Sally
User Guide

Contents

Who is Sally?

Sally is a chatbot developed for Counties Manukau District Health board. Sally can assist you with a wide range of inquiries, covering but not limited to human resources, health and safety, location finding, and project-related FAQs. You can speak with Sally using natural language through the Microsoft Teams interface.

In this guide, we will walk you through the process of building and deploying Sally. As a best practice, please remember to close your conversations with Sally after you are finished. This will maintain a clean interaction history in Teams.

How to build Sally

We will outline 3 different ways in which you can build and deploy Sally using Azure cloud services.

1. Power Virtual Agents
2. Bot Framework SDK via Bot Framework Composer
3. Language Studio

*Of these 3 options, we recommend using **Language studio**.*

The pros and cons of each option are briefly listed as follows:

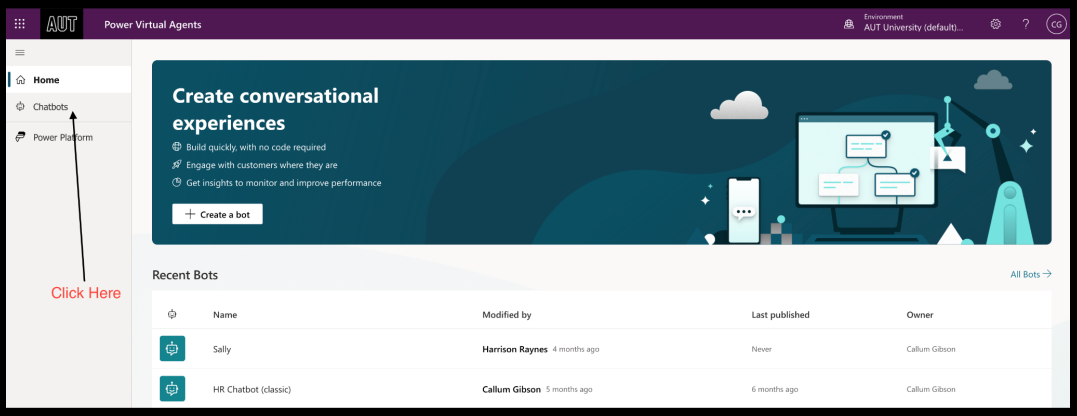
Option	**Pros**	**Cons**
Power Virtual Agents	Easy to use. Low-code. Has natural language processing. Highly customizable Adaptive Cards.	Most expensive.
Bot Framework Composer	Relatively cheap. Highly customizable Adaptive Cards.	Most complicated to implement.
Language Studio	Relatively Cheap. Easy to use. Low-code. Has natural language processing. Easy to import/export knowledge bases.	Marginally less output customization out-of-the-box.

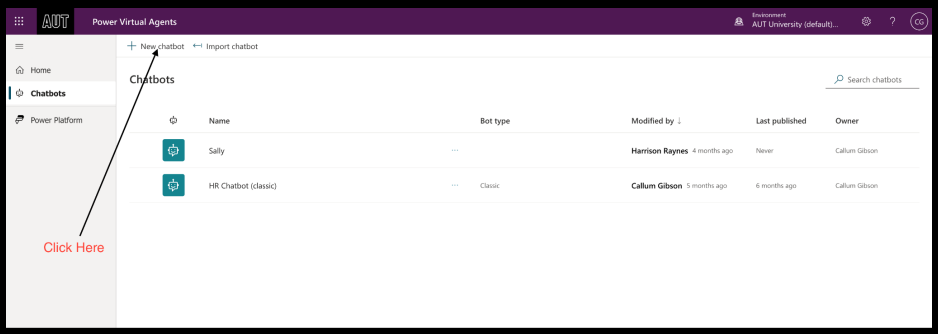
Implementing via. Power Virtual Agents

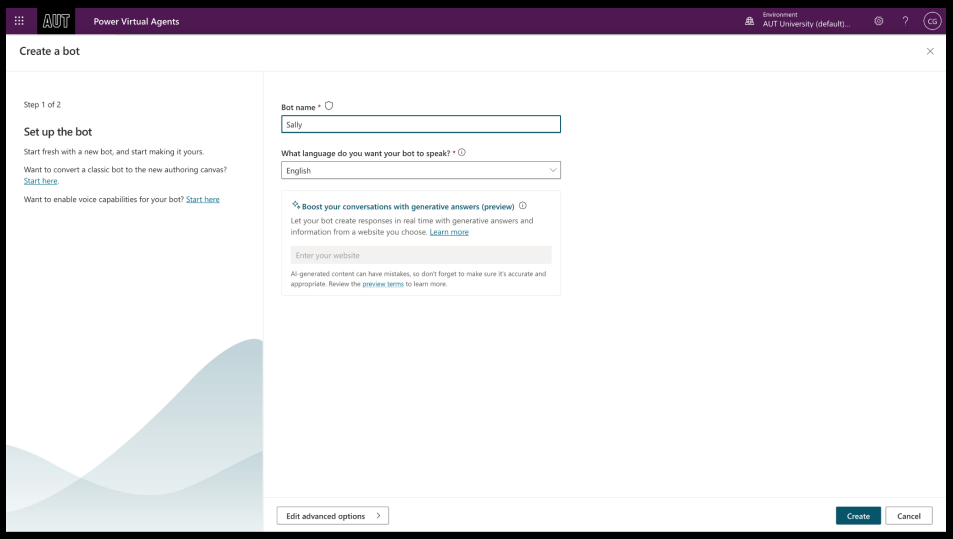
Power Virtual Agents (PVA) is an easy and robust way to build bots capable of natural language processing (NLP) in a visually simple, low-code environment.

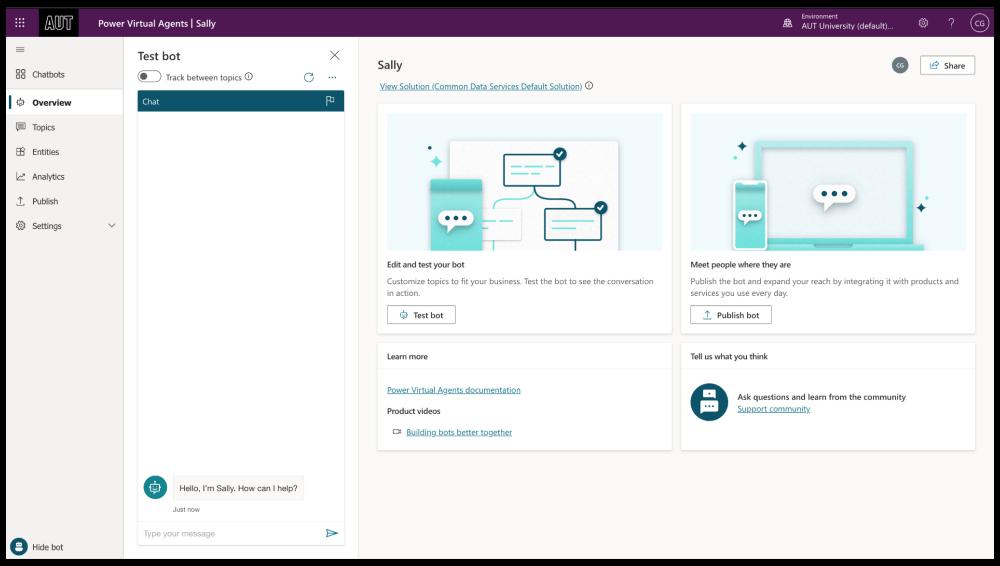
Step-by-step guide to building bots via PVA:

