



Add a WhatsApp Channel to your Power Virtual Agents Bot with Twilio

[Blog](#) / [Developers](#) / **Add a WhatsApp Channel to your Power...**

February 10, 2023

Written by [KK Gan](#) Twilio

Reviewed by [Miguel Grinberg](#) Twilio [Matthew Setter](#) Twilio

Add a WhatsApp Channel to your Power Virtual Agents Bot with Twilio

Prerequisites

Collect required parameters from Power Virtual Agents

Create a Relay Bot with ASP.NET Core Web API

Run and test the Relay Bot

Configure the Twilio WhatsApp Sandbox with Relay Bot

Tags

IVR and customer care

Code, tutorials, and best practices

Products

[WhatsApp Business API](#)

[Start for free](#)



Add a WhatsApp channel to your Microsoft Power Virtual Agents bot with Twilio

KK Gan
Developer Evangelist, Twilio Inc.

With [Microsoft Power Virtual Agents](#), you can create a bot without writing code. However, if you would like to [add the bot to Azure Bot Service channels](#), you will need to create a **Relay Bot** that acts as a bridge, and this task requires extensive programming knowledge.

This article demonstrates how to create a Relay Bot in C# to connect a bot built with Power Virtual Agents to Twilio Messaging for [SMS](#) or [WhatsApp](#). This exercise assumes that you already have a Power Virtual Agents bot created, and would like to bridge the bot with a WhatsApp channel.

There are four sections in this article:

- Collect required parameters from Power Virtual Agents

• Collect required parameters from Power Virtual Agents

- Create a Relay Bot with ASP.NET Core Web API
- Run and test the Relay Bot
- Configure Twilio Whatsapp Sandbox with Relay Bot

Prerequisites

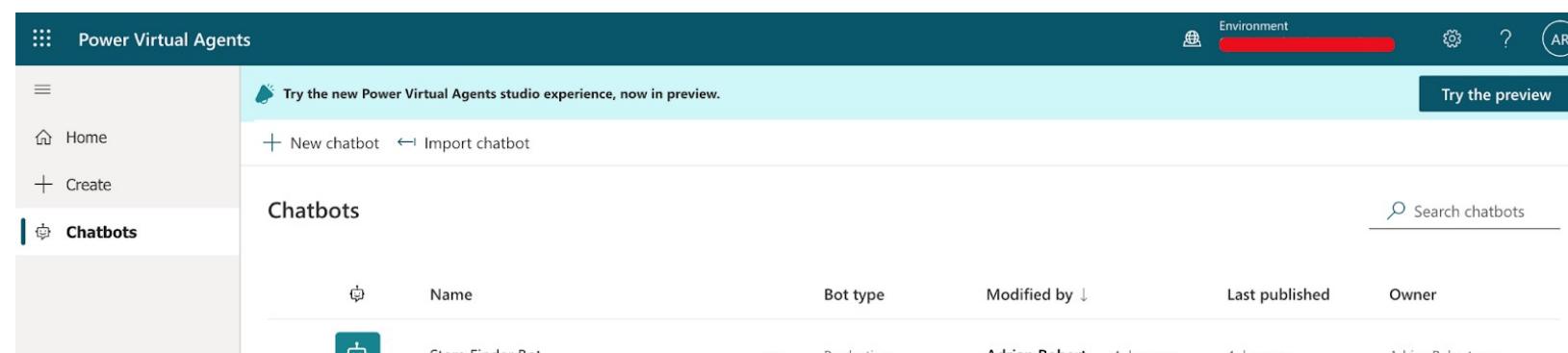
To complete this tutorial, you'll need an account with Twilio, a Power Virtual Agents subscription, a Power Virtual Agents bot created, and Ngrok installed and authenticated. If you have not done so already:

- Sign up for a [free Twilio account](#)
- [Sign up for a Power Virtual Agents trial](#)
- Create a Power Virtual Agents bot (i.e. please refer to [Quickstart - Create and deploy a chatbot](#) on how to create a Power Virtual Agents bot, if you have not done so already)
- Download, install and authenticate [ngrok](#)

Collect required parameters from Power Virtual Agents

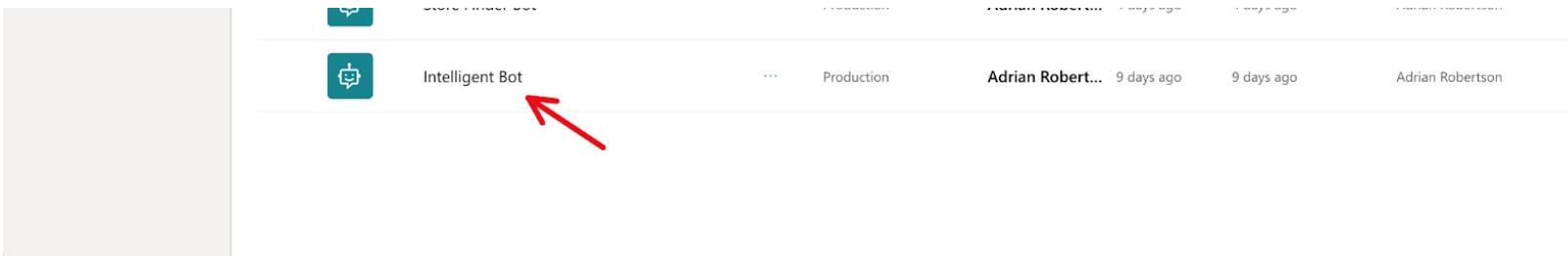
Log in to your [Power Virtual Agents dashboard](#).

Select the *Power Virtual Agents* bot you would like to add a **WhatsApp** channel to.



The screenshot shows the Power Virtual Agents dashboard. The left sidebar has 'Chatbots' selected. The main area displays a table of chatbots with columns for Name, Bot type, Modified by, Last published, and Owner. One row is visible for 'Store Finder Bot'. A banner at the top says 'Try the new Power Virtual Agents studio experience, now in preview.' and includes a 'Try the preview' button.

Name	Bot type	Modified by	Last published	Owner
Store Finder Bot	Production	Adrian Robertson	4 days ago	Adrian Robertson

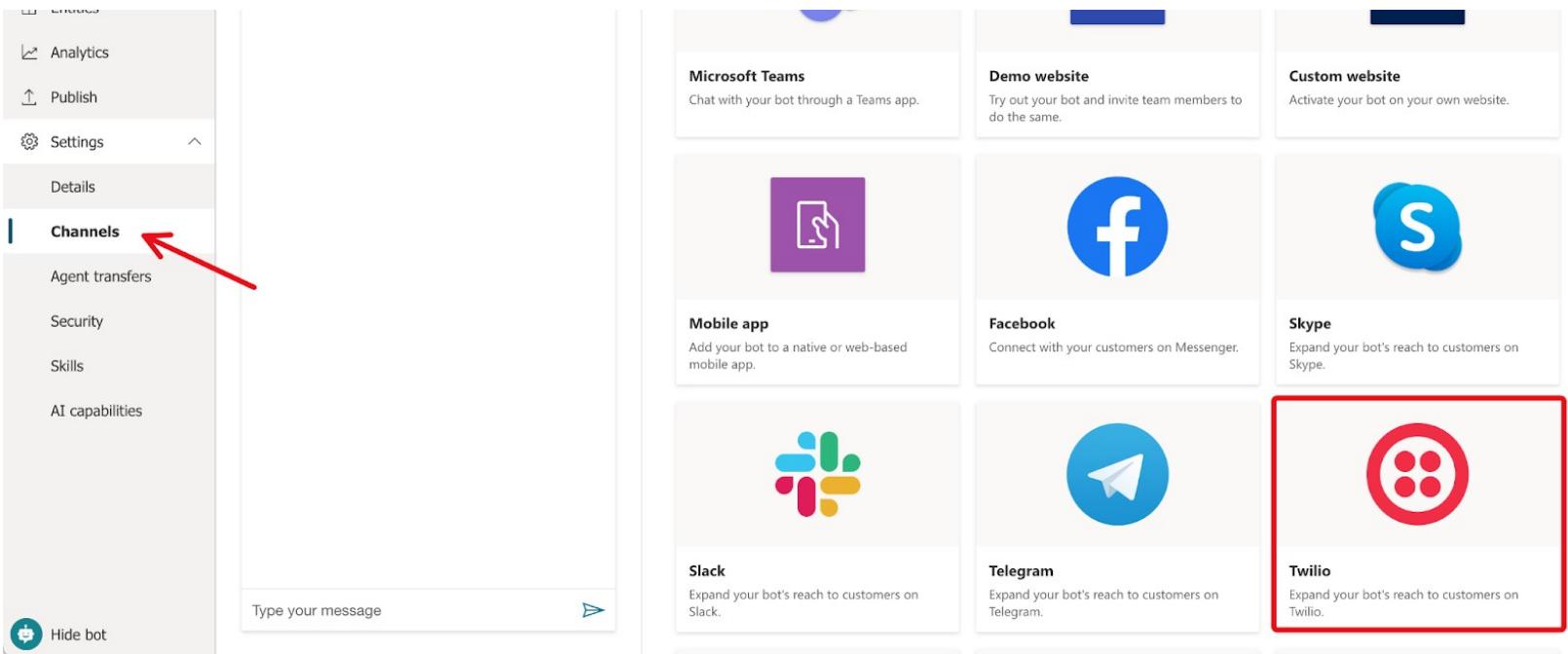


Select **Details** from the **Settings** menu of the selected bot. Then, copy the **Tenant ID** and **Bot app ID** from the bot details page as highlighted in the screenshot below. Save the values for later use.

