore**.

Cuando la conversación da inicio, el diálogo no contiene estado, por lo tanto, **Conversation.SendAsync** crea una instancia de la clase que implementa **IDialog** (*MyDialog*) e invoca a su método **StartAsync**. El método **StartAsync** invoca al método **IDialogContext.Wait** con el delegado de continuación para especificar el método que debe ser invocado cuando un nuevo mensaje sea recibido (**MessageReceivedAsync**).

El método **MessageReceivedAsync** espera un mensaje, publica una respuesta y espera el siguiente mensaje. Cada vez que **IDialogContext.Wait** es invocado, el bot entra en un estado suspendido y puede ser reiniciado en cualquier computadora que reciba el mensaje.

5. Agrega el siguiente código que permitirá procesar otro tipo de objetos **Activity** que podrían ser recibidos.

```
public async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    if(activity.Type == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new Dialogs.MyDialog());
    }
    else
    {
        HandleSystemMessage(activity);
    }
}
```

En un paso posterior crearemos el método *HandleSystemMessage*.

6. Agrega ahora el siguiente código para devolver la respuesta del método **Post**.

```
public async Task<HttpResponseMessage> Post([FromBody] Activity activity)
{
    if(activity.Type == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new Dialogs.MyDialog());
    }
    else
    {
        HandleSystemMessage(activity);
    }
    var Response = Request.CreateResponse(HttpStatusCode.OK);
    return Response;
}
```

7. Agrega el siguiente código dentro de la clase **MessagesController** para definir el método **HandleSystemMessage**.

```
void HandleSystemMessage(Activity message)
{
}
}
```

En este método podemos poner la lógica para procesar los tipos de **Activity** que decidamos implementar en el bot.

8. Agrega el siguiente código dentro del método **HandleSystemMessage** para tener la oportunidad de implementar algunos de los tipos más comunes de **Activity** soportados por el *Bot Builder SDK for .NET*. Por simplicidad de este ejercicio, no implementaremos la lógica de ellos pero explicaremos su funcionalidad.

```
void HandleSystemMessage(Activity message)
{
    switch(message.Type)
    {
        case ActivityTypes.ConversationUpdate:
            break;
        case ActivityTypes.ContactRelationUpdate:
            break;
        case ActivityTypes.Typing:
            break;
        case ActivityTypes.Ping:
            break;
        case ActivityTypes.DeleteUserData:
            break;
    }
}
```

La siguiente tabla contiene la descripción de los distintos tipos de *Activity* soportados por el *Bot Builder SDK for .NET*.

Tipo Activity	Interface	Descripción
Message	<code>IMessageActivity</code>	Representa una comunicación entre bot y usuario.
ConversationUpdate	<code>IConversationUpdateActivity</code>	Indica que el bot fue agregado a una conversación, otros miembros fueron agregados o eliminados de la conversación o que los metadatos de la conversación han cambiado.
ContactRelationUpdate	<code>IContactRelationUpdateActivity</code>	Indica que el bot fue agregado o eliminado de la lista de contactos de un usuario.
Typing	<code>ITypingActivity</code>	Indica que el usuario o bot en el otro extremo de la conversación está escribiendo una respuesta.
Ping	n/a	Representa un intento de determinar si el endpoint de un bot es accesible.
DeleteUserData	n/a	Indica a un bot que un usuario ha solicitado que el bot elimine cualquier dato del usuario que pueda haber almacenado.
EndOfConversation	<code>IEndOfConversationActivity</code>	Indica el final de una conversación.
Event	<code>IEventActivity</code>	Representa una comunicación enviada a un bot que no es visible para el usuario.

Invoke	IInvokeActivity	Representa una comunicación enviada a un bot para solicitar que realice una operación específica. Este tipo de actividad está reservado para uso interno por el Microsoft Bot Framework.
MessageReaction	IMessageReactionActivity	Indica que un usuario ha reaccionado a una actividad existente. Por ejemplo, un usuario hizo clic en el botón "Me gusta" de un mensaje.

9. Guarda los cambios.

Ejercicio 4: Probando el funcionamiento del bot

El siguiente paso será probar el funcionamiento del bot utilizando el **Bot Framework Emulator**. El emulador es una aplicación de escritorio que nos permite probar y depurar nuestro bot ejecutándose en la computadora local (*localhost*) o de manera remota a través de un túnel.

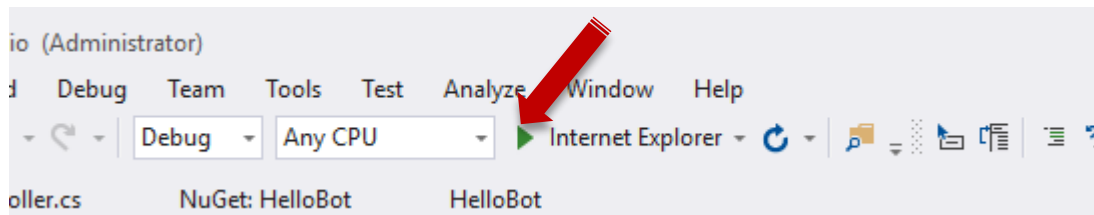
Tarea 1. Descargar e instalar el Bot Framework Emulator.

Realiza los siguientes pasos para descargar e instalar el **Bot Framework Emulator**.

1. Descarga el archivo **botframework-emulator-Setup-3.5.35.exe** desde el enlace <https://emulator.botframework.com/>.
2. Ejecuta el archivo **botframework-emulator-Setup-3.5.35.exe** para instalar el emulador

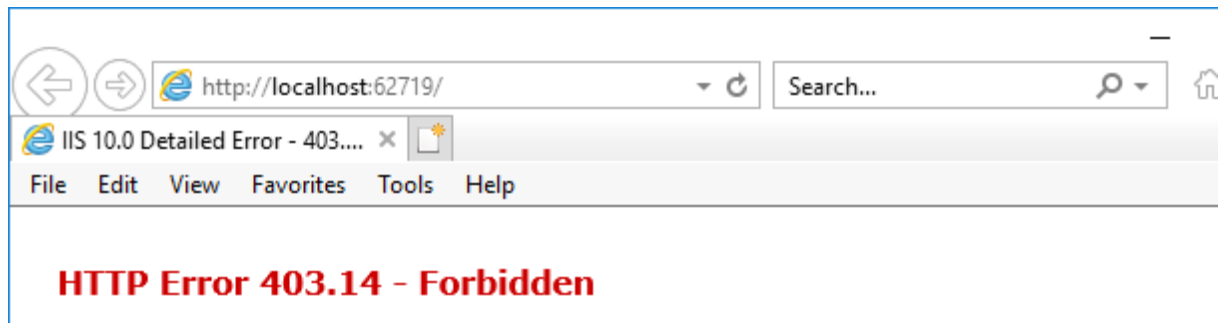
Tarea 2. Iniciar el bot.

1. Ejecuta tu bot en Visual Studio utilizando un navegador web como anfitrión de la aplicación. La imagen siguiente muestra que el bot se lanzará en *Internet Explorer* cuando el botón de ejecución sea seleccionado.



Al hacer clic en el botón de ejecución, Visual Studio compilará la aplicación, la desplegará hacia *localhost* y lanzará el explorador web.

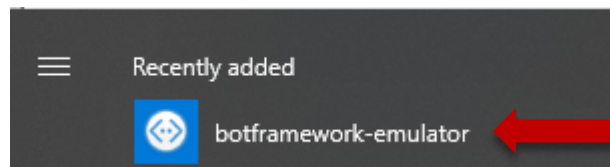
El explorador mostrará un mensaje de error debido a que no tenemos una interfaz de usuario disponible en nuestra Web API, sin embargo, el servicio estará ejecutándose y listo para procesar peticiones.



Tarea 3. Iniciar el emulador y conectar al bot.

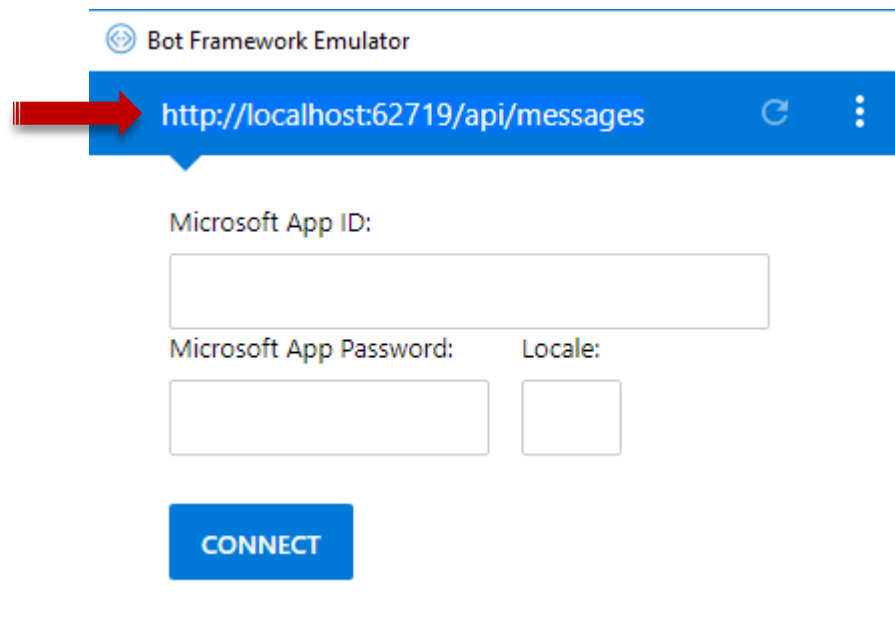
En este punto, tu bot se está ejecutando localmente. El siguiente paso será iniciar el emulador y conectarte a tu bot desde el emulador.

1. Abre el emulador.

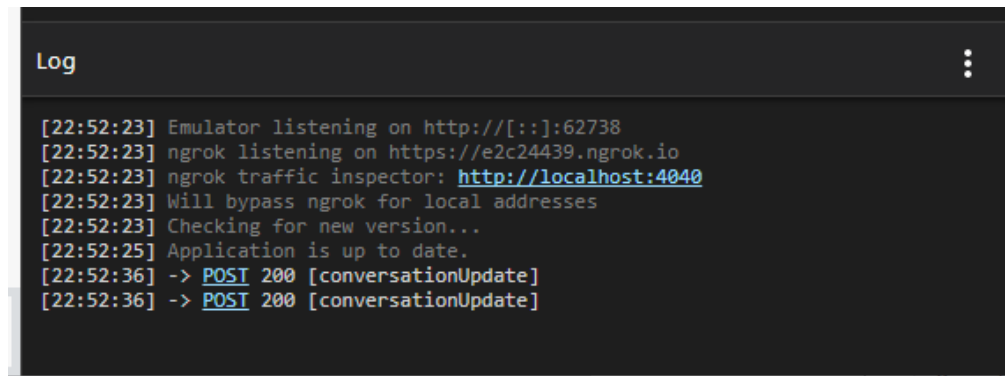


2. En la ventana **Bot Framework Emulator**, escribe la siguiente dirección remplazando el puerto correcto donde se está ejecutando tu bot y que es mostrado por el navegador web.