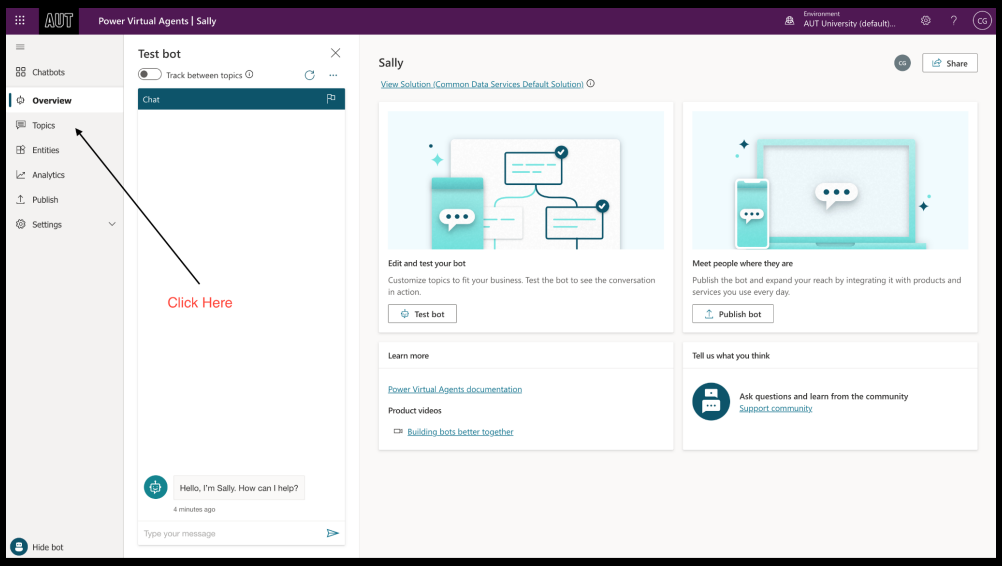
- Step 1 - Get Started at this website, and complete sign in and plan settings.
    - https://web.powerva.microsoft.com/

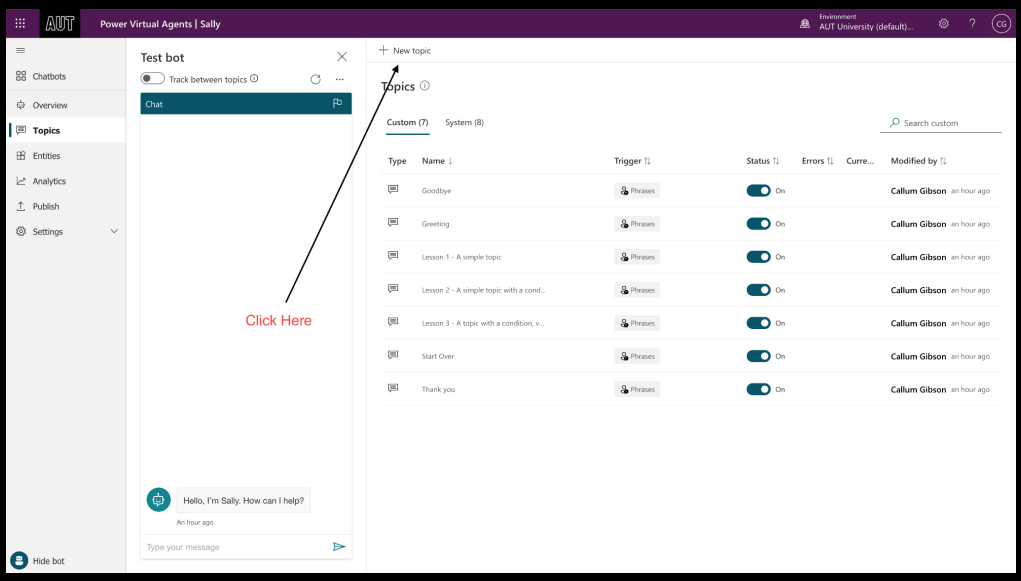
- Step 2 - Once on the home page, click the Chatbots option in the side menu.
    - 

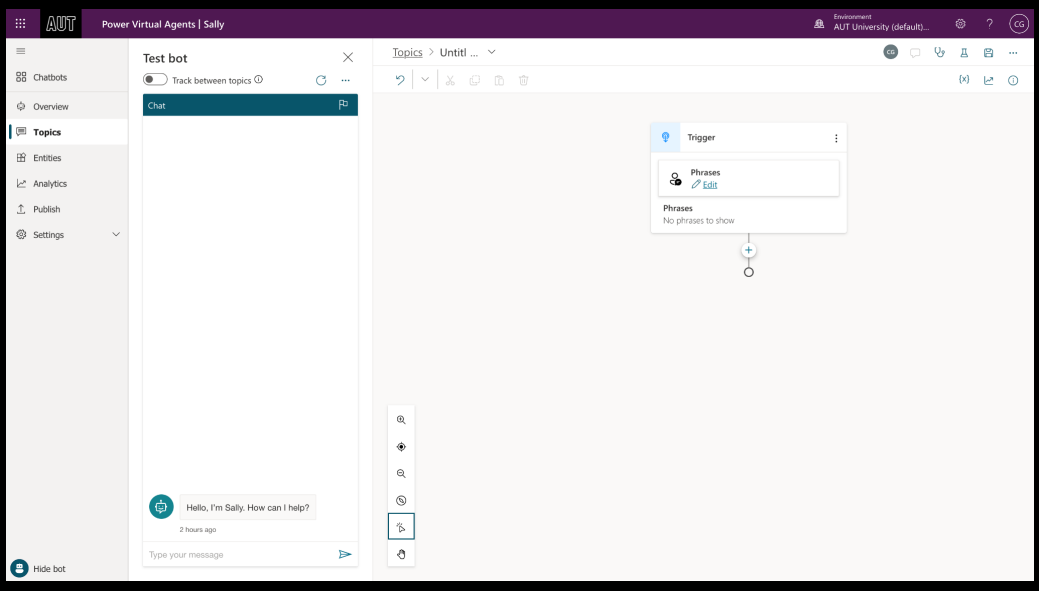
- Step 3 - Click “+ New chatbot” at the top of the page
  - 

- Step 4 - Set up the initial bot settings, assigning name and language.
  - 

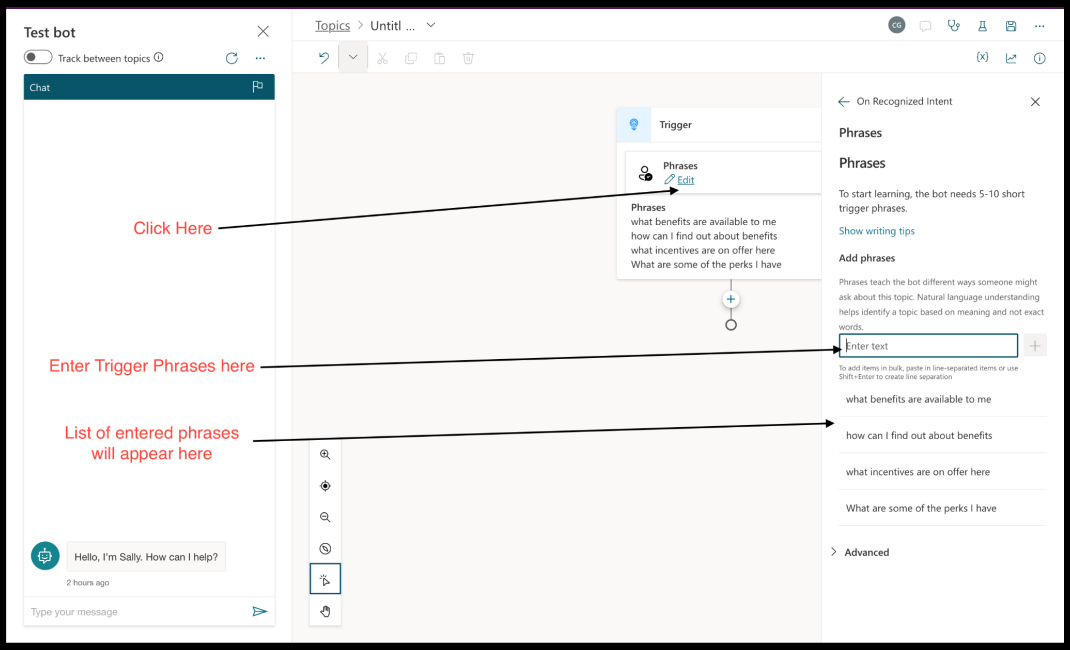
- Step 5 - Once the Bot has been created, you will be greeted by this screen.
  - 