A screenshot of the 'Power Virtual Agents' application showing the 'Test bot' details page. On the left, there's a sidebar with options like Chatbots, Overview, Topics, Entities, Analytics, Publish, Settings, and Details. Under 'Details', there are sections for Channels, Agent transfers, Security, Skills, and AI capabilities. The main area shows a preview of the bot named 'Test bot' with a 'Chat' tab selected. On the right, there's a 'Details' section with fields for Name (set to 'Intelligent Bot'), Icon (a blue robot icon), and Metadata. Below these are buttons for Environment ID (highlighted with a red arrow), Tenant ID (highlighted with a red arrow), Bot app ID (highlighted with a red arrow), and Download bot manifests. At the bottom, there are buttons for Test manifest and Published manifest.

Go to the **Channels** section of the bot's settings and select **Twilio**, as shown below.

A screenshot of the 'Power Virtual Agents' application showing the 'Test bot' channels page. The left sidebar has the same options as the previous screenshot. The main area is titled 'Channels' with the sub-instruction 'Configure your bot channels to meet your customers where they are. Learn more about channels'. Below this, there are three channel icons: Microsoft Teams (blue and white), Microsoft Bot Framework (blue globe), and Twilio (blue globe with a phone receiver). The Twilio icon is the one being selected.



Copy and save the **Token Endpoint** value shown for the Twilio channel for later use.

This screenshot shows the Power Virtual Agents interface with a 'Test bot' window open. The sidebar on the left includes 'Chatbots', 'Overview', 'Topics', 'Entities', 'Analytics', 'Publish', 'Settings', 'Details', and 'Channels'. The 'Channels' section is currently selected. In the main area, there's a 'Twilio' configuration panel with instructions to engage customers on the channel. A 'Token Endpoint' field is displayed, with its value redacted and a red arrow pointing to a 'Copy' button.

Create a Relay Bot with ASP.NET Core Web API

This section will guide you through creating a Relay Bot with ASP.NET Core in C#. The following prerequisites are needed:

- [Visual Studio Code](#)
- [C# for Visual Studio Code \(latest version\)](#)
- [.NET 7.0 SDK](#)

The project and code that we are going to create in the following steps can be found in the [BotConnectorAPI GitHub repository](#).

If you do not want to create the project from scratch, you can clone the repository, set the required bot parameters that you collected from the previous section in the project's *appsettings.json* file, and run the project directly.

If you choose to clone the project, you may skip this section and jump straight to the next section to **Run and test a Relay Bot project**.

If you prefer to create the project from scratch, the following instructions will guide you step by step on how to do so.

It is important to ensure that you have the right version of .NET. Verify the .NET SDK and version with the `dotnet -list-sdks` and `dotnet --version` commands. The sample output from these commands is shown below.

Bash

```
1 | (base) kogan@WV4F9DM7Q0 azure % dotnet --list-sdks
2 | 2.1.818 [/usr/local/share/dotnet/sdk]
3 | 6.0.400 [/usr/local/share/dotnet/sdk]
4 | 6.0.402 [/usr/local/share/dotnet/sdk]
5 | 7.0.102 [/usr/local/share/dotnet/sdk]
6 | (base) kogan@WV4F9DM7Q0 azure %
7 | (base) kogan@WV4F9DM7Q0 azure % dotnet --version
8 | 7.0.102
```

Use the command `dotnet new webapi -o myBotConnector` to create a new .NET Core Web API project.

Bash

```
1 | (base) kogan@WV4F9DM7Q0 azure % dotnet new webapi -o myBotConnector
2 | The template "ASP.NET Core Web API" was created successfully.
3 |
4 | Processing post-creation actions...
5 | Restoring /Users/kogan/git/azure/myBotConnector/myBotConnector.csproj:
6 |   Determining projects to restore...
7 |     Restored /Users/kogan/git/azure/myBotConnector/myBotConnector.csproj (in 145 ms).
8 | Restore succeeded.
```

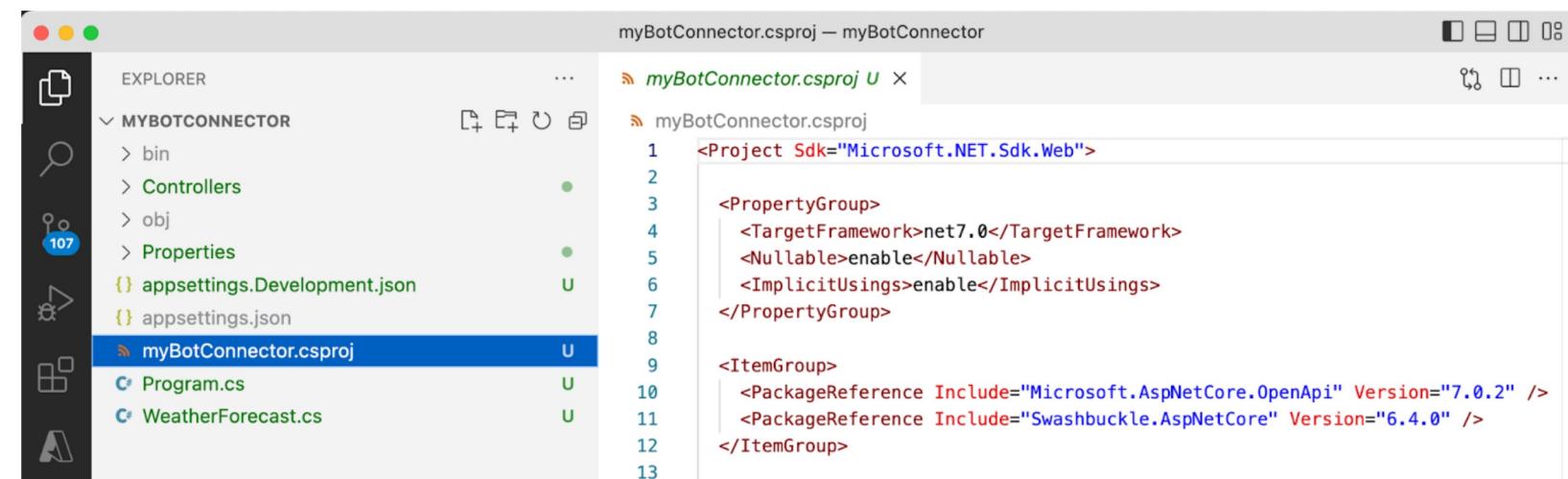
Once completed, change into the project folder and open the folder with Visual Studio Code.

Bash

```
1 | (base) kogan@WV4F9DM7Q0 azure % cd myBot*
2 | (base) kogan@WV4F9DM7Q0 myBotConnector % code .
```

Visual Studio Code will open the project with the folder where the `code .` command was executed.

The screenshot below shows how the project folder structure will look.





```
14    </Project>
15
```

Open the `myBotConnector.csproj` file, and you will notice that two packages have been installed by default:

XML

```
1 <ItemGroup>
2     <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.2" />
3     <PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />
4 </ItemGroup>
```

We now need to install the `Microsoft.Rest.ClientRuntime` and `Microsoft.Bot.Connector.DirectLine` packages manually. Run the `dotnet` commands below from your terminal to install these packages:

Bash

```
1 dotnet add package Microsoft.Rest.ClientRuntime -version 2.3.24  
2 dotnet add package Microsoft.Bot.Connector.DirectLine
```

You can verify that the packages were added to our project file as shown below.