`http://localhost:puerto/api/messages`



3. Haz clic en **CONNECT**. No es necesario especificar **Microsoft App ID** ni **Microsoft App Password**. Esa información será necesaria solo cuando registres tu bot. Si la conexión se realiza exitosamente podrás ver los mensajes en el panel de detalle similar a los siguientes.

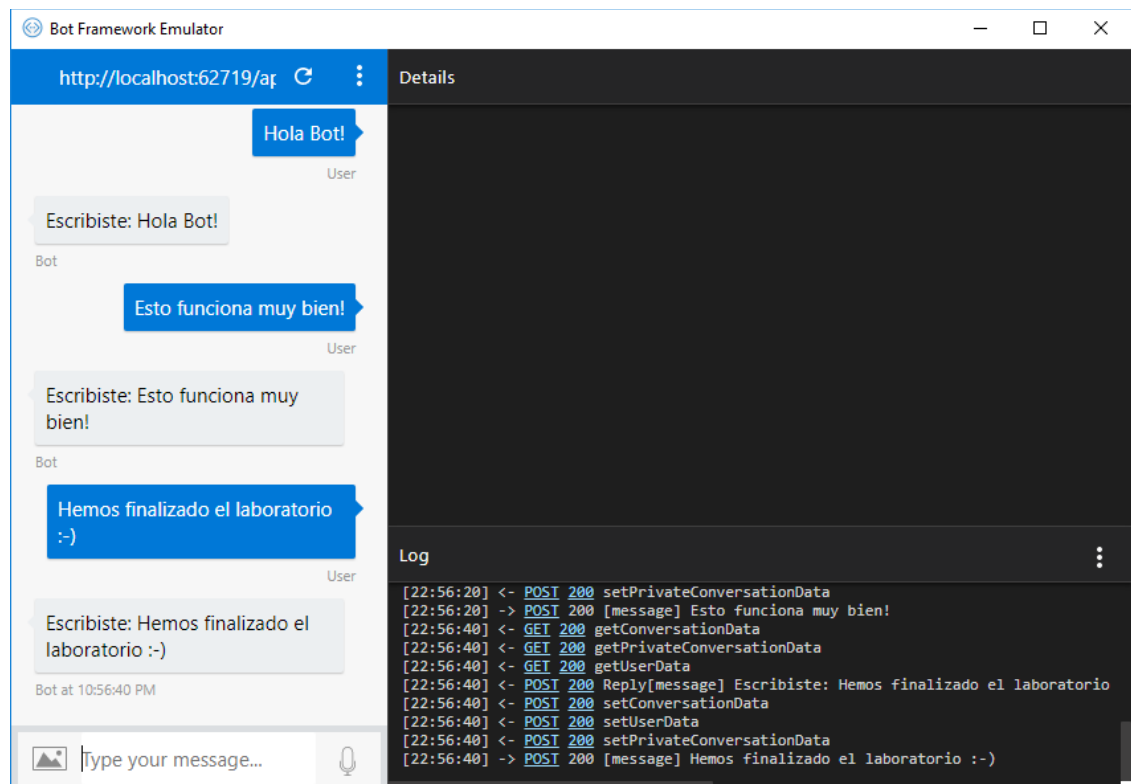


```
[22:52:23] Emulator listening on http://[::]:62738
[22:52:23] ngrok listening on https://e2c24439.ngrok.io
[22:52:23] ngrok traffic inspector: http://localhost:4040
[22:52:23] Will bypass ngrok for local addresses
[22:52:23] Checking for new version...
[22:52:25] Application is up to date.
[22:52:36] -> POST 200 [conversationUpdate]
[22:52:36] -> POST 200 [conversationUpdate]
```

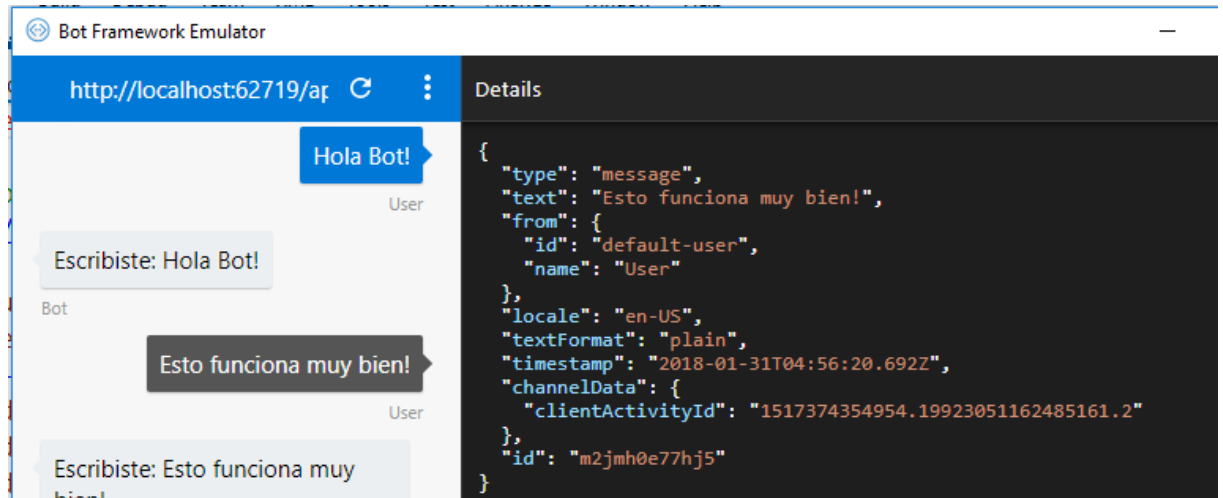
Tarea 4. Probar el bot.

Ahora que tu bot se está ejecutando localmente y está conectado al emulador, es momento de probar tu bot escribiendo algunos mensajes en el emulador.

1. Escribe algunos mensajes en el emulador. Podrás notar que el bot responde a cada uno de tus mensajes.



2. Haz clic en cualquier “burbuja” de dialogo. Podrás notar que el detalle del mensaje aparece en la ventana de detalle en formato JSON.



3. Regresa a Visual Studio y detén la ejecución.
4. Cierra Visual Studio.

Ejercicio 5: Creando un bot con Bot Service (opcional)

Bot Service proporciona los componentes principales para la creación de bots, incluido el **Bot Builder SDK** para desarrollar bots y el **Bot Framework** para conectar bots a canales. **Bot Service** proporciona cinco plantillas de bot con soporte para .NET y Node.js que podemos elegir al crear nuestros bots.

En este ejercicio, aprenderemos a utilizar **Bot Service** para crear un nuevo bot que utilice el **Bot Builder SDK**.

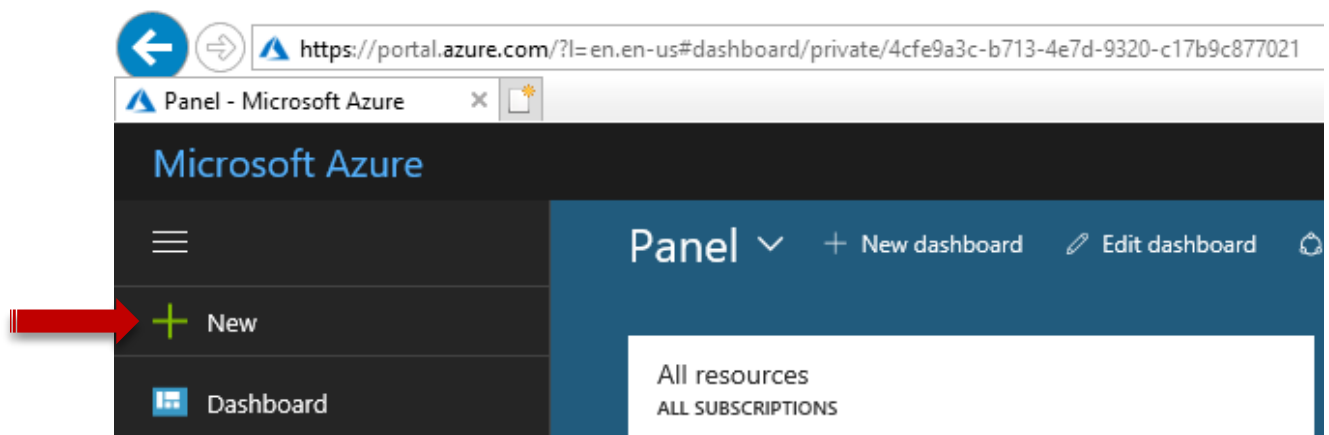
Para la realización de este ejercicio es necesario contar con una suscripción a Microsoft Azure. Puede obtenerse una cuenta de prueba en el siguiente enlace: <https://azure.microsoft.com/free/>

Tarea 1. Crear un nuevo servicio bot.