- Step 6 - To add new queries, click on topics in the side menu.
  - 

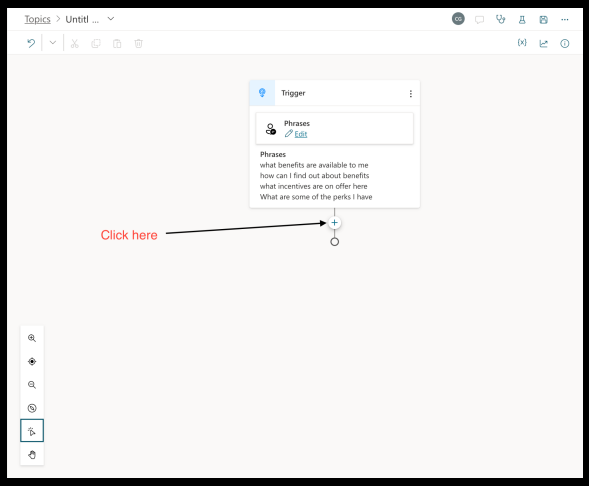
- Step 7 - Click the “+ New Topic” in the top row to add a new query.
  - 

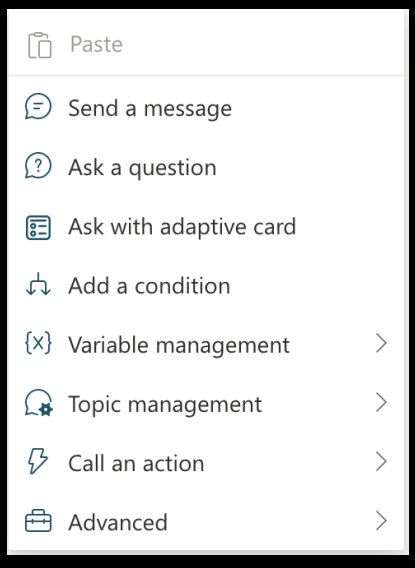
- Step 8 - You will now be on the page where you can add how the query can be triggered, how it will respond, as well as any other systems required.
  - 

- Step 9 - For this example, if you click on Edit under Phrases, you will be able to start loading trigger phrases for this particular query. Queries should be short phrases, between 5 and 15 words, and can cover your expected responses in a variety of ways. This allows the bot to learn how a question may be asked, and will allow for further self improvements over time.

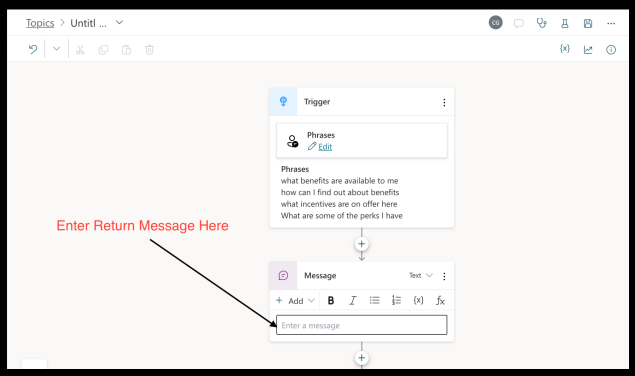


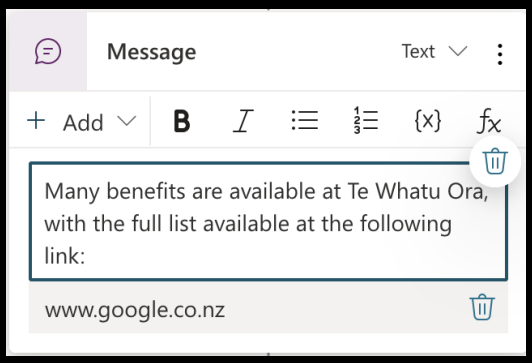
- Step 10 - Now that the trigger phrases have been loaded, click on the plus to open up options of the next part of the flow in the form of the popup as seen on the right



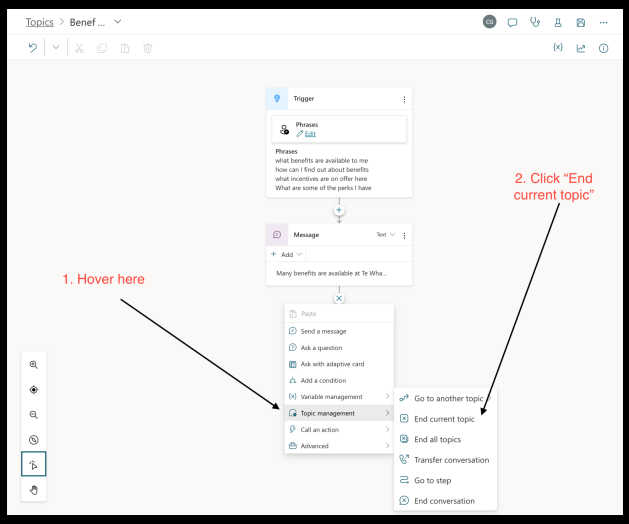


- Step 11 - Click the “send a message” option at the top of the pop-up menu, and you will see a new flow added with an input available for a message. In this case, I have just returned a basic output, with a link to Google.

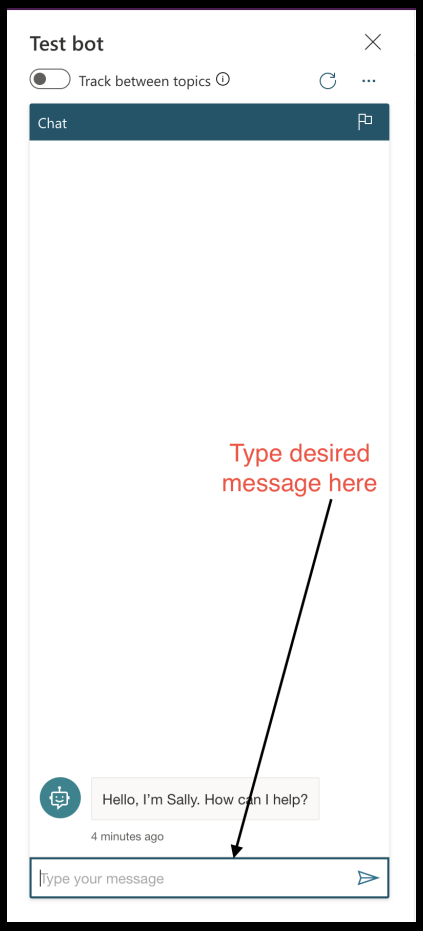


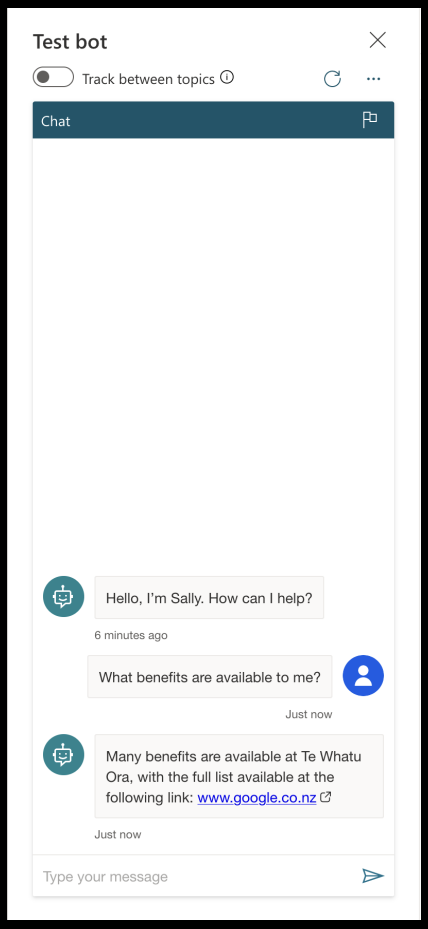


- Step 12 - Finally, click the plus below message again, and if you hover over “topic management” you will see an option to “End current topic”. Do this to formally end the current chain of conversation.