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="7.0.2" />
    <PackageReference Include="Microsoft.Bot.Connector.DirectLine" Version="3.0.2" /> Red Box
    <PackageReference Include="Microsoft.Rest.ClientRuntime" Version="2.3.24" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.4.0" />
  </ItemGroup>
</Project>
```

The `dotnet new webapi -o myBotConnector` command created our project with default `WeatherForecast.cs` and `Controllers\WeatherForecastController.cs` files.

I would recommend we delete the unwanted `WeatherForecast.cs` file, clean up the unwanted code inside the `WeatherForecastController.cs` and rename the `WeatherForecastController.cs` to `myBotConnector.cs` as shown below.

WeatherForecastController.cs — myBotConnector

```
namespace myBotConnector.Controllers;
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
```

EXPLORER

- bin
- Controllers
- WeatherForecastController.cs
- obj
- Properties
- appsettings.Development.json
- appsettings.json
- myBotConnector.csproj
- Program.cs
- WeatherForecast.cs

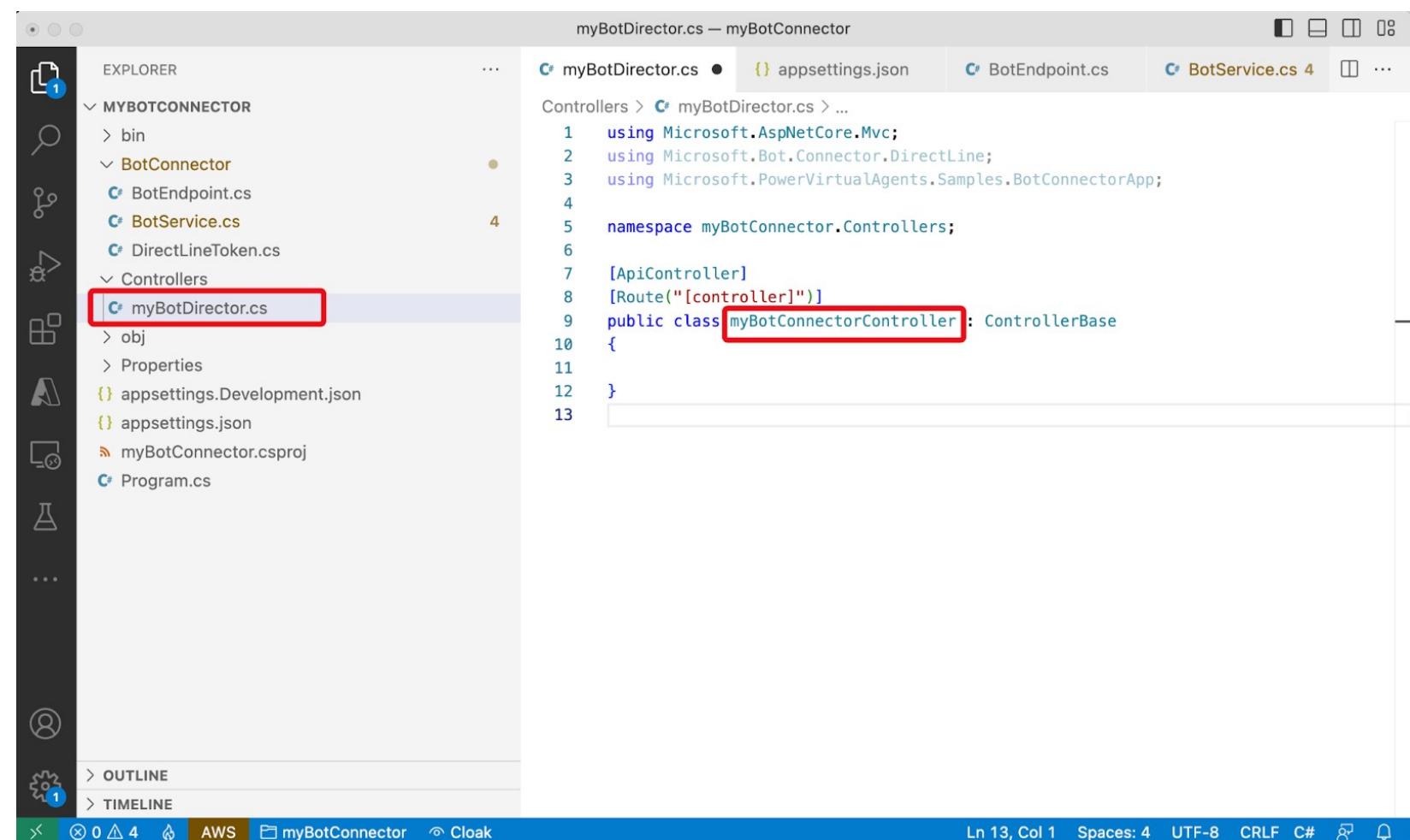
Delete this file

Rename these

Remove the code here..

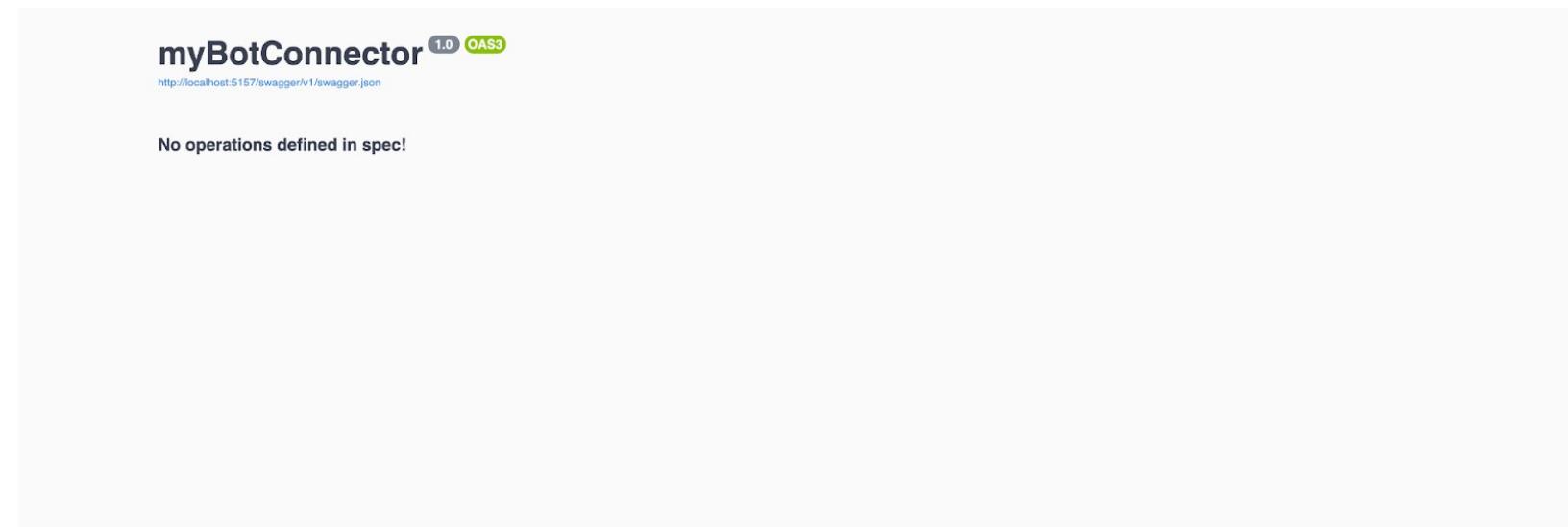


Your project folder should look like the below screenshot.

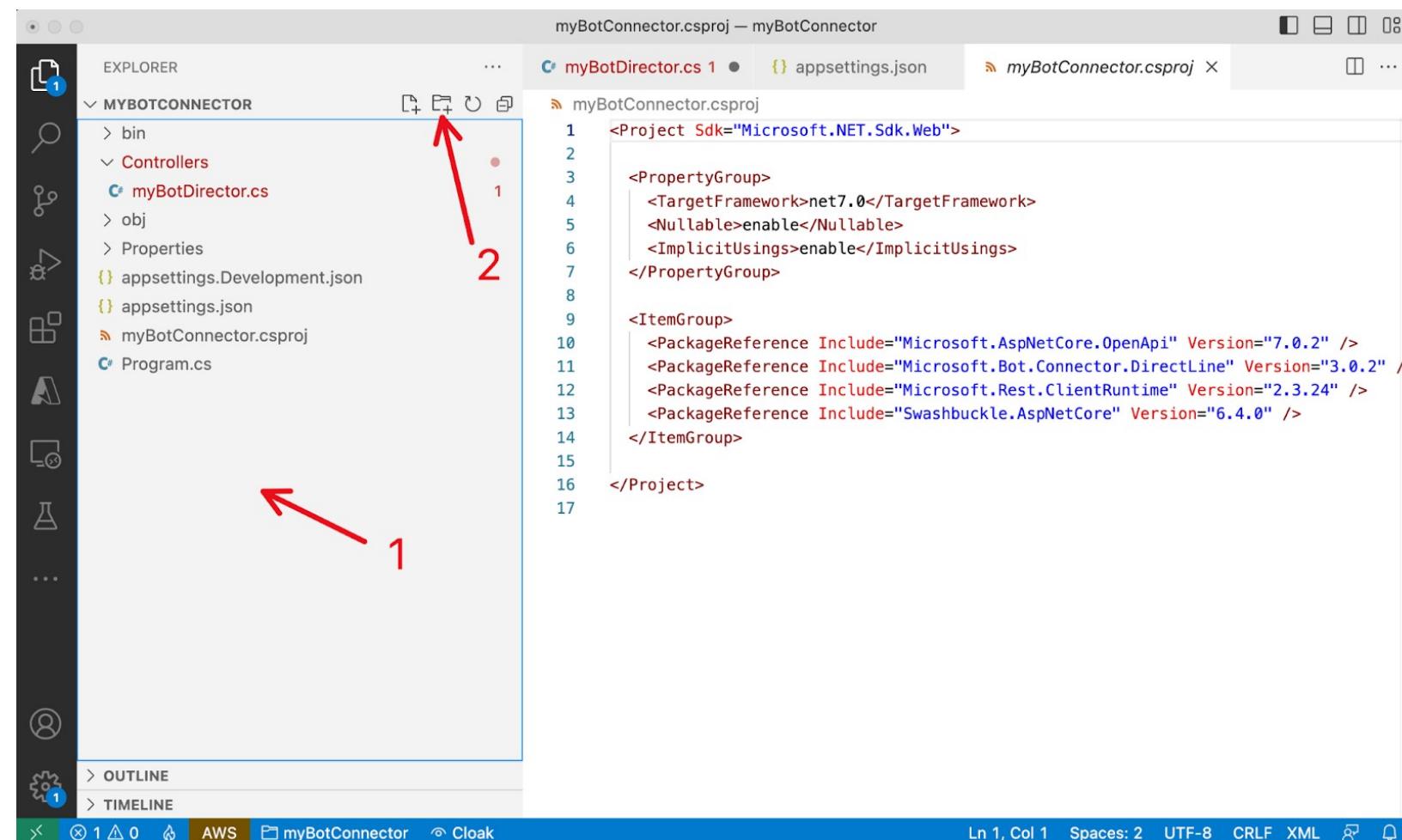


Run the project with the `dotnet watch run` command. The documentation page should open, stating that "**No operations defined in spec!**", as shown below.





Back on Visual Studio Code, click the **Explorer** pane and select “New Folder” to create a new folder. Call the folder *BotConnector*.



Add the following three files for the classes under the new *BotConnector* folder:

1. BotEndpoint.cs

C#