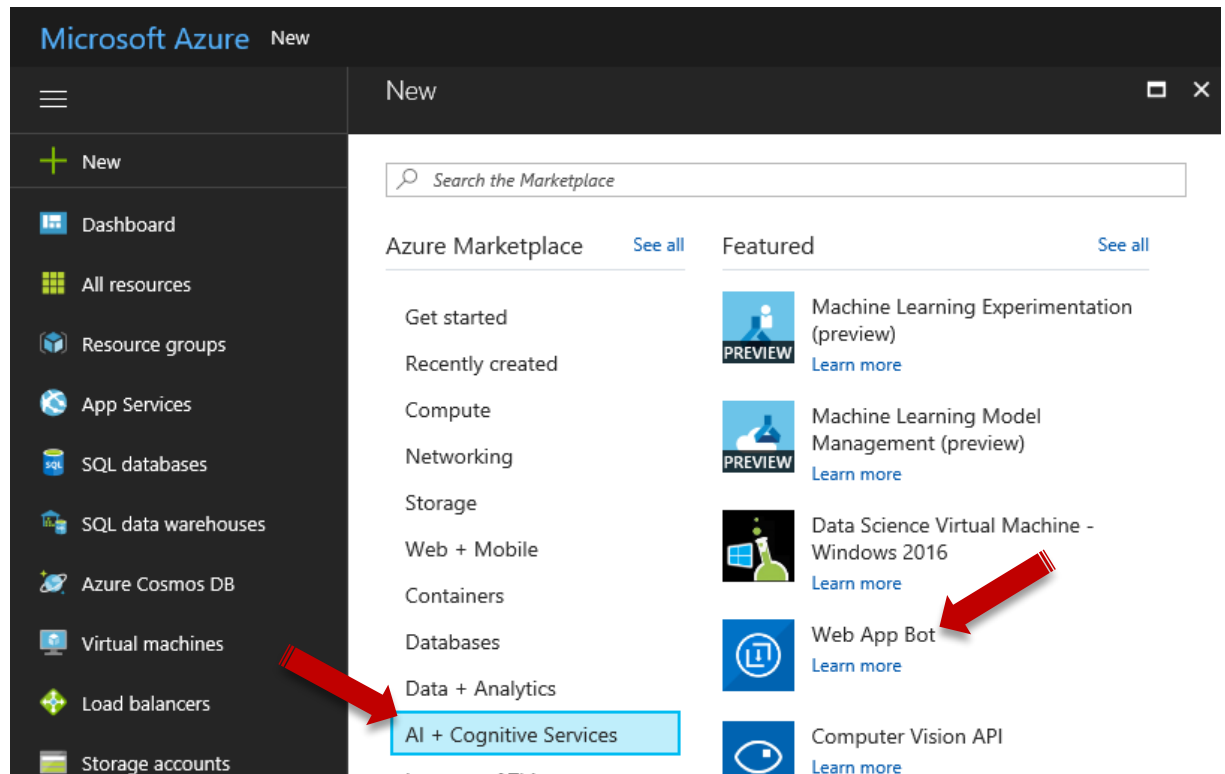
1. Inicia sesión en el portal de Microsoft Azure utilizando el siguiente enlace:

<https://portal.azure.com>

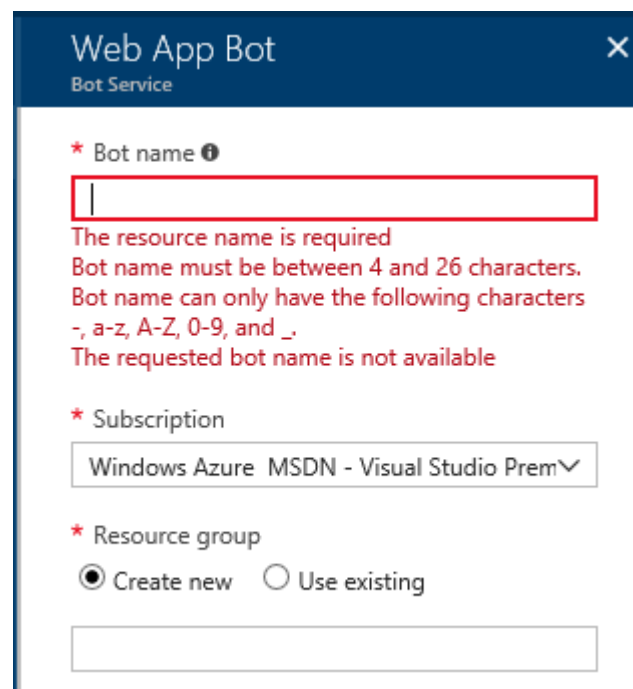
2. En el portal de Azure, haz clic en el botón **New**.



3. En el panel **New**, selecciona la opción **AI + Cognitive Services** > **Web App bot**.



Una nueva hoja se abrirá con información acerca de **Web App Bot**.



The screenshot shows the 'Web App Bot' configuration page in the Azure portal. The page has a title bar 'Web App Bot Bot Service'. It contains three sections: 'Bot name' with a text input field and error messages, 'Subscription' with a dropdown menu, and 'Resource group' with radio buttons for 'Create new' and 'Use existing'.

* Bot name ⓘ

The resource name is required
Bot name must be between 4 and 26 characters.
Bot name can only have the following characters -, a-z, A-Z, 0-9, and _.
The requested bot name is not available

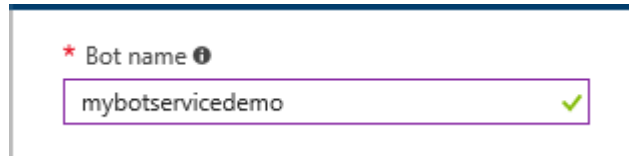
* Subscription

Windows Azure MSDN - Visual Studio Prem▼

* Resource group

☒ Create new ☐ Use existing

- Proporciona el nombre del Bot siguiendo las reglas mostradas en rojo. Este es el nombre del bot que aparecerá en canales y directorios. Este nombre se puede cambiar en cualquier momento.



* Bot name ⓘ

mybot servicedemo ✓

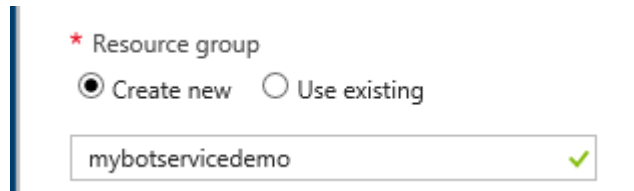
- Selecciona la suscripción Azure a utilizar.



* Subscription

Windows Azure MSDN - Visual Studio Prem ✓

- Selecciona un grupo de recursos. En caso de no tener un grupo de recursos puedes aceptar la creación del grupo de recursos sugerido.

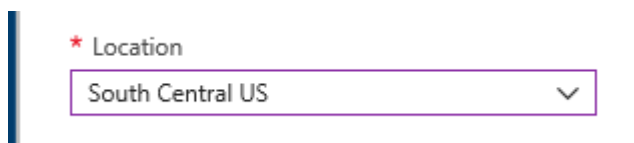


* Resource group

☒ Create new ☐ Use existing

mybot servicedemo ✓

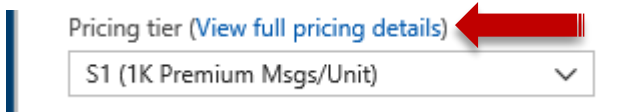
- Selecciona la ubicación geográfica para el grupo de recursos. Es recomendable seleccionar la ubicación más cercana a los usuarios del bot. La ubicación no se puede cambiar una vez que el bot haya sido creado.



* Location

South Central US ✓

- Haz clic en el enlace **View full pricing details** para ver los distintos niveles de pago.



Pricing tier ([View full pricing details](#))

S1 (1K Premium Msgs/Unit) ✓

La hoja con opciones será mostrada. Al momento de crear este documento se encuentran disponibles únicamente 2 opciones. Una de ellas gratuita.

Web App Bot

Bot Service

* Bot name

mybotsservicedemo

* Subscription

Windows Azure MSDN - Visual Studio Prem

* Resource group

☒ Create new ☐ Use existing

mybotsservicedemo

* Location

South Central US

Pricing tier (View full pricing details)

S1 (1K Premium Msgs/Unit)

* App name

mybotsservicedemo

.azurewebsites.net

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)

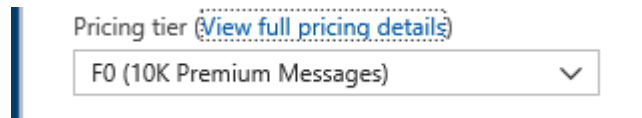
9. Selecciona la opción **F0 Free** y haz clic en **Select** para aceptar.

Bot Service Premium Messages pricing includes messages sent/received via t [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

Select

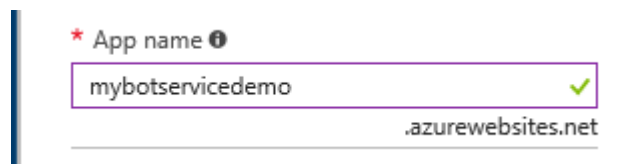
El plan seleccionado será mostrado en el cuadro de lista desplegable. En ese mismo cuadro de lista también es posible seleccionar el nivel de precio.



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

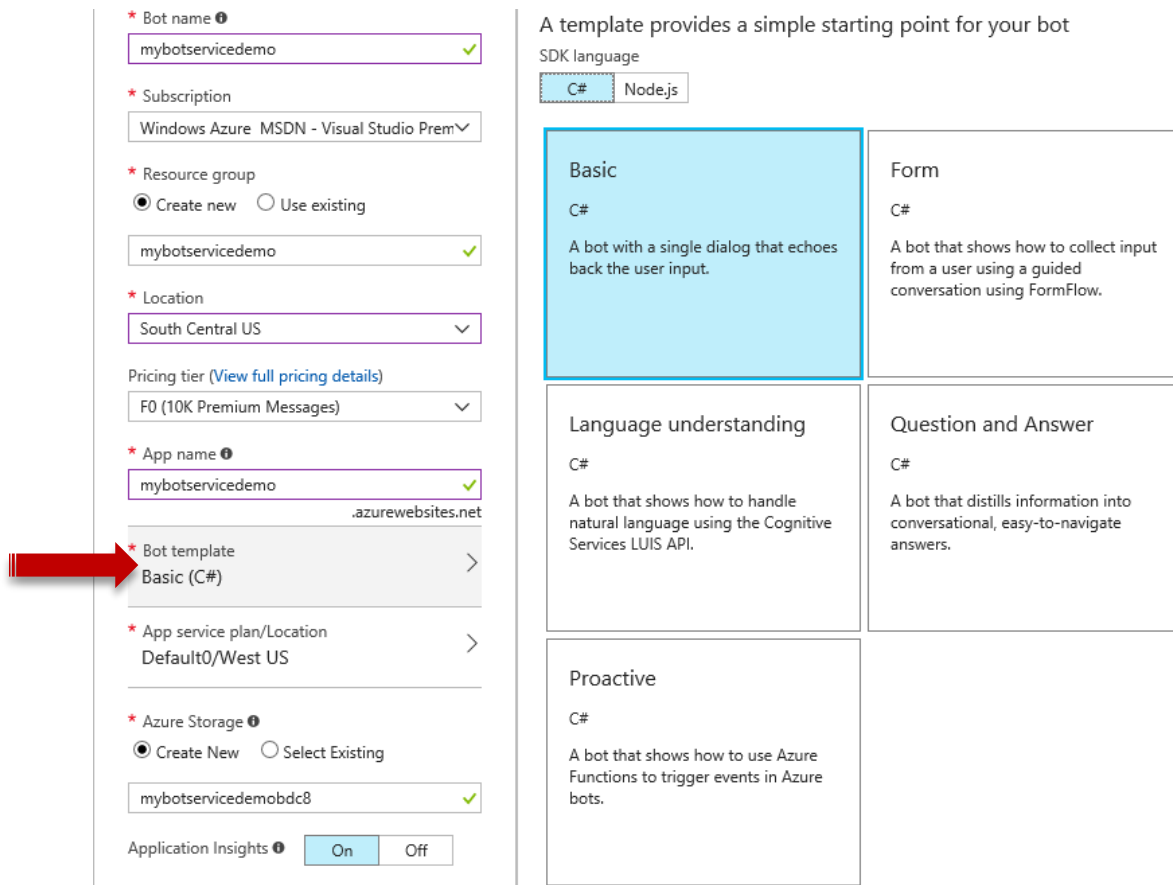
10. Proporciona el nombre de la aplicación. Este nombre formará parte del URL del endpoint de tu bot. Por ejemplo, si el nombre es *mybot servicedemo*, el URL del bot será *https://mybot servicedemo.azurewebsites.net*

El nombre debe contener únicamente caracteres alfanuméricos y de subrayado. Hay un límite de 35 caracteres para este campo. El nombre de la aplicación no se puede cambiar una vez que se crea el bot.