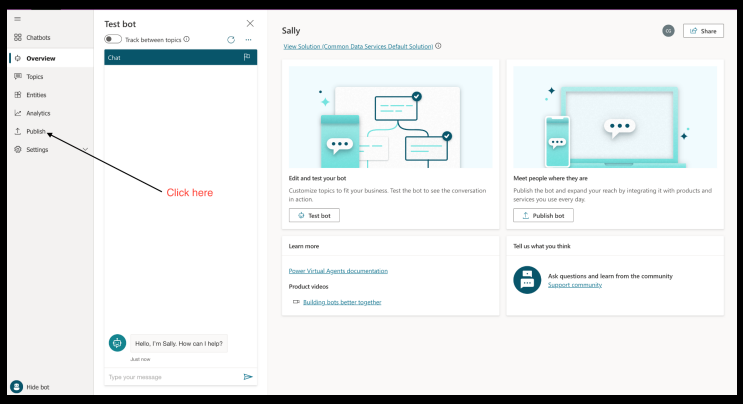
- Step 13 - Now you can test the query in the side chat that has the bot running. By typing in “What benefits are available for me”, Sally will be able to return the message as stated in this chat. Now you will see Sally return the output that was specified in the earlier step.



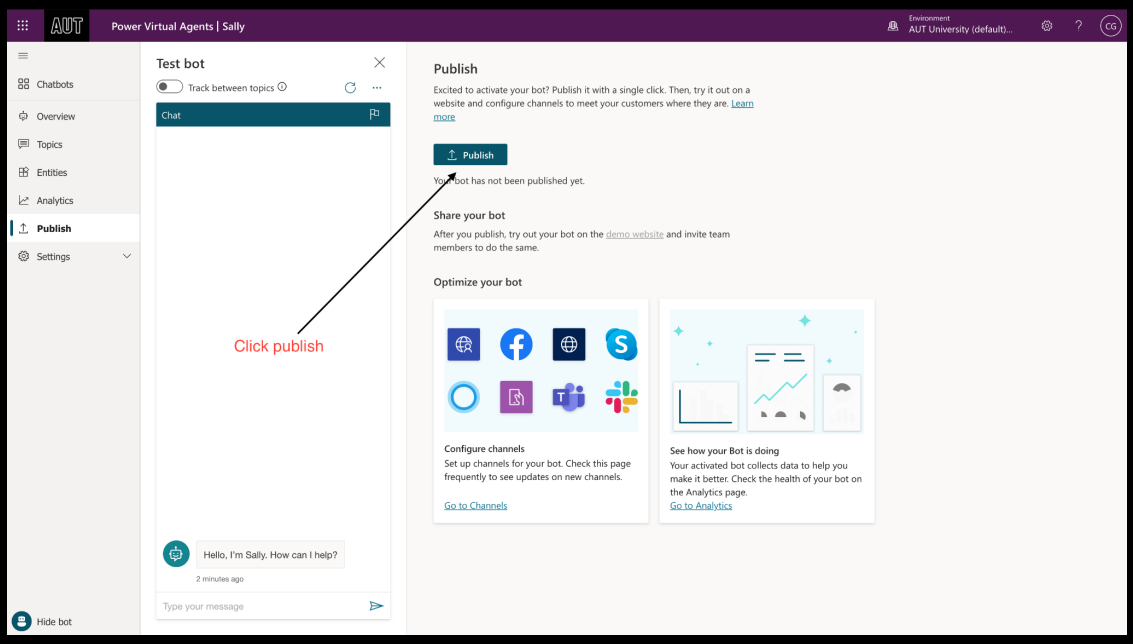


- Step 14 - Congratulations, you have just made your first query, there are many more ways you can configure this bot, with different information, different sources, and different forms of output, and while we have not been able to use this software fully due to costs, compiled at the end of the section will be a list of resources for your own research purposes should you choose to go this route.

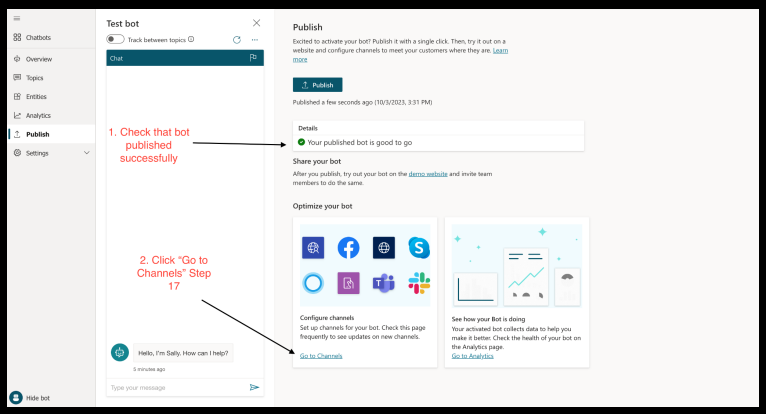
- Step 15 - Finally, let's quickly go through publishing the bot to Microsoft Teams. Firstly, click on the publish button on the side menu that can be seen anywhere in the bot space.



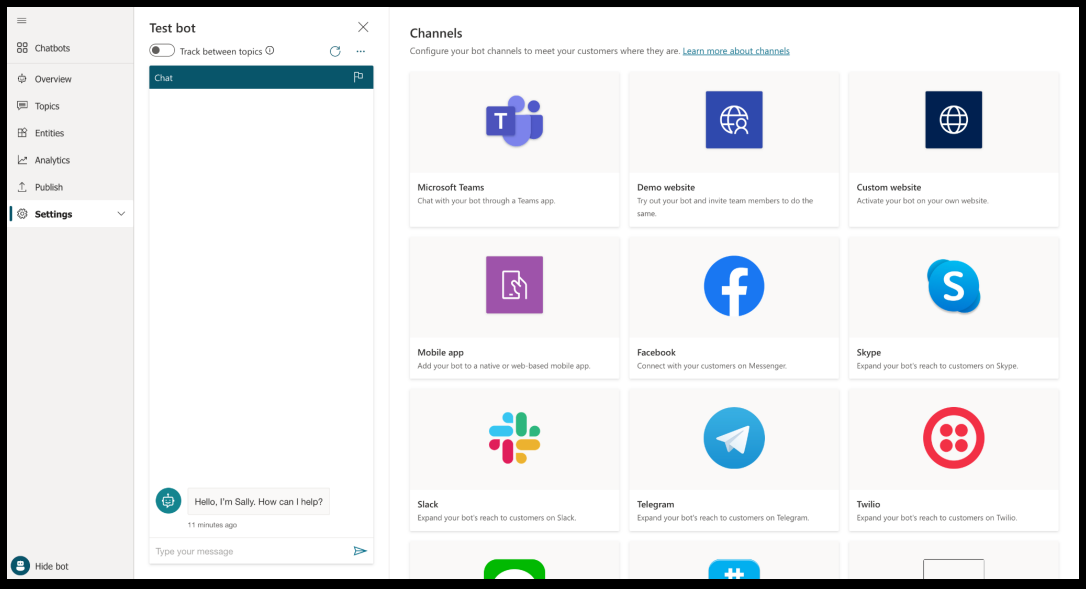
- Step 16 - Now click publish in the middle of the screen, and click confirmation on the pop up window to officially publish the bot.