```
1 // Copyright (c) Microsoft Corporation. All rights reserved.
2 // Licensed under the MIT License.
3 
4 using System;
5 
6 namespace Microsoft.PowerVirtualAgents.Samples.BotConnectorApp
7 {
8     /// <summary>
9     /// class with bot info
10    /// </summary>
11    public class BotEndpoint
12    {
13        /// <summary>
14        /// constructor
15        /// </summary>
16        /// <param name="botId">Bot Id GUID</param>
17        /// <param name="tenantId">Bot tenant GUID</param>
18        /// <param name="tokenEndPoint">REST API endpoint to retreive directline token</pa
19        public BotEndpoint(string botId, string tenantId, string tokenEndPoint)
20        {
21            BotId = botId;
22            TenantId = tenantId;
23            UriBuilder uriBuilder = new UriBuilder(tokenEndPoint);
24            uriBuilder.Query = $"botId={BotId}&tenantId={TenantId}";
25            TokenUrl = uriBuilder.Uri;
26        }
27
28        public string BotId { get; }
29
30        public string TenantId { get; }
31
32        public Uri TokenUrl { get; }
33    }
34}
```

2. BotService.cs

C#

```
1 // Copyright (c) Microsoft Corporation. All rights reserved.
2 // Licensed under the MIT License.
3
4 using Microsoft.Rest.Serialization;
5 using System;
6 using System.Net.Http;
7 using System.Threading.Tasks;
8
9 namespace Microsoft.PowerVirtualAgents.Samples.BotConnectorApp
10 {
11     /// <summary>
12     /// Bot Service class to interact with bot
13     /// </summary>
14     public class BotService
15     {
16         private static readonly HttpClient s_httpClient = new HttpClient();
17
18         public string BotName { get; set; }
19
20         public string BotId { get; set; }
21
22         public string TenantId { get; set; }
23
24         public string TokenEndPoint { get; set; }
25
26         /// <summary>
27         /// Get directline token for connecting bot
28         /// </summary>
29         /// <returns>directline token as string</returns>
30         public async Task<string> GetTokenAsync()
31         {
32             string token;
33             using (var httpRequest = new HttpRequestMessage())
34             {
35                 httpRequest.Method = HttpMethod.Get;
36                 UriBuilder uriBuilder = new UriBuilder(TokenEndPoint);
37                 uriBuilder.Query = $"api-version=2022-03-01-preview&botId={BotId}&tenantId={TenantId}";
38                 httpRequest.RequestUri = uriBuilder.Uri;
39                 using (var response = await s_httpClient.SendAsync(httpRequest))
40                 {
41                     var responseString = await response.Content.ReadAsStringAsync();
42                 }
43             }
44         }
45 }
```

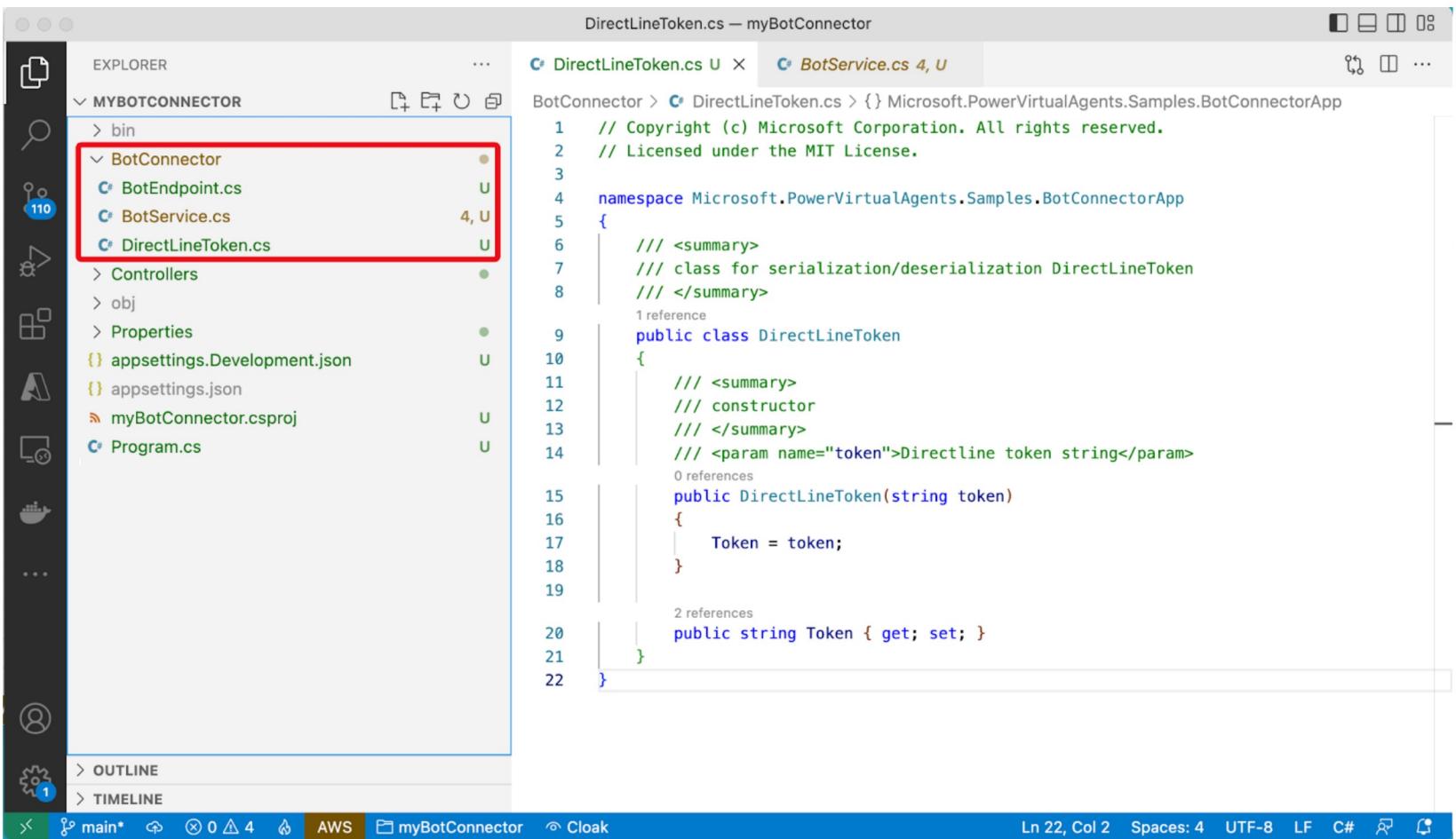
```
41         var responseString = await response.Content.ReadAsStringAsync(),
42         token = SafeJsonConvert.DeserializeObject<DirectLineToken>(responseStr
43     }
44 }
45
46     return token;
47 }
48 }
49 }
```

3. DirectLineToken.cs

C#