* App name ⓘ
mybot servicedemo ✓
.azurewebsites.net

11. Haz clic en **Bot template** para ver las plantillas de aplicaciones bot disponibles.



* Bot name ⓘ
mybot servicedemo ✓

* Subscription
Windows Azure MSDN - Visual Studio Prem ▼

* Resource group
☒ Create new ☐ Use existing
mybot servicedemo ✓

* Location
South Central US ▼

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

* App name ⓘ
mybot servicedemo ✓
.azurewebsites.net

* Bot template
Basic (C#) >

* App service plan/Location
Default0/West US >

* Azure Storage ⓘ
☒ Create New ☐ Select Existing
mybot servicedemobdc8 ✓

Application Insights ⓘ ☒ On ☐ Off

A template provides a simple starting point for your bot

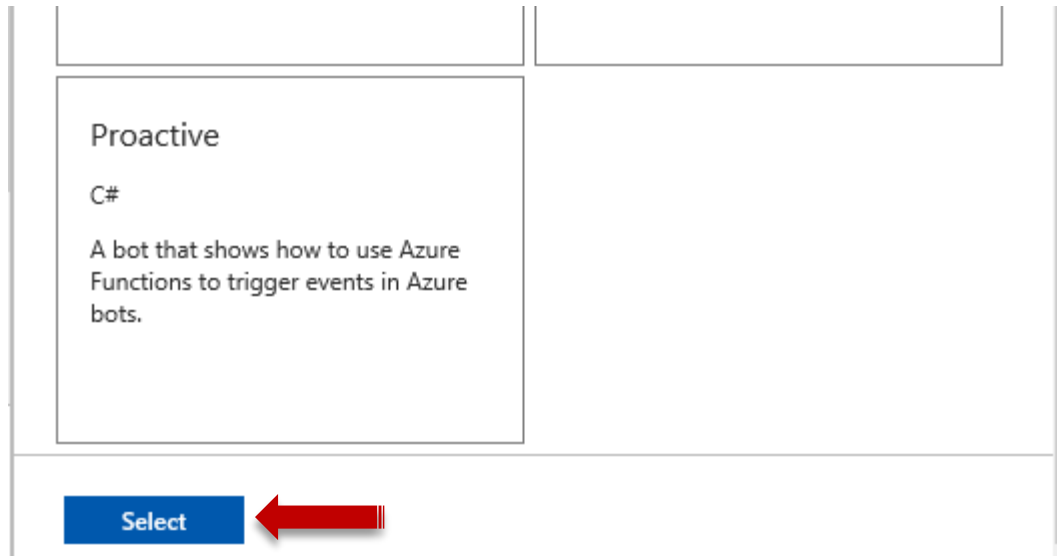
SDK language
☒ C# ☐ Node.js

<p>Basic</p> <p>C#</p> <p>A bot with a single dialog that echoes back the user input.</p>	<p>Form</p> <p>C#</p> <p>A bot that shows how to collect input from a user using a guided conversation using FormFlow.</p>
<p>Language understanding</p> <p>C#</p> <p>A bot that shows how to handle natural language using the Cognitive Services LUIS API.</p>	<p>Question and Answer</p> <p>C#</p> <p>A bot that distills information into conversational, easy-to-navigate answers.</p>
<p>Proactive</p> <p>C#</p> <p>A bot that shows how to use Azure Functions to trigger events in Azure bots.</p>	

En esta hoja puedes seleccionar una de 5 posibles plantillas con soporte a C# o Node.js.

12. Selecciona la plantilla **Basic** con soporte de C#, esta plantilla es similar al bot que creamos anteriormente en este laboratorio.

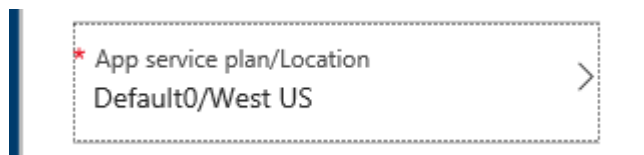
13. Haz clic en el botón **Select** ubicado en la parte inferior de la hoja para aceptar la selección.



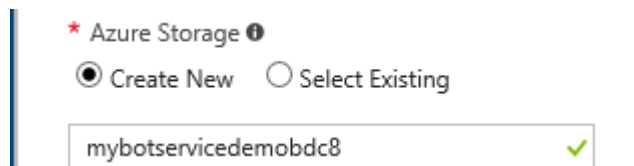
La plantilla seleccionada será mostrada.



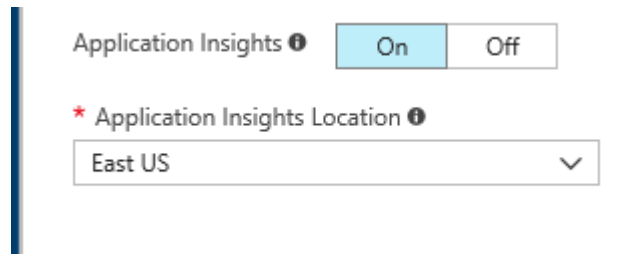
14. Selecciona la ubicación del plan de *App Service*. Es recomendable seleccionar la ubicación más cercana a los posibles usuarios del bot.



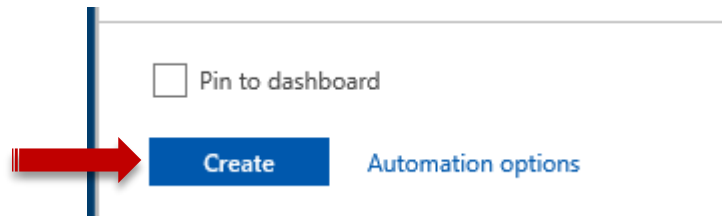
15. Selecciona o crea una cuenta de almacenamiento de Azure.



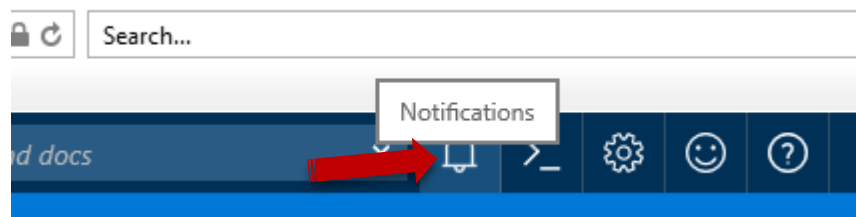
16. Decide si deseas activar o desactivar las estadísticas de aplicaciones. Para este ejercicio puedes aceptar las opciones predeterminadas.



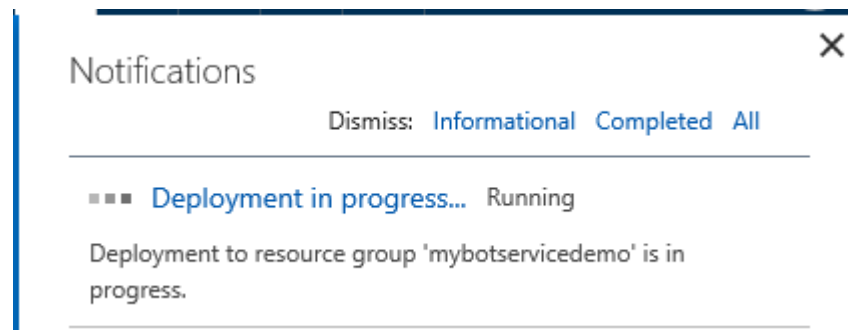
17. Haz clic en **Create** para crear el servicio y desplegar el bot a la nube. Este proceso puede tardar varios minutos.



18. Haz clic en el botón de notificaciones para verificar el estado del despliegue.



Las notificaciones cambiarán de *Despliegue en curso ...* a *Despliegue exitoso*.



Notifications

Dismiss: [Informational](#) [Completed](#) [All](#)**Deployment succeeded** 4:19 PM

Deployment 'mybot servicedemo_csharp_basic_sdk' to resource group 'mybot servicedemo' was successful.