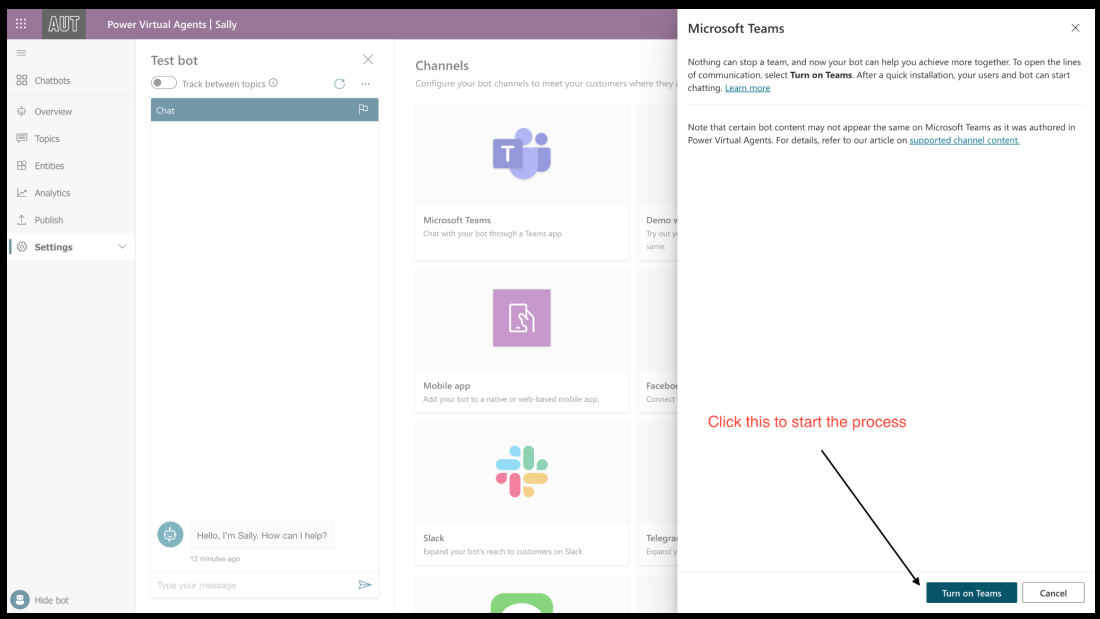
- Step 17 - Upon successfully launching, you should now see a notification at point 1 like the screenshot. Now click on the “Go to Channels” section in the middle of the screen to set up Teams launching.



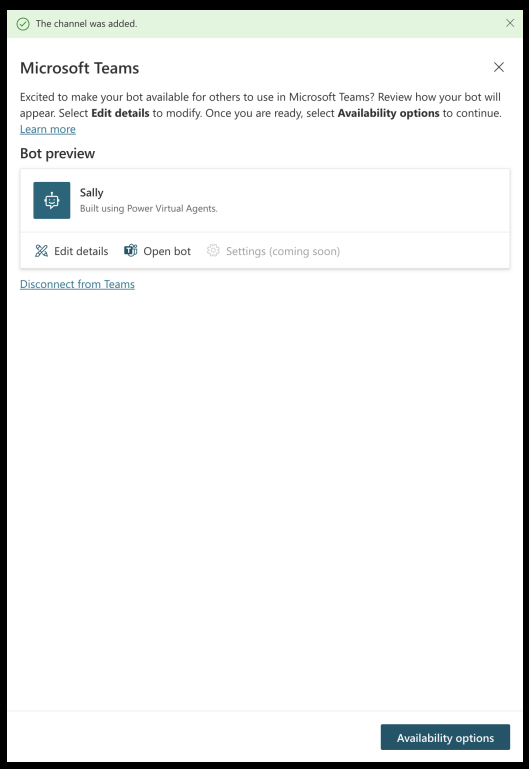
- Step 18 - Now you will see a screen with all the different places to publish to, for the focus of this tutorial we will be only looking at Teams. Click on the Microsoft Teams in the top left of the window to start the process.

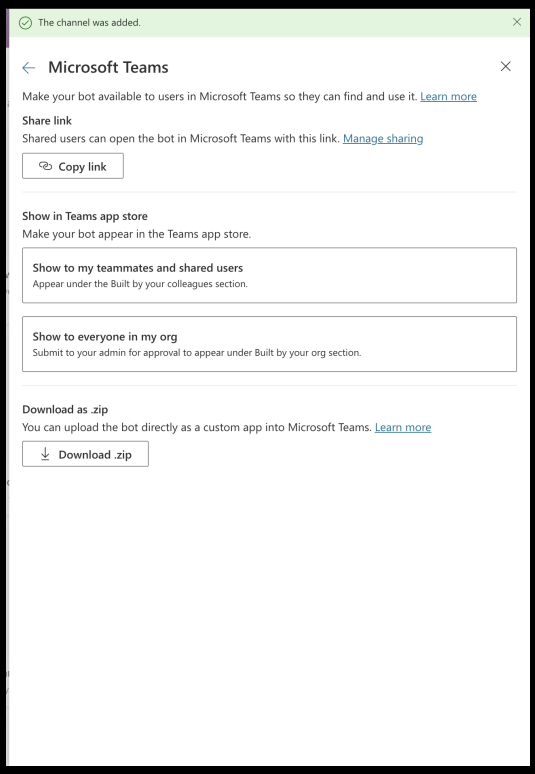


- Step 19 - A confirmation slide window will appear on the right, at the bottom, click the button on the left to begin the process of launching to teams.

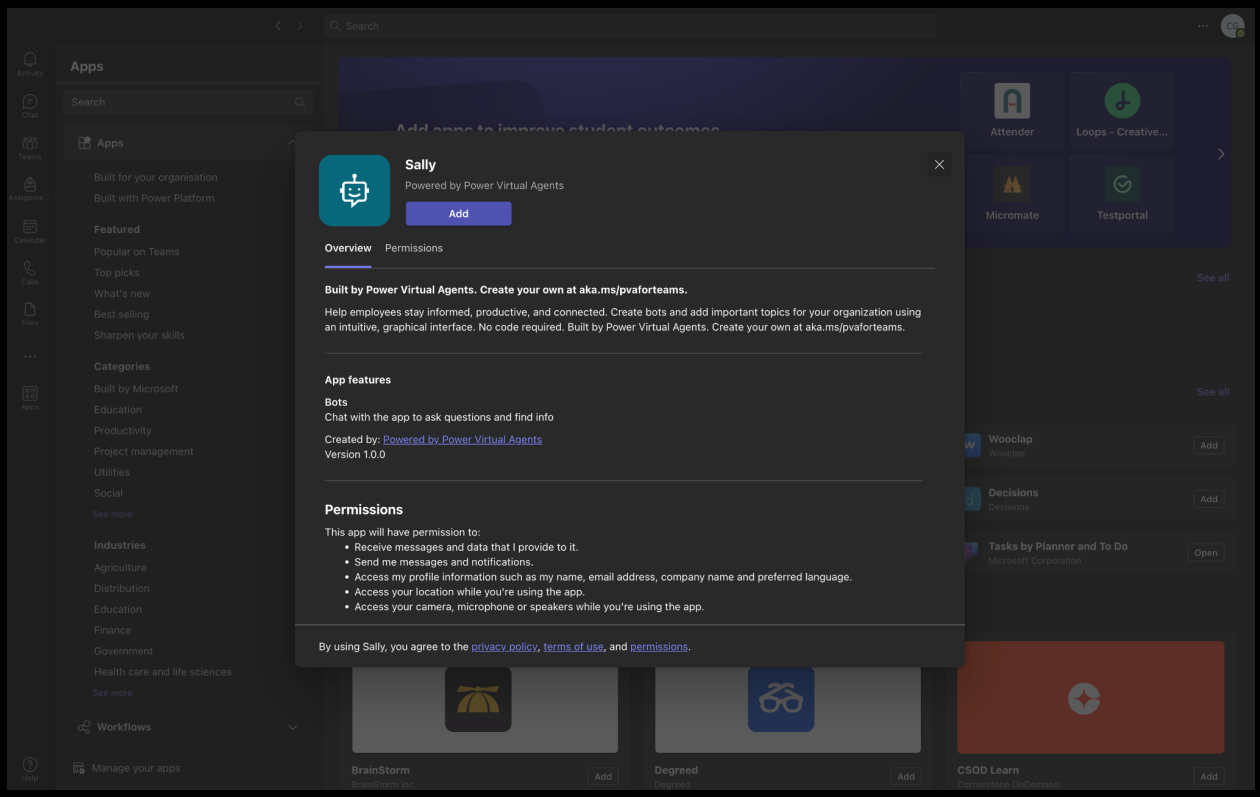


- Step 20 - Once completed, you will have a new screen with Teams settings. Availability options allows you to determine who will have access to the bot, whether it is select users, or everyone in your organisation.