```
1 // Copyright (c) Microsoft Corporation. All rights reserved.
2 // Licensed under the MIT License.
3
4 namespace Microsoft.PowerVirtualAgents.Samples.BotConnectorApp
5 {
6     /// <summary>
7     /// class for serialization/deserialization DirectLineToken
8     /// </summary>
9     public class DirectLineToken
10    {
11        /// <summary>
12        /// constructor
13        /// </summary>
14        /// <param name="token">Directline token string</param>
15        public DirectLineToken(string token)
16        {
17            Token = token;
18        }
19
20        public string Token { get; set; }
21    }
22}
```

The project folder should now look like the screenshot below.



The screenshot shows the Visual Studio IDE interface. The Explorer pane on the left displays the project structure of 'MYBOTCONNECTOR'. The 'BotConnector' folder contains four files: BotEndpoint.cs, BotService.cs, DirectLineToken.cs, and Program.cs. The DirectLineToken.cs file is selected and highlighted with a red box. The code editor on the right shows the content of DirectLineToken.cs:

```

1 // Copyright (c) Microsoft Corporation. All rights reserved.
2 // Licensed under the MIT License.
3 
4 namespace Microsoft.PowerVirtualAgents.Samples.BotConnectorApp
5 {
6     /// <summary>
7     /// class for serialization/deserialization DirectLineToken
8     /// </summary>
9     public class DirectLineToken
10    {
11        /// <summary>
12        /// constructor
13        /// </summary>
14        /// <param name="token">Directline token string</param>
15        public DirectLineToken(string token)
16        {
17            Token = token;
18        }
19
20        public string Token { get; set; }
21    }
22 }

```

The status bar at the bottom indicates the code is in C# and shows the file path as 'Ln 22, Col 2 Spaces: 4 UTF-8 LF C#'. The tab bar shows 'myBotConnector' is the active project.

Replace the content of *myBotConnector.cs* with the below code.

C#

```

1 using Microsoft.AspNetCore.Mvc;
2 using Microsoft.Bot.Connector.DirectLine;
3 using Microsoft.PowerVirtualAgents.Samples.BotConnectorApp;
4 
5 namespace myBotConnector.Controllers;
6 
7 [ApiController]
8 [Route("[controller]")]
9 public class myBotConnectorController : ControllerBase
10 {
11     private readonly IConfiguration _configuration;
12     private static string? _watermark = null;
13     private const int _botReplyWaitIntervalInMilSec = 3000;
14     private const string _botDisplayName = "Bot";
15     private const string _userDisplayName = "You".

```

```

15     private const string _userAgentHeaderValue = "you ";
16     private static string? s_endConversationMessage;
17     private static BotService? s_botService;
18     public static IDictionary<string, string> s_tokens = new Dictionary<string, string>();
19     public myBotConnectorController(IConfiguration configuration)
20     {
21         _configuration = configuration;
22         var botId = _configuration.GetValue<string>("BotId") ?? string.Empty;
23         var tenantId = _configuration.GetValue<string>("BotTenantId") ?? string.Empty;
24         var botTokenEndpoint = _configuration.GetValue<string>("BotTokenEndpoint") ?? string.Empty;
25         var botName = _configuration.GetValue<string>("BotName") ?? string.Empty;
26         s_botService = new BotService()
27         {
28             BotName = botName,
29             BotId = botId,
30             TenantId = tenantId,
31             TokenEndPoint = botTokenEndpoint,
32         };
33         s_endConversationMessage = _configuration.GetValue<string>("EndConversationMessage");
34         if (string.IsNullOrEmpty(botId) || string.IsNullOrEmpty(tenantId) || string.IsNullOrEmpty(botName))
35         {
36             Console.WriteLine("Update App.config and start again.");
37             Console.WriteLine("Press any key to exit");
38             Console.Read();
39             Environment.Exit(0);
40         }
41     }
42
43     [HttpPost]
44     [Route("StartBot")]
45     [Consumes("application/x-www-form-urlencoded")]
46     //public async Task<ActionResult> StartBot(HttpContext req)
47     public async Task<ActionResult> StartBot([FromForm] string From, [FromForm] string Body)
48     {
49         Console.WriteLine("From: " + From + ", " + Body);
50         var token = await s_botService.GetTokenAsync();
51         if (!s_tokens.ContainsKey(From)) {
52             s_tokens.Add(From, token);
53         }
54         Console.WriteLine("s_tokens: " + s_tokens[From]);
55         var response = await StartConversation(Body, s_tokens[From]);
56
57         return Ok(response);
58     }
59
60     //private static async Task<string> StartConversation(string inputMsg)