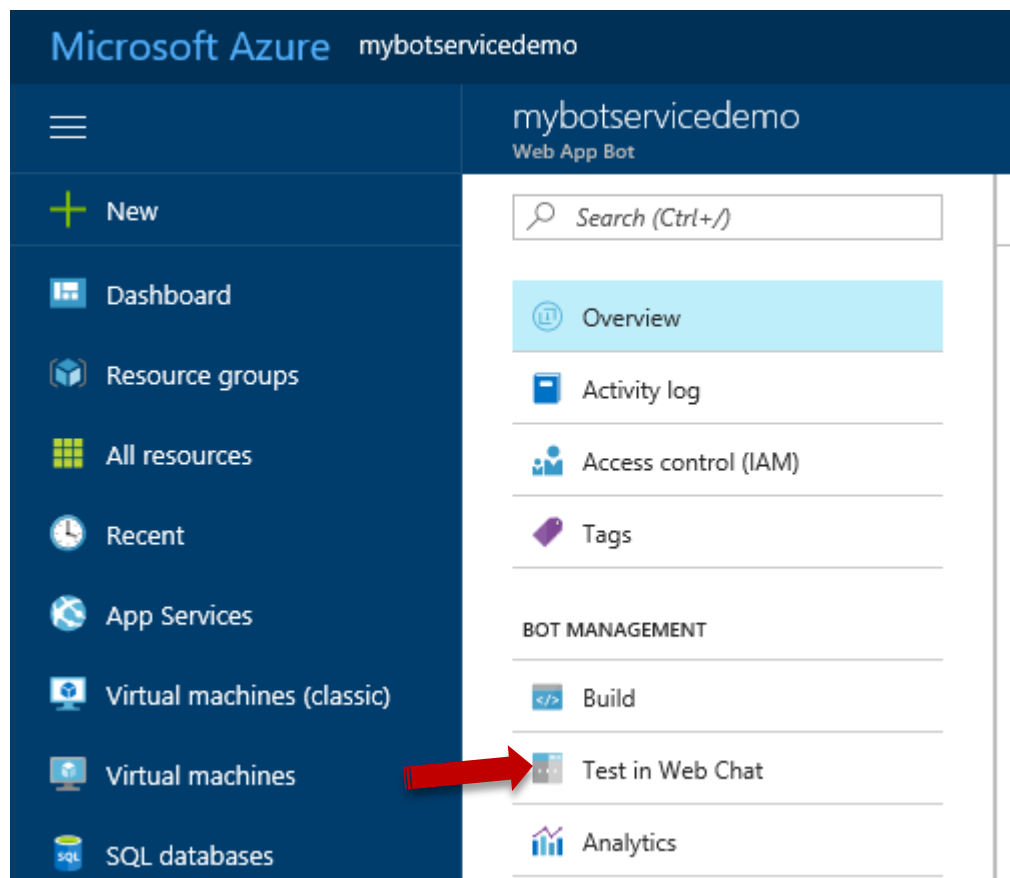
[Go to resource](#)[Pin to dashboard](#)

19. Haz clic en el botón **Go to resource** para abrir la hoja de recursos del bot.

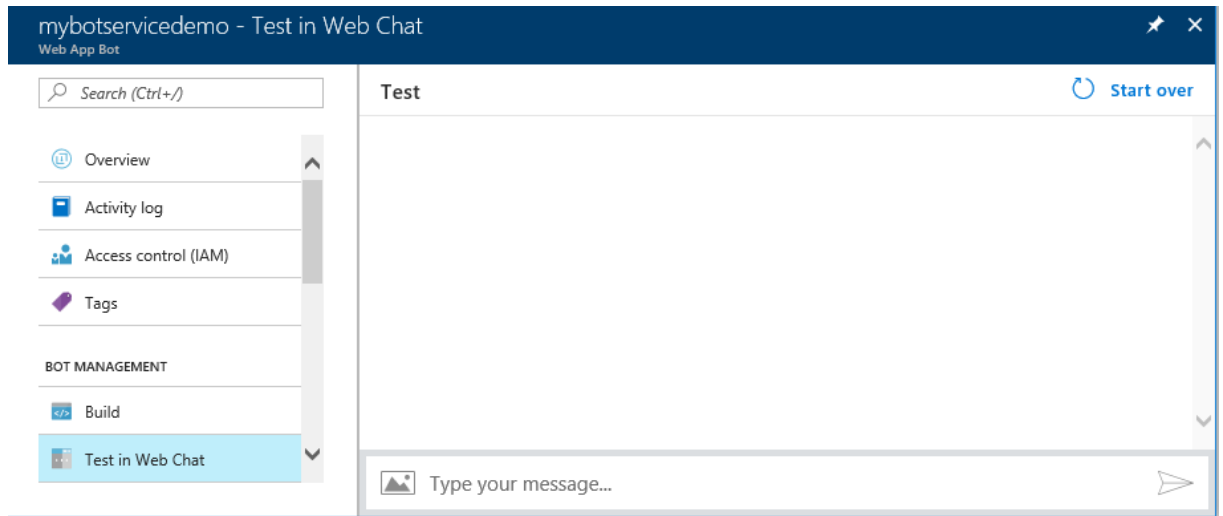
Tarea 2. Probar el bot en el Web Chat de Azure.

En esta tarea probaremos el funcionamiento del bot creado utilizando una herramienta que proporciona *Bot Service*: el control **Web Chat**.

1. En la hoja de recursos del bot, selecciona la opción **Test in Web Chat**.



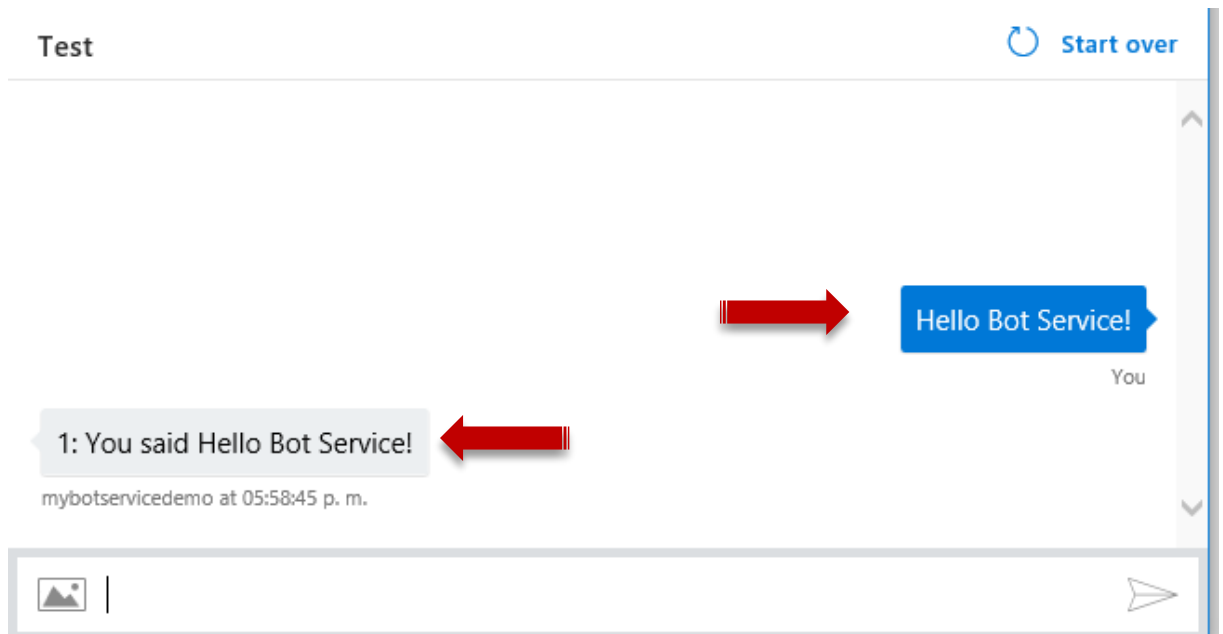
Bot Service cargará el control Web Chat y se conectará a tu bot.



2. Escribe algún texto para probar el funcionamiento del bot y presiona **<enter>**.



3. Podrás ver tu mensaje y la respuesta del bot.



Tarea 3. Probar el bot con el emulador.

Utilizando el emulador, podemos chatear con nuestro bot e inspeccionar los mensajes que el bot envía y recibe. El emulador muestra los mensajes tal y como aparecerían en una interfaz de usuario de un chat web. El emulador también registra las solicitudes y respuestas JSON conforme intercambiamos mensajes con el bot.

Cuando utilizamos el emulador en Windows y lo ejecutamos detrás de un firewall u otro límite de red y deseamos conectarnos a un bot alojado de forma remota, debemos instalar y configurar el software **ngrok tunneling**. El *Bot Framework Emulator* se integra estrechamente con el software *ngrok tunneling* (desarrollado por *inconshreveable*), y puede iniciarlo automáticamente cuando sea necesario.

En esta tarea configurarás el *Bot Framework Emulator* para utilizar **ngrok** y poder conectarte con el bot hospedado en Azure.

1. Descarga **ngrok** desde el siguiente enlace.

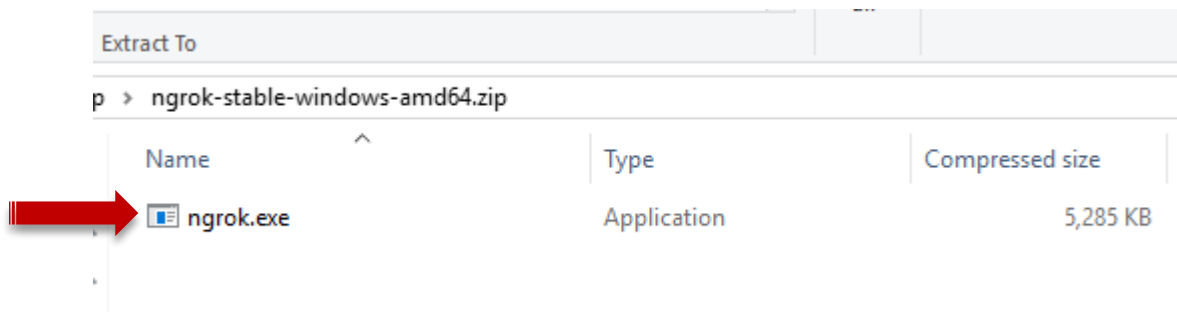
<https://ngrok.com/download>

Download ngrok

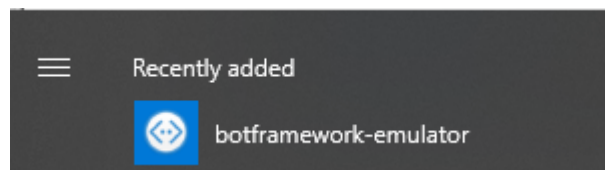
ngrok is easy to install. Download a single binary with *zero run-time dependencies* for any major platform.