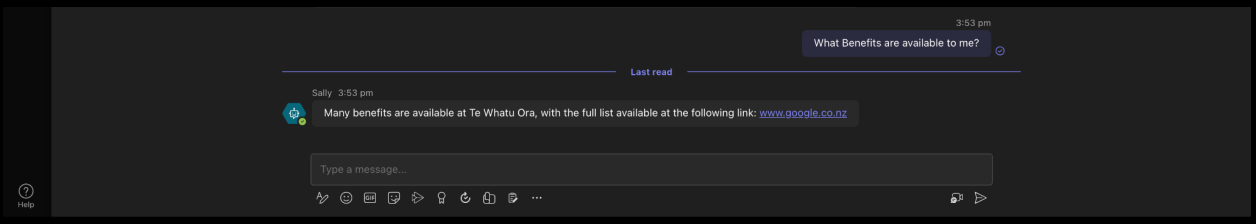


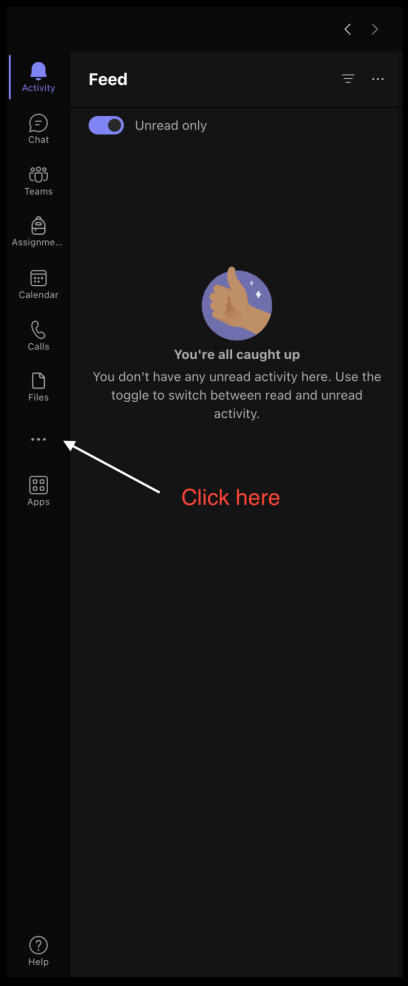


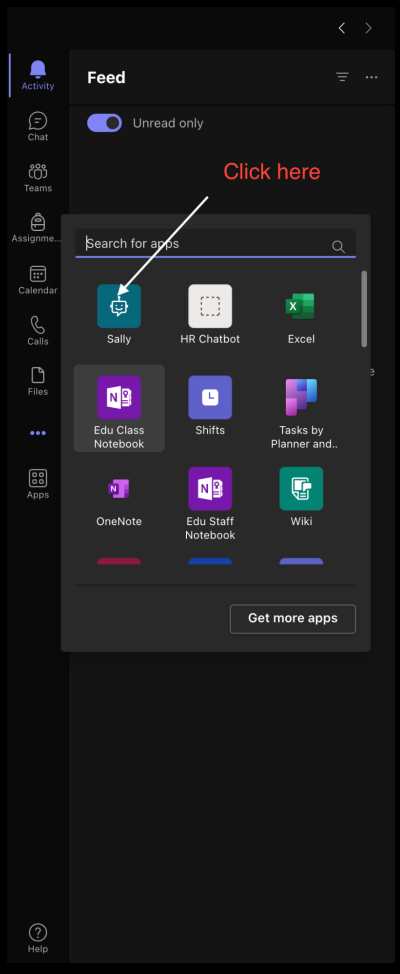
- Step 21 - If you clicked “Open bot” on the teams menu, you will be asked if you would like to add it to your teams.



- Step 22 - Now in teams, Sally will operate in the same way as it did in the development suite, being able to answer the same query from before. It is located on the left side of teams under apps.







- Step 23 - You have done it! The bot is now hosted on teams, and you can start the queries just as you could online. As mentioned before please see below for more resources for adding more functionality to Sally.

Implementing via. Bot Framework Composer

The utilisation of Bot Framework SDK via Bot Framework Composer is a simple way to build regex (regular expression) based bots.

Step-by-step guide to building bots via Bot Framework Composer:

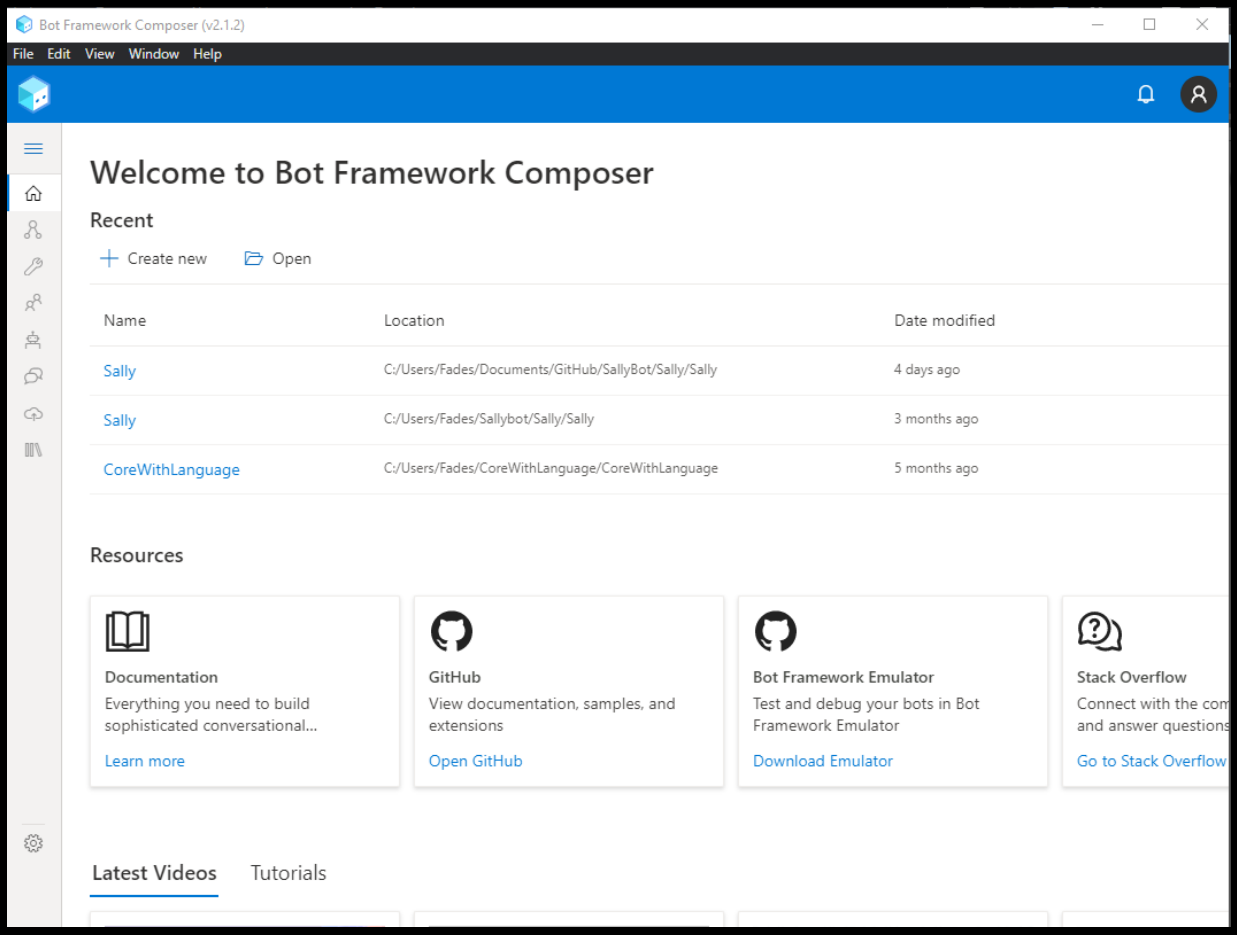
Prerequisites:

Download and install the following:

- Node.js (includes npm) : https://nodejs.org/en/download
- .NET Core SDK and ASP.NET Core Runtime: https://dotnet.microsoft.com/en-us/download/dotnet/3.1
- Bot framework composer (direct download link): https://github.com/microsoft/BotFramework-Composer/releases/download/v2.1.2/BotFramework-Composer-2.1.2-windows-setup.exe

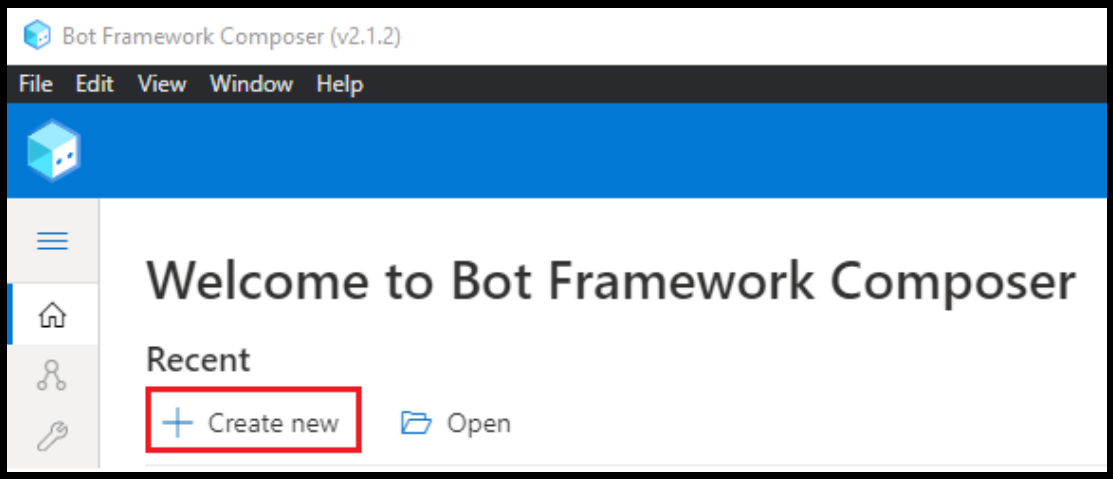
Part 1. Creating your bot template.

Step 1. Open bot framework composer.

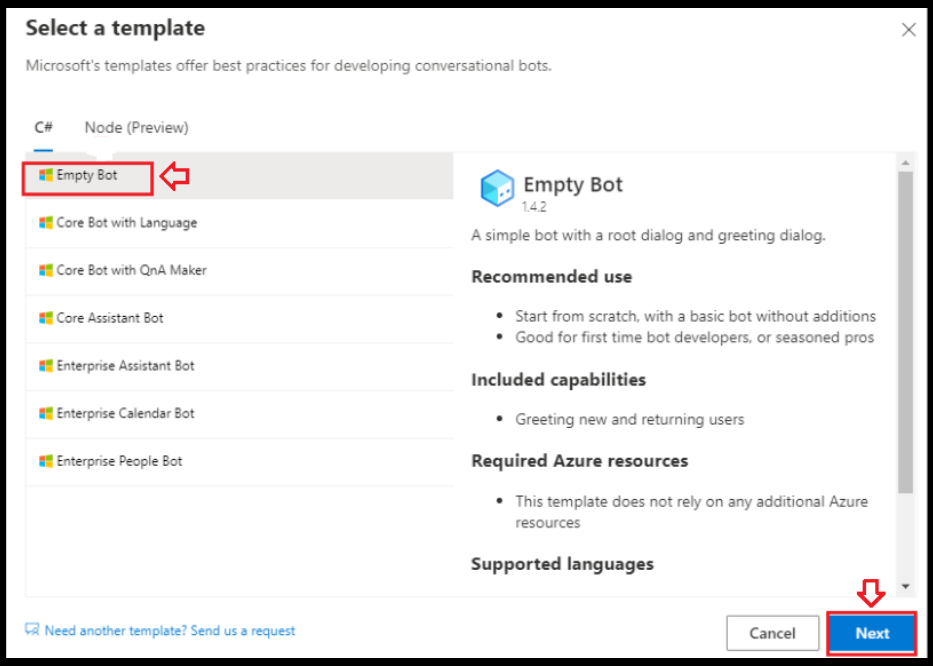


Step 2. Click on “Create New”.

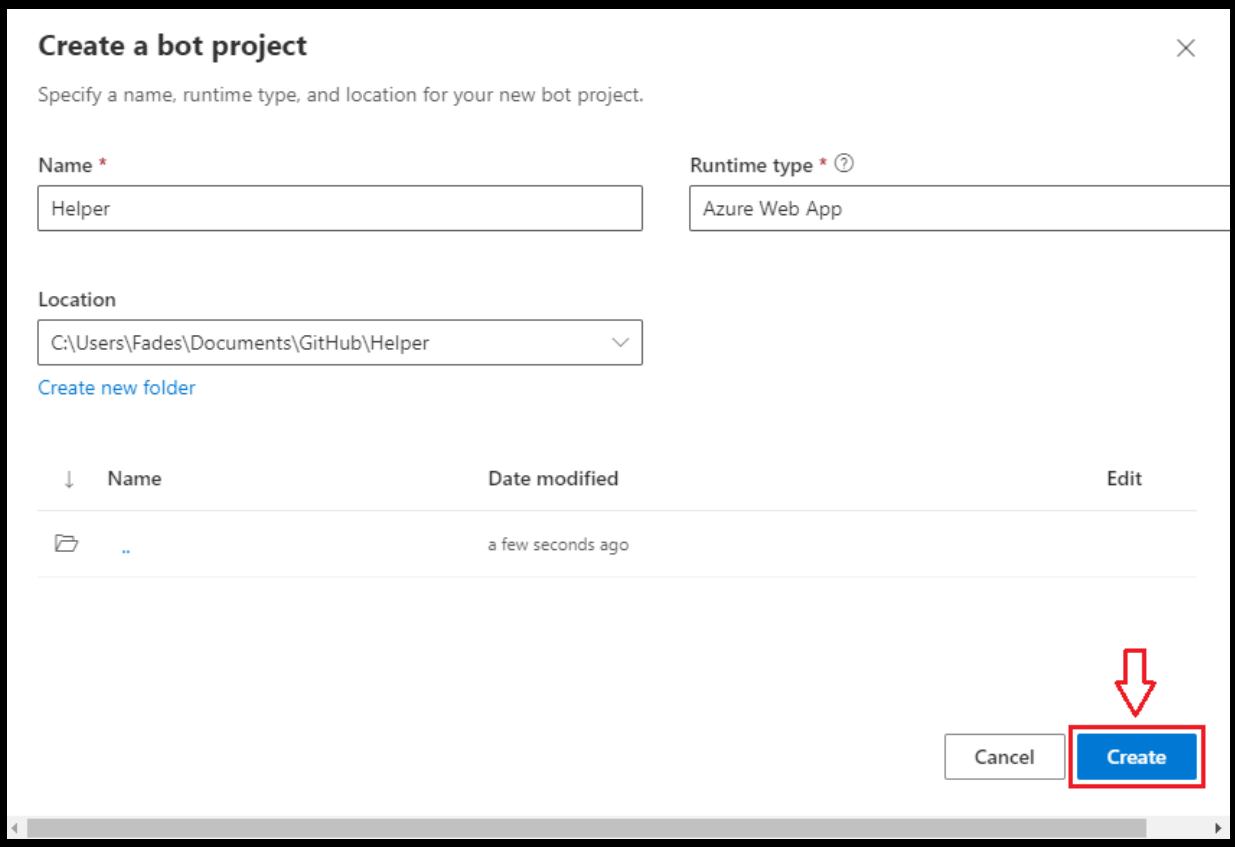
Important Note: If you already have a bot, then click “open” and navigate to the location of the bot you wish to deploy. Once the bot is opened, skip ahead to Part 5 of this tutorial.