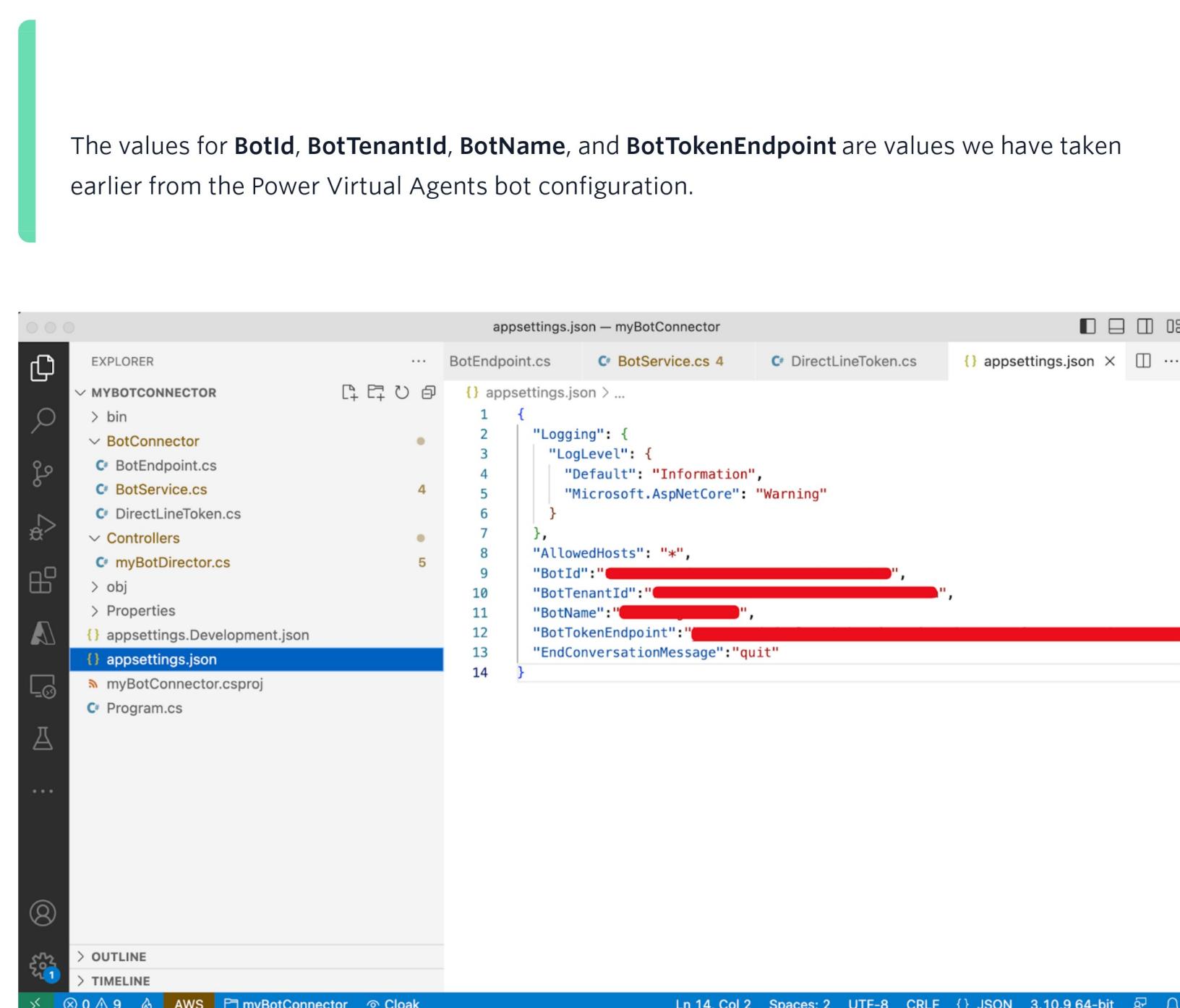
```

```
61     private async Task<string> StartConversation(string inputMsg, string token = "")  
62     {  
63         Console.WriteLine("token: " + token);  
64         using (var directLineClient = new DirectLineClient(token))  
65         {  
66             var conversation = await directLineClient.Conversations.StartConversationAsync();  
67             var conversationId = conversation.ConversationId;  
68             //string inputMessage;  
69  
70             Console.WriteLine(conversationId + ": " + inputMsg);  
71             //while (!string.Equals(inputMessage, s_endConversationMessage, StringComparison.OrdinalIgnoreCase))  
72  
73             if (!string.IsNullOrEmpty(inputMsg) && !string.Equals(inputMsg, s_endConversationMessage, StringComparison.OrdinalIgnoreCase))  
74             {  
75                 // Send user message using directlineClient  
76                 await directLineClient.Conversations.PostActivityAsync(conversationId, new Activity  
77                 {  
78                     Type = ActivityTypes.Message,  
79                     From = new ChannelAccount { Id = "userId", Name = "userName" },  
80                     Text = inputMsg,  
81                     TextFormat = "plain",  
82                     Locale = "en-US",  
83                 });  
84  
85                 // Get bot response using directlinClient  
86                 List<Activity> responses = await GetBotResponseActivitiesAsync(directLineClient);  
87                 return BotReplyAsAPIResponse(responses);  
88             }  
89  
90             return "Thank you.";  
91         }  
92     }  
93  
94     private static string BotReplyAsAPIResponse(List<Activity> responses)  
95     {  
96         string responseStr = "";  
97         responses?.ForEach(responseActivity =>  
98         {  
99             // responseActivity is standard Microsoft.Bot.Connector.DirectLine.Activity  
100            // See https://github.com/Microsoft/botframework-sdk/blob/master/specs/botframe  
101            // Showing examples of Text & SuggestedActions in response payload  
102            Console.WriteLine(responseActivity.Text);  
103            if (!string.IsNullOrEmpty(responseActivity.Text))  
104            {  
105                responseStr = responseStr + string.Join(Environment.NewLine, responseActiv  
106            }  
107        }  
108    }  
109
```

```
107         if (responseActivity.SuggestedActions != null && responseActivity.SuggestedAct
108             {
109                 var options = responseActivity.SuggestedActions?.Actions?.Select(a => a.Ti
110                     responseStr = responseStr + $"{string.Join(" | ", options)}";
111                 }
112             }
113         );
114
115         return responseStr;
116     }
117
118 /// <summary>
119 /// Use directlineClient to get bot response
120 /// </summary>
121 /// <returns>List of DirectLine activities</returns>
122 /// <param name="directLineClient">directline client</param>
123 /// <param name="conversationId">current conversation ID</param>
124 /// <param name="botName">name of bot to connect to</param>
125 private static async Task<List<Activity>> GetBotResponseActivitiesAsync(DirectLineClien
126     {
127         ActivitySet response = null;
128         List<Activity> result = new List<Activity>();
129
130         do
131             {
132                 response = await directLineClient.Conversations.GetActivitiesAsync(conversatio
133                 if (response == null)
134                 {
135                     // response can be null if directLineClient token expires
136                     Console.WriteLine("Conversation expired. Press any key to exit.");
137                     Console.Read();
138                     directLineClient.Dispose();
139                     Environment.Exit(0);
140                 }
141
142                 _watermark = response?.Watermark;
143                 result = response?.Activities?.Where(x =>
144                     x.Type == ActivityTypes.Message &&
145                     string.Equals(x.From.Name, s_botService.BotName, StringComparison.OrdinalIgnoreCase)
146
147                     //Console.WriteLine(result);
148                     if (result != null && result.Any())
149                     {
150                         return result;
151                     }
152             }
153         );
154
155         return result;
156     }
157 }
```

```
153
154     Thread.Sleep(1000);
155 } while (response != null && response.Activities.Any());
156
157     return new List<Activity>();
158 }
159 }
```

Update the *appsettings.json* file with the required application settings as shown below.



The values for **BotId**, **BotTenantId**, **BotName**, and **BotTokenEndpoint** are values we have taken earlier from the Power Virtual Agents bot configuration.

```
appsettings.json — myBotConnector
EXPLORER ... BotEndpoint.cs BotService.cs 4 DirectLineToken.cs appsettings.json ...
MYBOTCONNECTOR
  > bin
  > BotConnector
    > BotEndpoint.cs
    > BotService.cs
    > DirectLineToken.cs
  > Controllers
    > myBotDirector.cs
  > obj
  > Properties
  > appsettings.Development.json
  > appsettings.json
  myBotConnector.csproj
  Program.cs