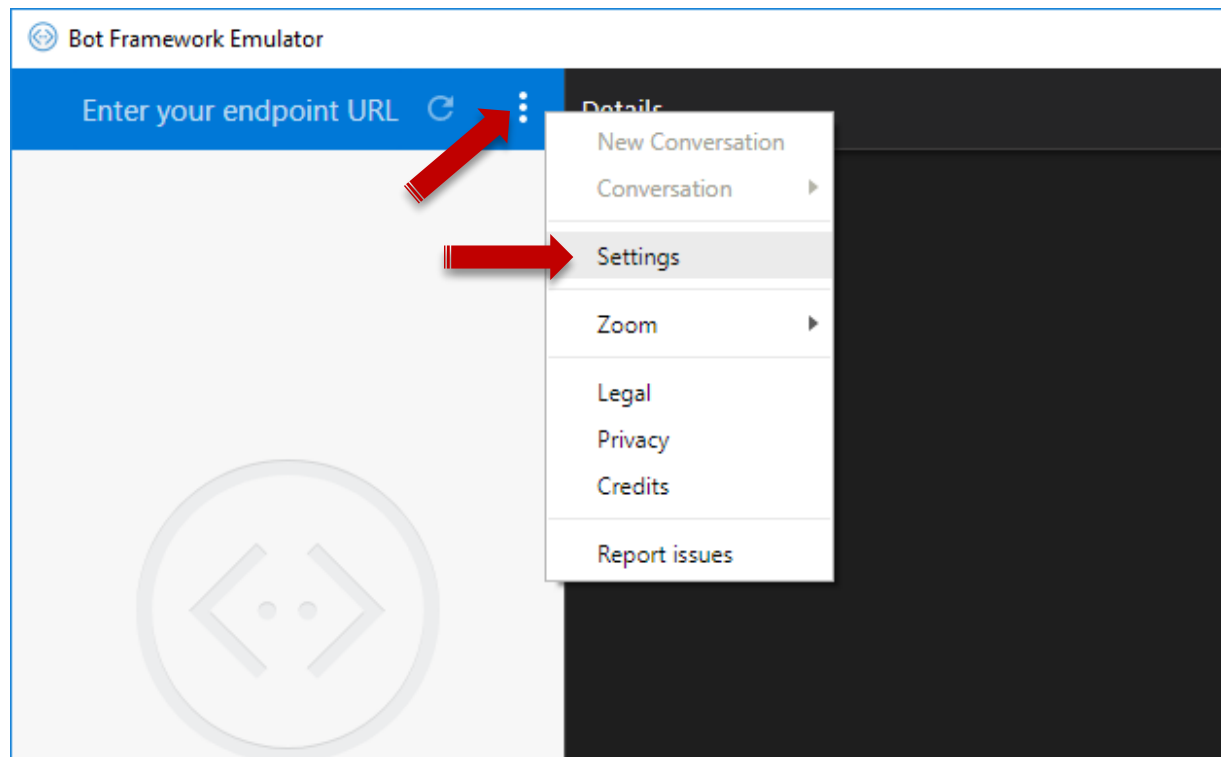
2. Extrae el ejecutable hacia una carpeta de tu disco local.



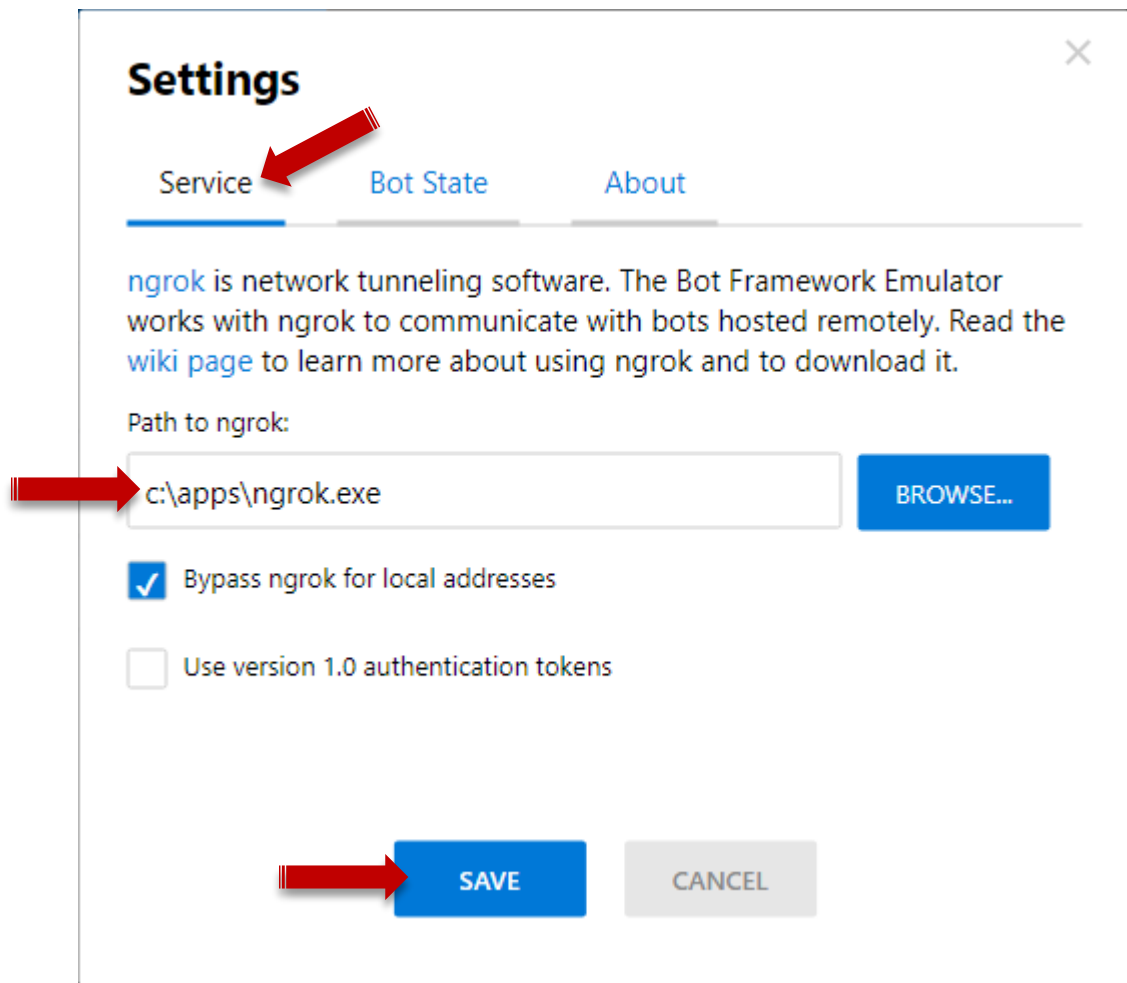
3. Abre el *Bot Framework Emulator*.



4. Selecciona la opción **Settings** del emulador.

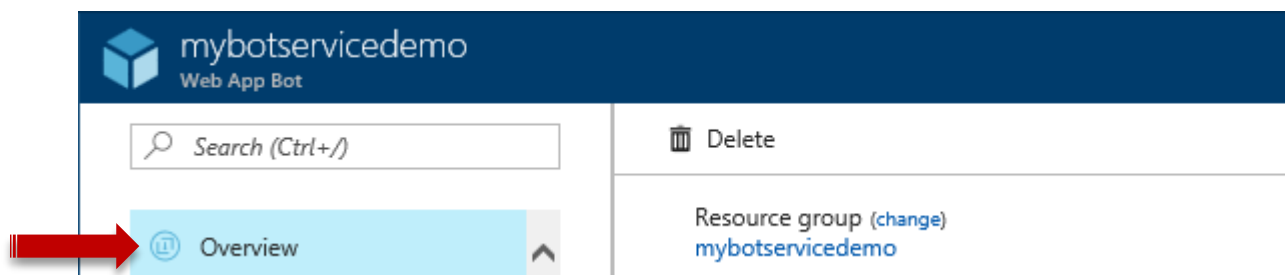


5. En la ventana **Settings** proporciona la ruta del archivo ejecutable **ngrok.exe** y haz clic en **SAVE** para guardar los cambios.



Ahora es momento de probar el funcionamiento de tu bot hospedado en Azure.

6. Dirígete a la hoja de recursos de tu bot en Azure.
7. Haz clic en la opción **Overview**.



8. Copia el URL del endpoint de tu bot.

Delete

Resource group (change)
mybotsservicedemo
Subscription (change)
Windows Azure MSDN - Visual Studio Premium

Bot Service pricing tier
F0
Messaging endpoint
https://mybotsservicedemo.azurewebsites.net/api/messages

Click to copy

- Pega el URL de tu servicio en la barra de direcciones del emulador.

Bot Framework Emulator

https://mybotsservicedemo.azurewebsites.net/api/messages

Details

Microsoft App ID:

Microsoft App Password:

Locale:

CONNECT

- Dirígete a la hoja de recursos de tu bot en Azure.

- Haz clic en la opción **Application Settings**.

APP SERVICE SETTINGS

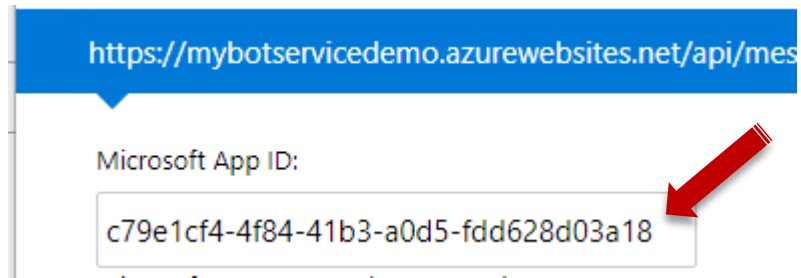
Application Settings

All App service settings

- Copia el valor de la llave **MicrosoftAppId**.

BotId	mybotsservicedemo
MicrosoftAppId	c79e1cf4-4f84-41b3-a0d5-fdd628d03a18

- Pega el valor en el cuadro **Microsoft App ID** del emulador.

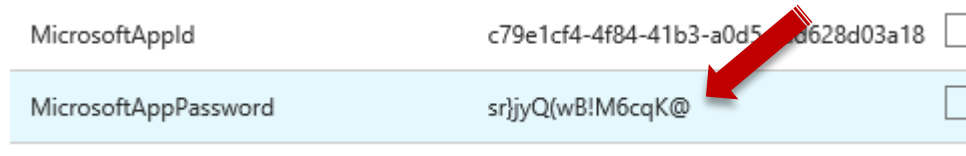


https://mybot servicedemo.azurewebsites.net/api/mes

Microsoft App ID:

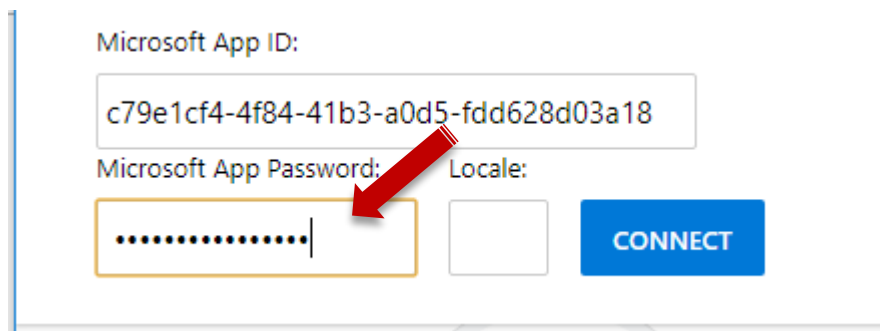
c79e1cf4-4f84-41b3-a0d5-fdd628d03a18

14. Copia el valor de la llave **MicrosoftAppPassword** desde la hoja de recursos de tu bot.



MicrosoftAppId	c79e1cf4-4f84-41b3-a0d5-fdd628d03a18
MicrosoftAppPassword	sr}jyQ(wB!M6cqK@

15. Pega el valor en el cuadro **Microsoft App Password** del emulador.



Microsoft App ID:

c79e1cf4-4f84-41b3-a0d5-fdd628d03a18

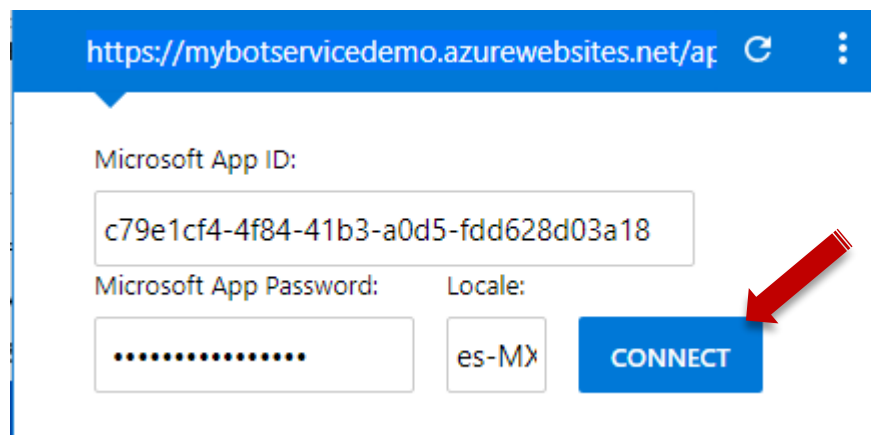
Microsoft App Password:

Locale:

CONNECT

16. Opcionalmente puedes escribir tu configuración regional en el campo **Locale** del emulador.

17. Haz clic en **CONNECT**.



https://mybot servicedemo.azurewebsites.net/api/mes

Microsoft App ID:

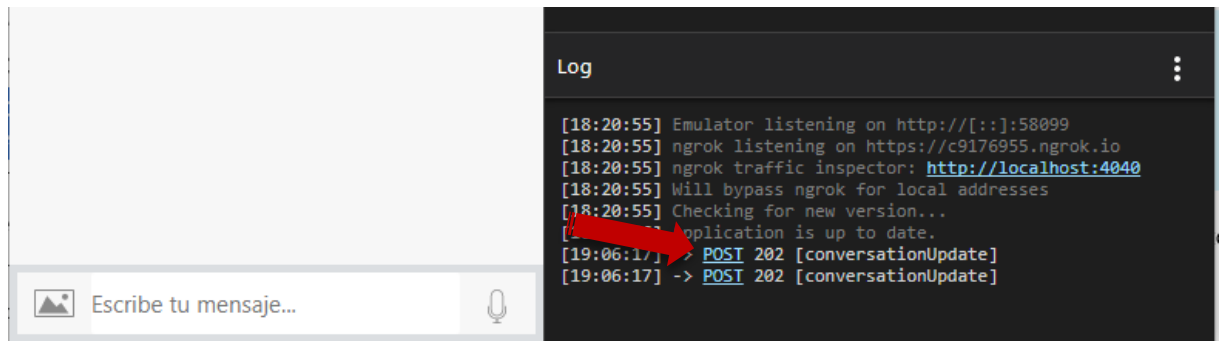
c79e1cf4-4f84-41b3-a0d5-fdd628d03a18

Microsoft App Password:

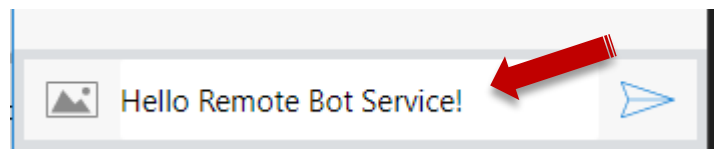
Locale:

CONNECT

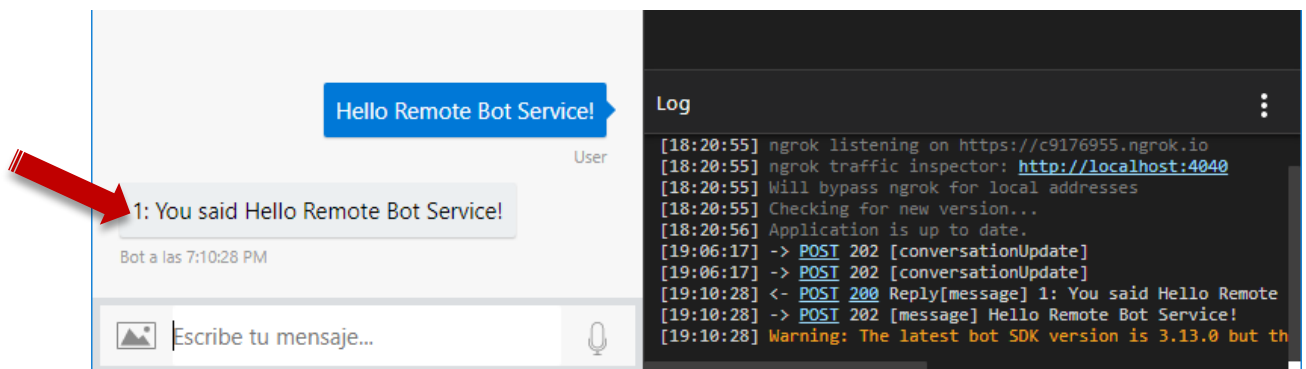
El panel **Details** mostrará dos mensajes **POST** enviados a tu bot.



18. Escribe algún texto para probar la funcionalidad de tu bot y presiona **<enter>**.



Podrás ver la respuesta de tu bot.

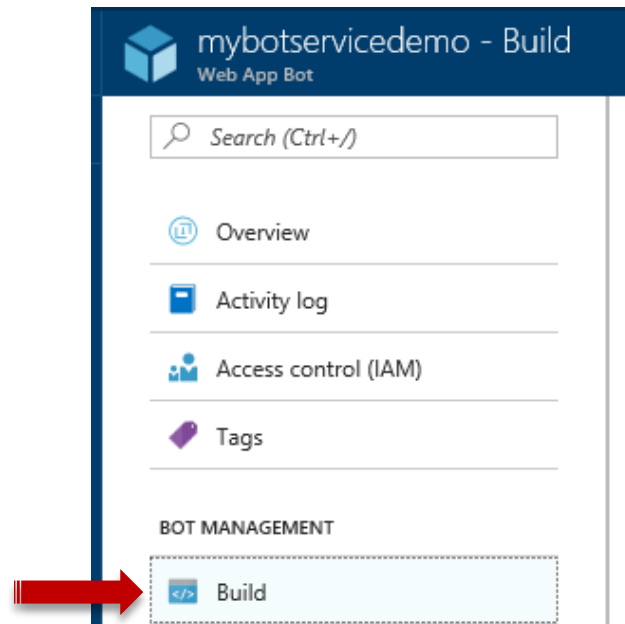


Tarea 4. Editar el bot con el editor en línea.

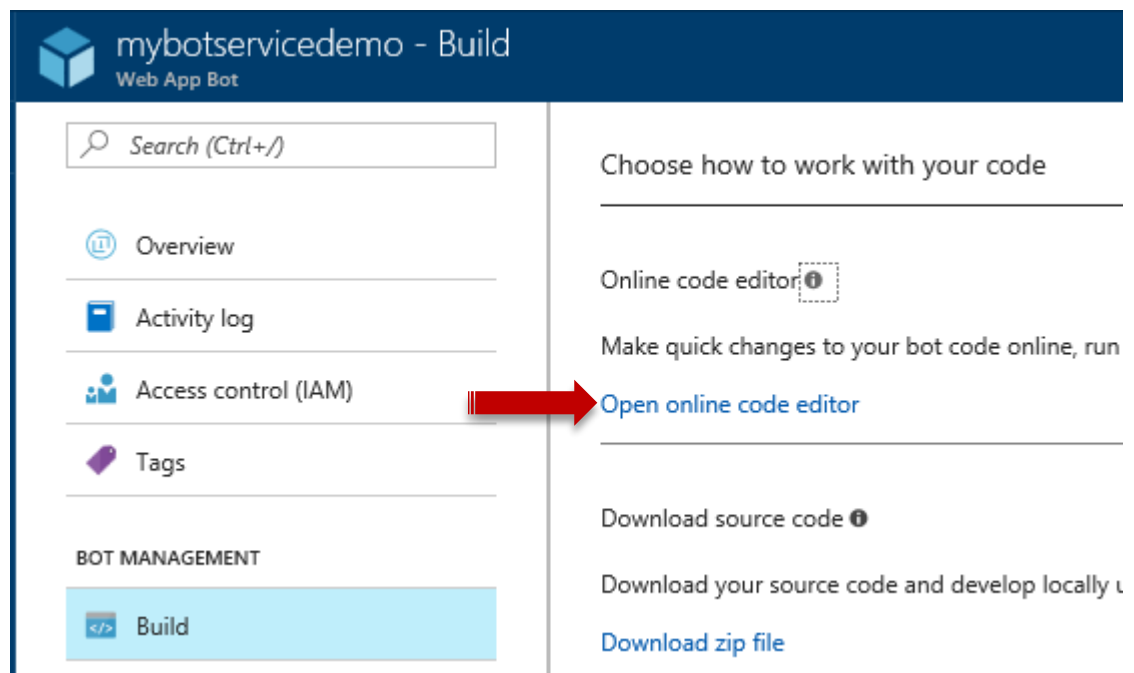
Es posible utilizar el editor de código en línea para modificar y compilar el bot sin la necesidad de un IDE.

En esta tarea mostraremos la forma de modificar el código del bot utilizando el editor de código en línea.