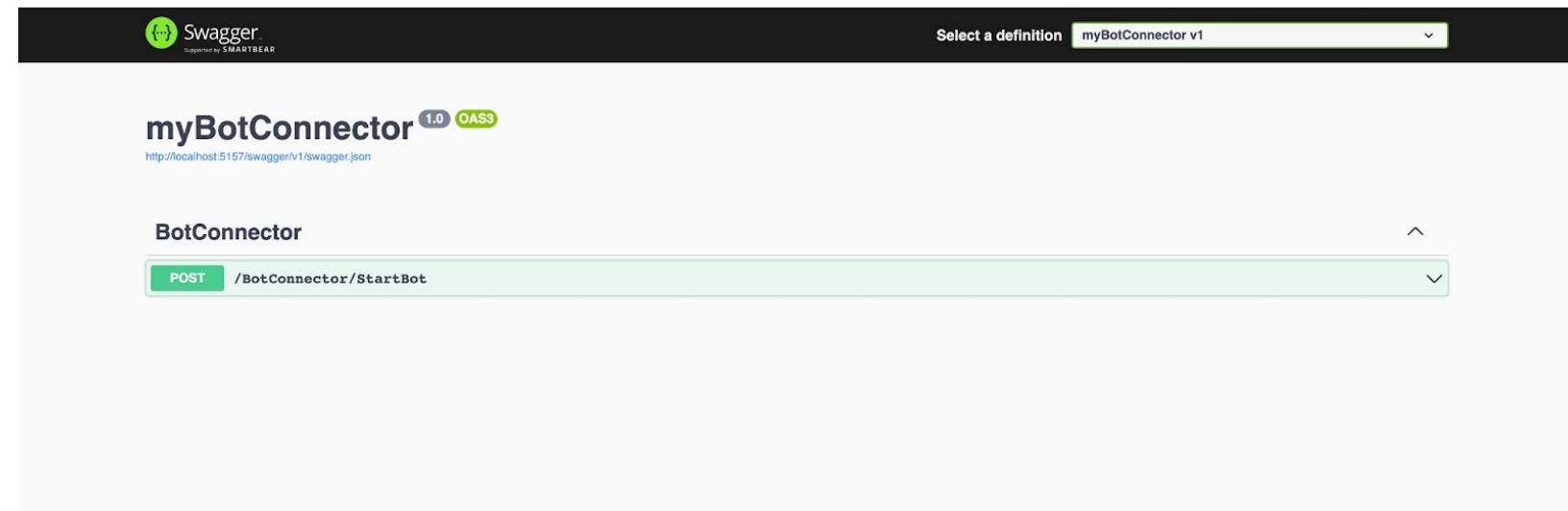
appsettings.json
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft.AspNetCore": "Warning"
6     }
7   },
8   "AllowedHosts": "*",
9   "BotId": "REDACTED",
10  "BotTenantId": "REDACTED",
11  "BotName": "REDACTED",
12  "BotTokenEndpoint": "REDACTED",
13  "EndConversationMessage": "quit"
14 }
```

The BotConnector is now ready to relay messages between a front end client (WhatsApp in our case) and the Power Virtual Agents bot.

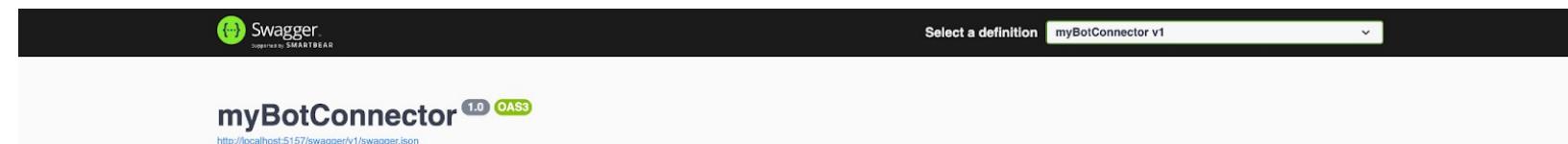
Run and test the Relay Bot

Before you run and test the Relay Bot, please make sure that you have updated the *appsettings.json* file with the values collected from the Power Virtual Agents bot. Please refer to the **Collect required parameters from Power Virtual Agent** section above for details.

Run the project with `dotnet watch run` from the project folder. The project documentation page should now look as follows.



In this page, click on the only endpoint and proceed to test it by supplying the "**From**" and "**Body**" fields with any values as shown in the below screenshot.



The screenshot shows the BotConnector API configuration interface. It's a POST request to the endpoint `/BotConnector/StartBot`. The **Parameters** section shows "No parameters". The **Request body** section has two fields: "From" with value "+6583327738" and "Body" with value "hi there". The **Content-Type** is set to `application/x-www-form-urlencoded`. At the bottom are "Execute" and "Clear" buttons.

Hit the **Execute** button, and you should see the response from the API, as shown below.

The screenshot shows the API response details. It includes:

- Curl:** A terminal command to make the POST request.
- Request URL:** `http://localhost:5157/BotConnector/StartBot`
- Server response:**

Code	Details
200	Response body <pre>Hi! I'm a virtual agent. I can help with account questions, orders, store information, and more. If you'd like to speak to a human agent, let me know at any time. So, what can I help you with today?</pre> <div style="text-align: right;"> Copy Download </div> Response headers <pre>content-type: text/plain; charset=utf-8 date: Mon, 06 Feb 2023 07:18:19 GMT server: Kestrel transfer-encoding: chunked</pre>
- Responses:**

Code	Description	Links
200	Success	No links

The Relay Bot is now ready for the Twilio messaging configuration. Take note of the endpoint path from the Relay Bot documentation page, highlighted in the screenshot below.



Configure the Twilio WhatsApp Sandbox with Relay Bot

Since our project is now running on localhost, we will use **ngrok** to set up a tunnel to expose it to the internet. To do so, start **ngrok** in a separate terminal session with the http port of the project, for example `ngrok http 5157`.

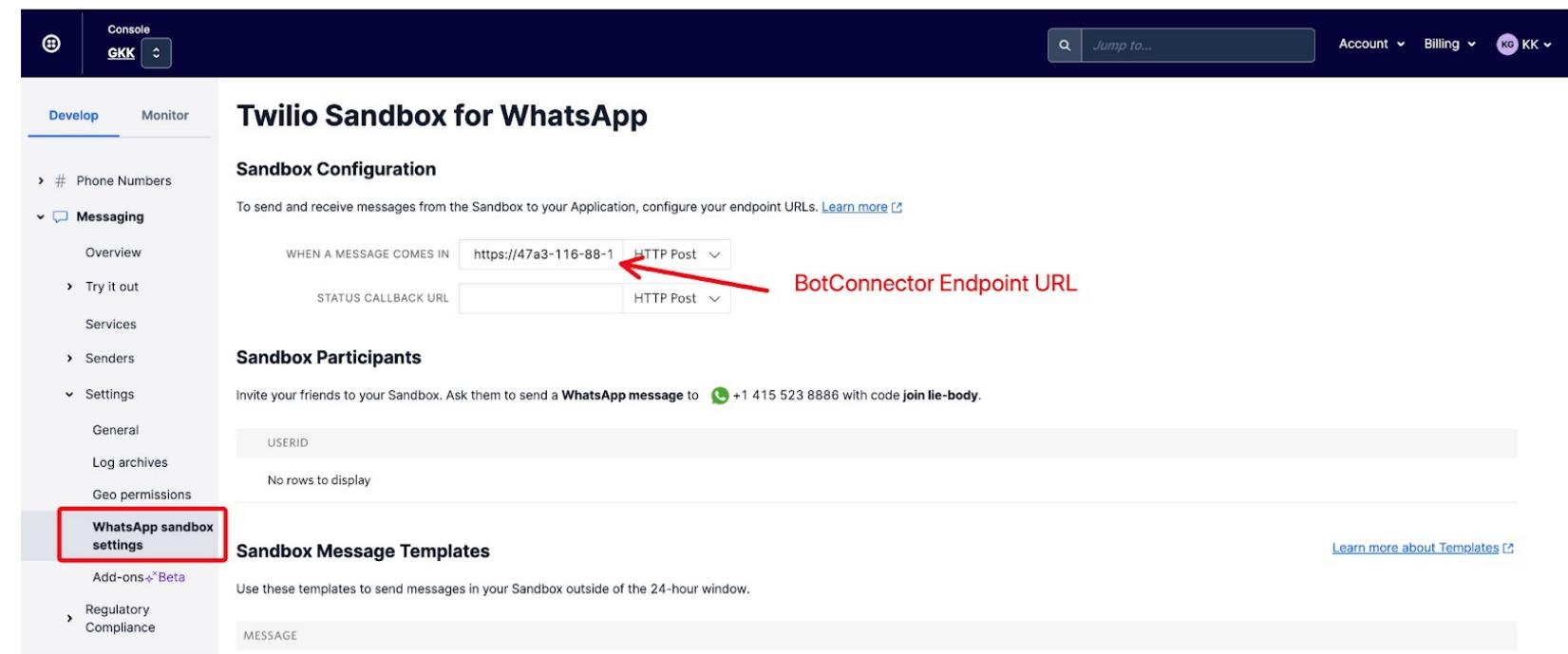
A screenshot of a terminal window titled "kogan — ngrok http 5157 — 109x24". The window displays the output of the ngrok command. It shows the session status as "online" for the "twilio" account, version 3.0.2, in the "Asia Pacific (ap)" region. It also shows the forwarded URL "https://47a3-116-88-10-205.ap.ngrok.io" which maps to "http://localhost:5157".

```
ngrok
Session Status      online
Account            twilio (Plan: Free)
Update             update available (version 3.1.1, Ctrl-U to update)
Version            3.0.2
Region             Asia Pacific (ap)
Latency            20.572583ms
Web Interface     http://127.0.0.1:4040
Forwarding         https://47a3-116-88-10-205.ap.ngrok.io -> http://localhost:5157