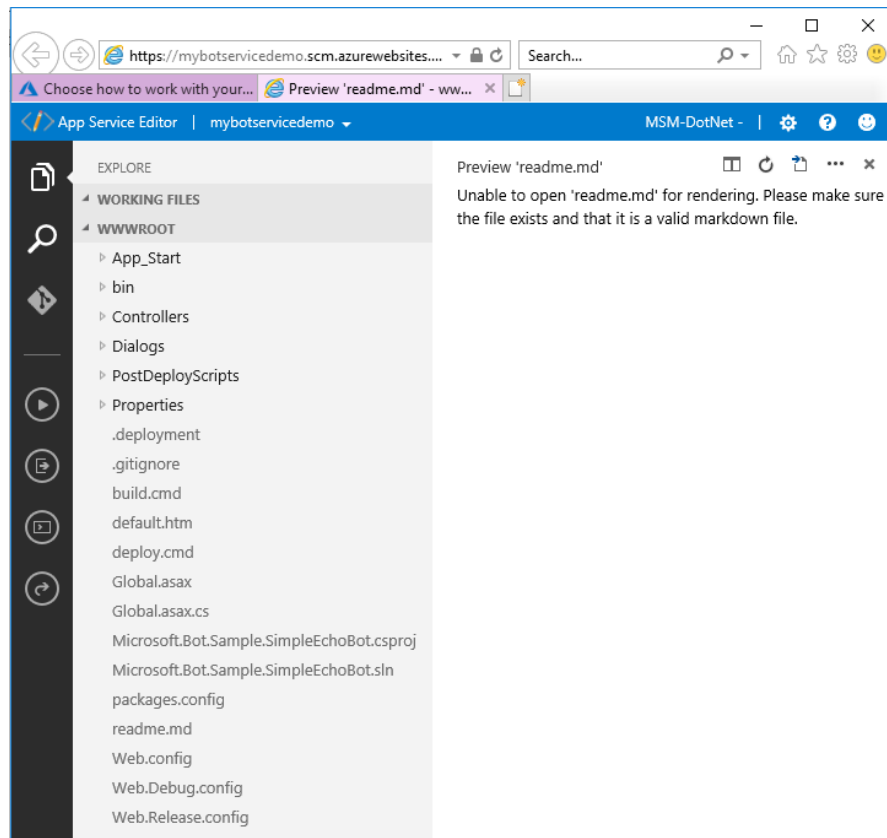
1. Selecciona la opción **Build** desde la hoja de recursos de tu bot en el portal de Azure.



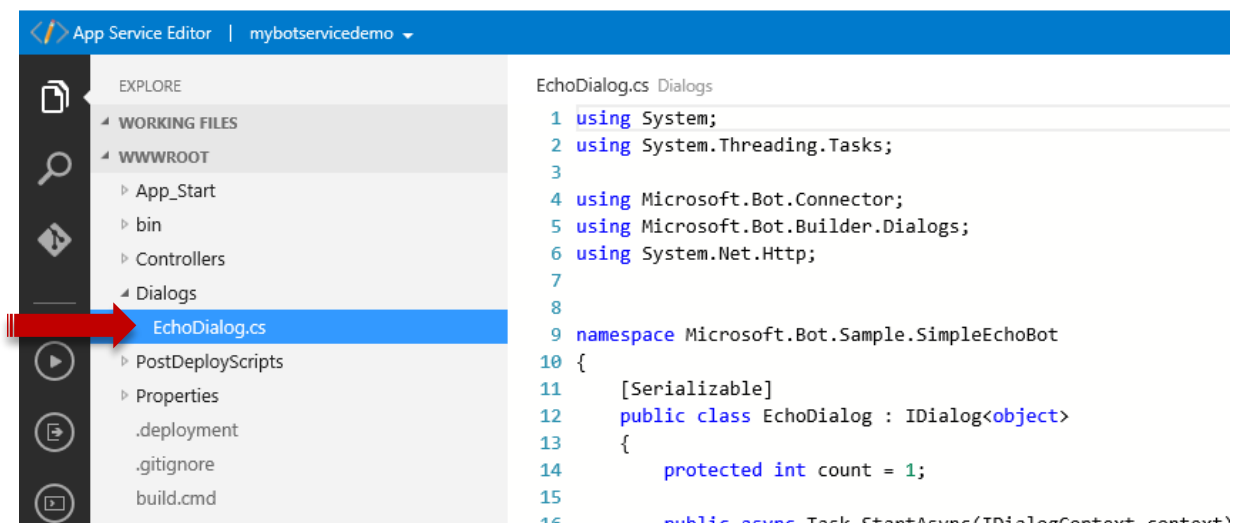
2. Haz clic en el enlace ***Open online code editor***.



Esto abrirá el código del bot en una nueva pestaña del navegador.



- Haz clic sobre el archivo **EchoDialog.cs**. En el panel de la derecha podrás ver el código fuente de la clase **EchoDialog**.



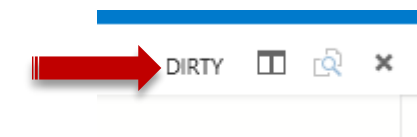
- Modifica el código de la línea 36 para que en lugar de mostrar el texto **"You said"** muestre el texto **"Tú escribiste"**.

```
--
36 |      await context.PostAsync($"{{this.count++}}: You said {message.Text}");
37 |      context.Wait(MessageReceivedAsync);
```

Después de modificar, el código será similar al siguiente.

```
--
36 |      await context.PostAsync($"{{this.count++}}: Tú escribiste {message.Text}");
37 |      context.Wait(MessageReceivedAsync);
```

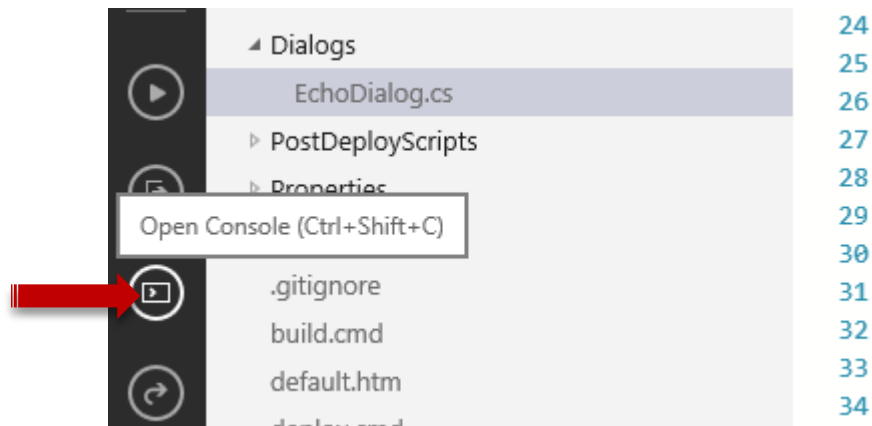
El editor en línea guarda los cambios automáticamente y lo puedes notar con el botón indicador del extremo superior derecho. Mientras hay cambios pendientes de guardar muestra el texto **DIRTY**.



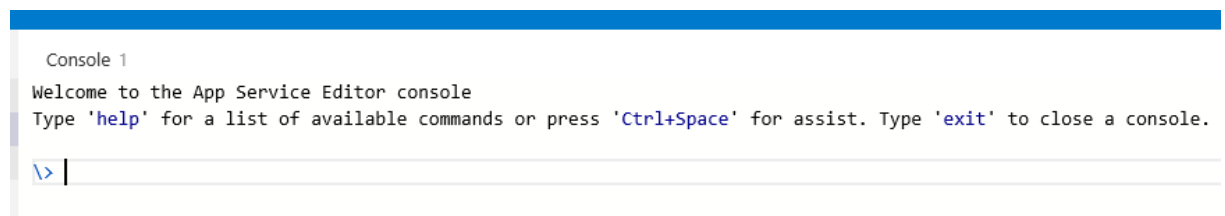
Cuando los cambios son salvados muestra el texto **SAVED**.



5. Haz clic sobre el icono **Open Console** para abrir la consola.



El panel de la derecha mostrará la consola.



6. Escribe el comando **build.cmd** y presiona <enter> para compilar y publicar el bot.

```
Console 1
Welcome to the App Service Editor console
Type 'help' for a list of available command

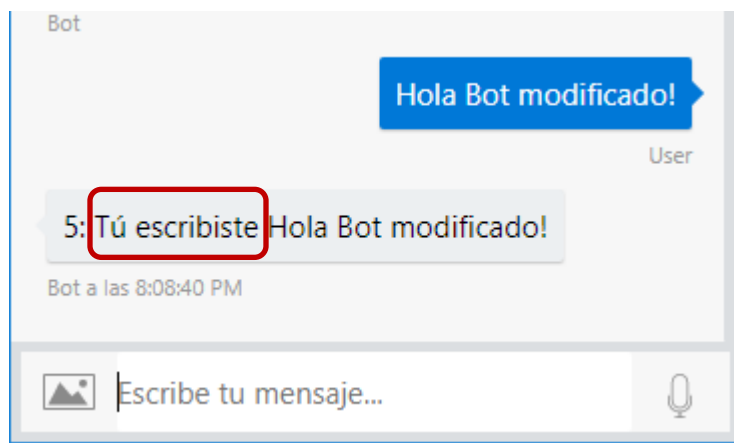
\> build.cmd
```

Al finalizar el proceso podrás ver un mensaje similar al siguiente.

```
Feeds used:
  https://api.nuget.org/v3/index.json

Installed:
  29 package(s) to packages.config projects
  Microsoft.Bot.Sample.SimpleEchoBot -> D:\home\site\wwwroot\bin\SimpleEchoBot.dll
Finished successfully.
\> |
```

7. Regresa al **Bot Framework Emulator** y conéctate al bot en caso de que te hayas desconectado.
8. Escribe un mensaje. Puedes ver que el cambio realizado con el editor de código en línea se refleja en la respuesta que envía el bot.



Resumen

En este laboratorio crearemos un Chatbot utilizando **Bot Builder SDK for .NET** y las herramientas que nos proporciona **Microsoft Bot Framework**. También si cuentas con una suscripción a Microsoft Azure, tuviste la oportunidad de crear un bot utilizando *Azure Bot Service*

Esto apenas es una introducción al desarrollo de Bots con el **Bot Builder** y el **Microsoft Bot Framework**.

En los siguientes laboratorios exploraremos los distintos tipos de mensajes que podemos utilizar en un dialogo para crear una experiencia rica de conversación. También exploraremos la forma de implementar Inteligencia Artificial a nuestros bots y la forma de conectar nuestros bots a los distintos canales como Skype, Facebook, Slack, etc.

¡Continúa tu aprendizaje en el desarrollo de Bots!