Connections        ttl     opn      rt1      rt5      p50      p90
                      0       0       0.00     0.00     0.00     0.00
```

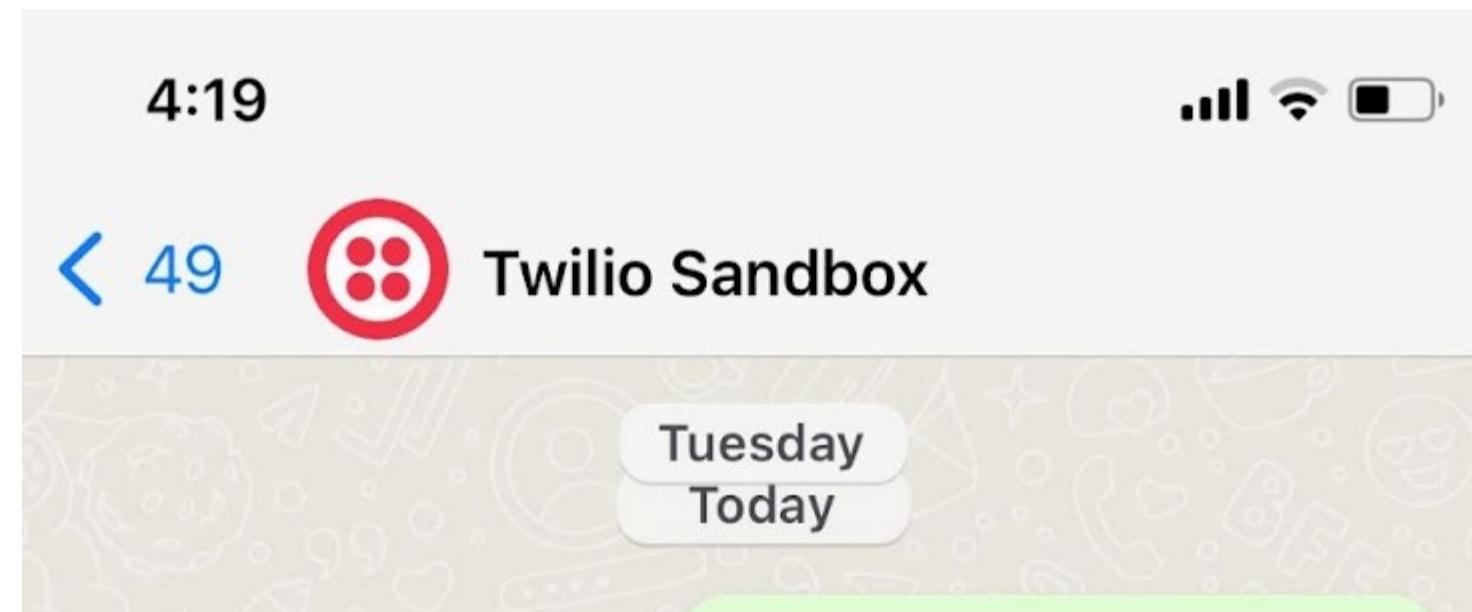
Open the [Twilio console](#) and navigate to the **Messaging - Settings - WhatsApp Sandbox Settings**.

There, enter the full URL for the Relay Bot in the “**When a message comes in**” field. The URL is composed with the ngrok forwarding URL with the Relay Bot’s endpoint added at the end. An example URL should look like <https://47a3-116-88-10-205.ap.ngrok.io/BotConnector/StartBot>.



The screenshot shows the Twilio Sandbox for WhatsApp settings page. On the left, there's a sidebar with 'Develop' selected. Under 'Messaging', 'WhatsApp sandbox settings' is highlighted with a red box. In the main area, under 'Sandbox Configuration', there are two input fields: 'WHEN A MESSAGE COMES IN' containing 'https://47a3-116-88-10-205.ap.ngrok.io/BotConnector/StartBot' and 'STATUS CALLBACK URL' containing 'HTTP Post'. A red arrow points from the text 'BotConnector Endpoint URL' to the first input field.

Save the WhatsApp Sandbox Settings. You can now chat with the Power Virtual Agents bot by initiating a WhatsApp message to your Twilio Sandbox for WhatsApp at the number shown in the **Sandbox Participants** section of the **Twilio Sandbox for WhatsApp** settings page. The below screenshot shows a sample interaction with the Power Virtual Agents bot over WhatsApp.



join lie-body 4:18 PM ✓

Twilio Sandbox: ✅ You are all set! The sandbox can now send/receive messages from whatsapp:[+14155238886](https://api.twilio.com/2010-04-01/messages.json). Reply stop to leave the sandbox any time.

4:18 PM

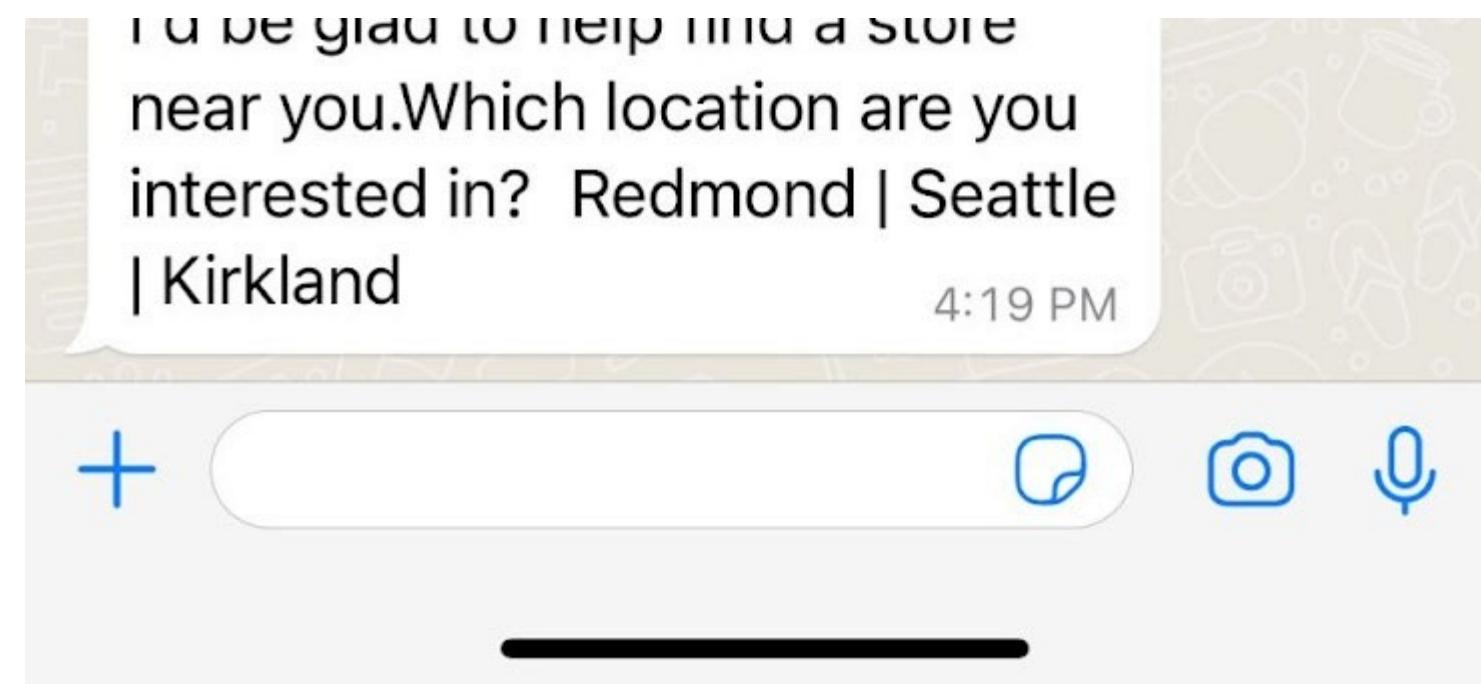
hi there 4:18 PM ✓

Hi! I'm a virtual agent. I can help with account questions, orders, store information, and more. If you'd like to speak to a human agent, let me know at any time. So, what can I help you with today?

4:18 PM

location 4:19 PM ✓

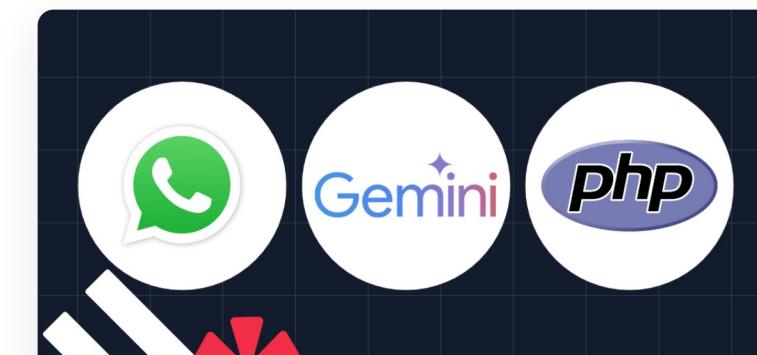
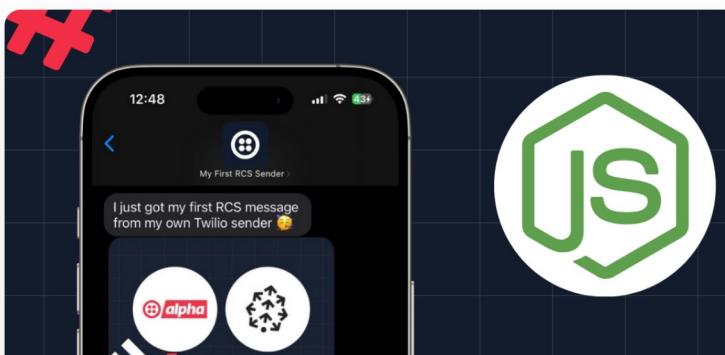
I'd be glad to help find a store



Congratulations! You've now created a Relay Bot, connecting a Power Virtual Agents bot and WhatsApp with Twilio. You can interact with the bot by texting to your WhatsApp enabled Twilio Phone Number. You may explore further on [Formatting, location, and other features in WhatsApp messaging](#) to further enhance your Power Virtual Agents bot in responding with advanced messaging features.

KK Gan is a Developer Evangelist at Twilio. He's been involved in developing and implementing business process automation, communication and collaboration solutions for different clients in the past. He loves and enjoys sharing and learning with the developer communities. He can be reached at linkedin.com/in/kkgan or kogan[at]twilio.com.

Related Posts



Related Resources

How to send an RCS message with

[Twilio Docs](#)

From APIs to SDKs to sample apps

API reference documentation, SDKs, helper libraries, quickstarts, and tutorials for your language and platform.

Build Automations for Utilities from

[Resource Center](#)

The latest ebooks, industry reports, and webinars

Learn from customer engagement experts to improve your own communication.

How to Build a Multimodal WhatsApp

[Ahoy](#)

Twilio's developer community hub

Best practices, code samples, and inspiration to build communications and digital engagement experiences.