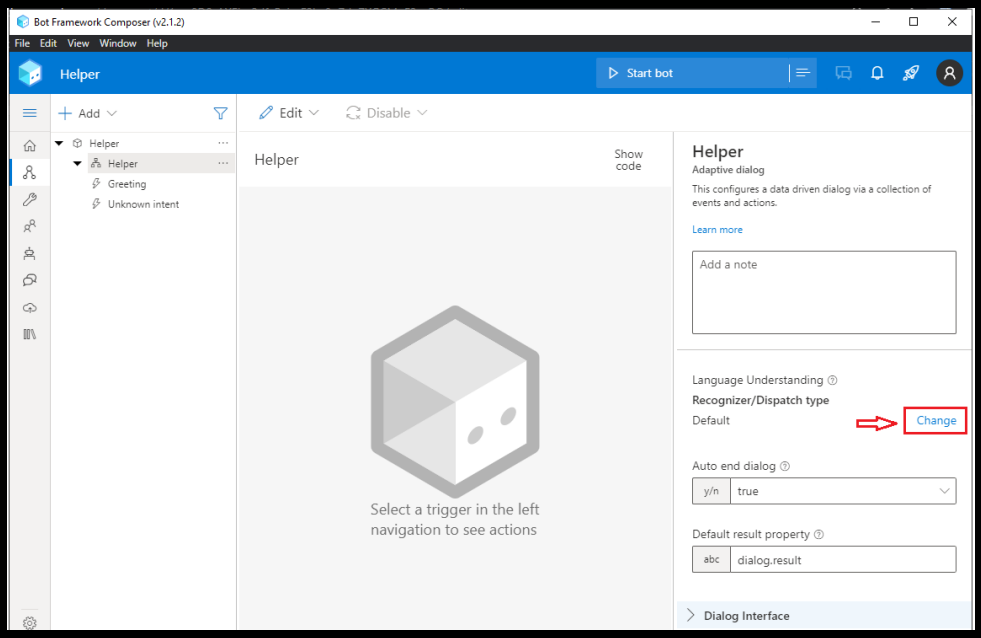
Step 3. Select on Empty Bot under C# and Click Next.



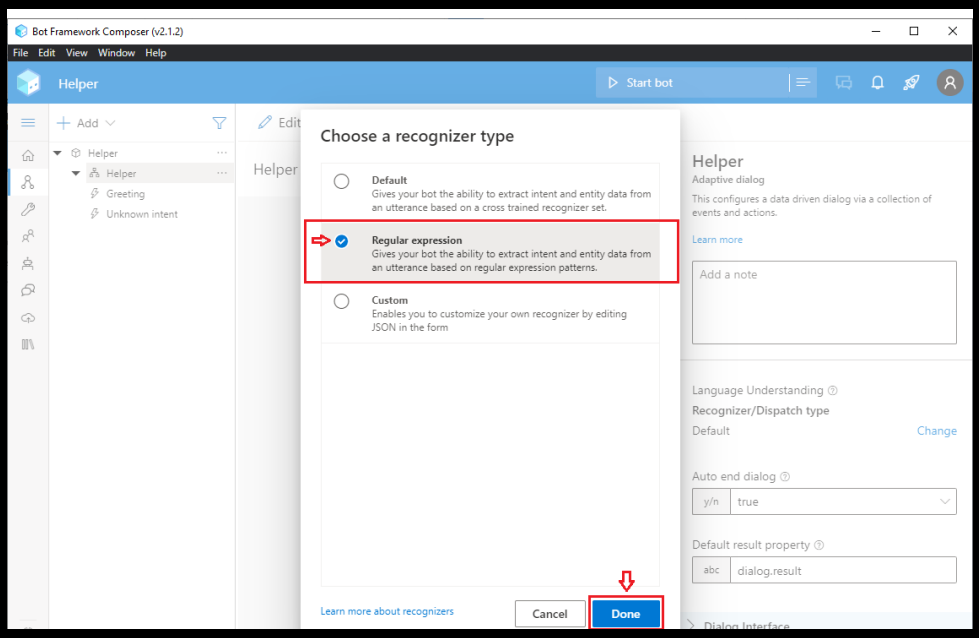
Step 4. Name your bot. Choose the Azure Web App Runtime. Create or select a folder to locally store your bot.



Step 5. Select the dialog with the same name as your bot and click “Change” under language understanding.



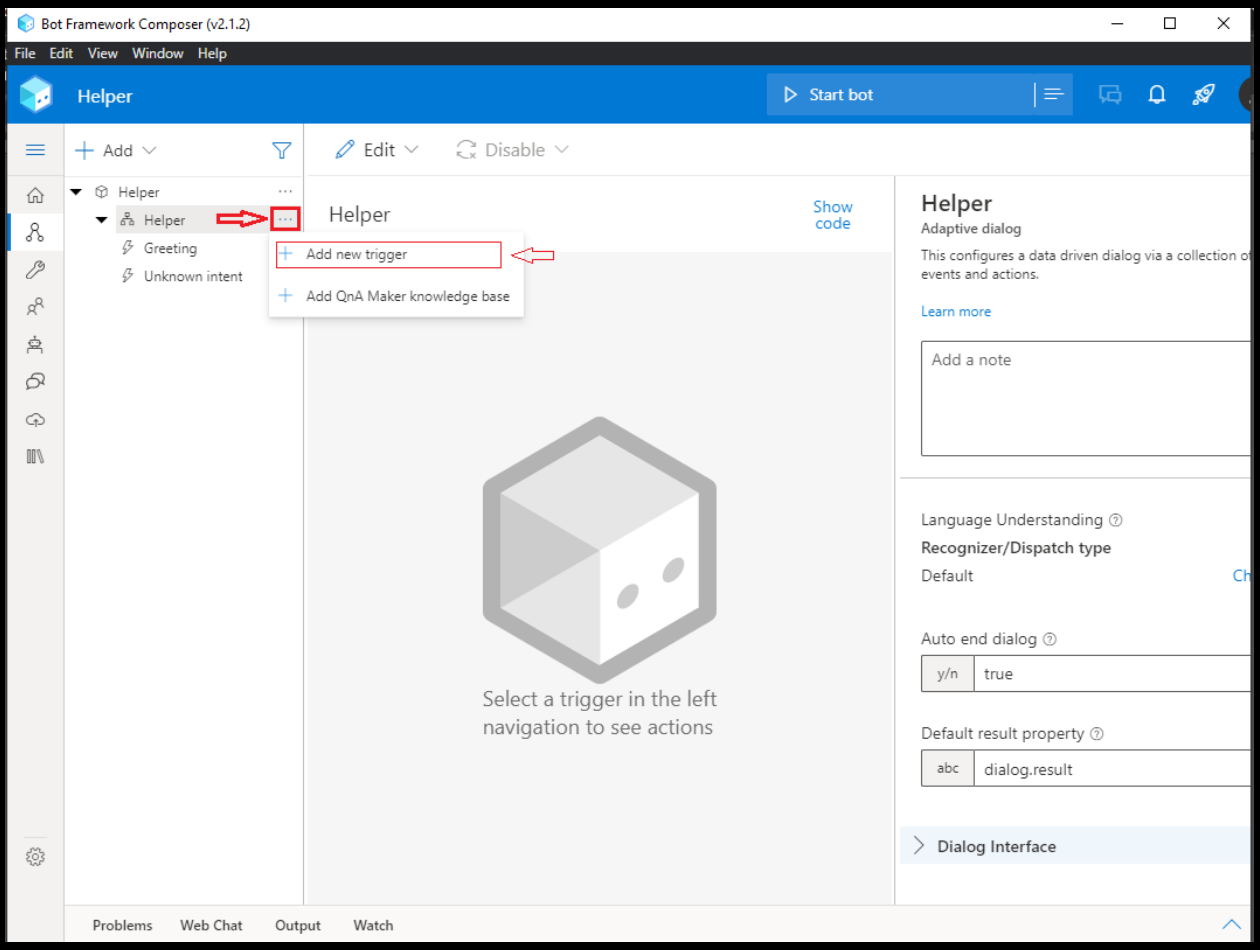
Step 6. Select “Regular Expression” and click “Done”.



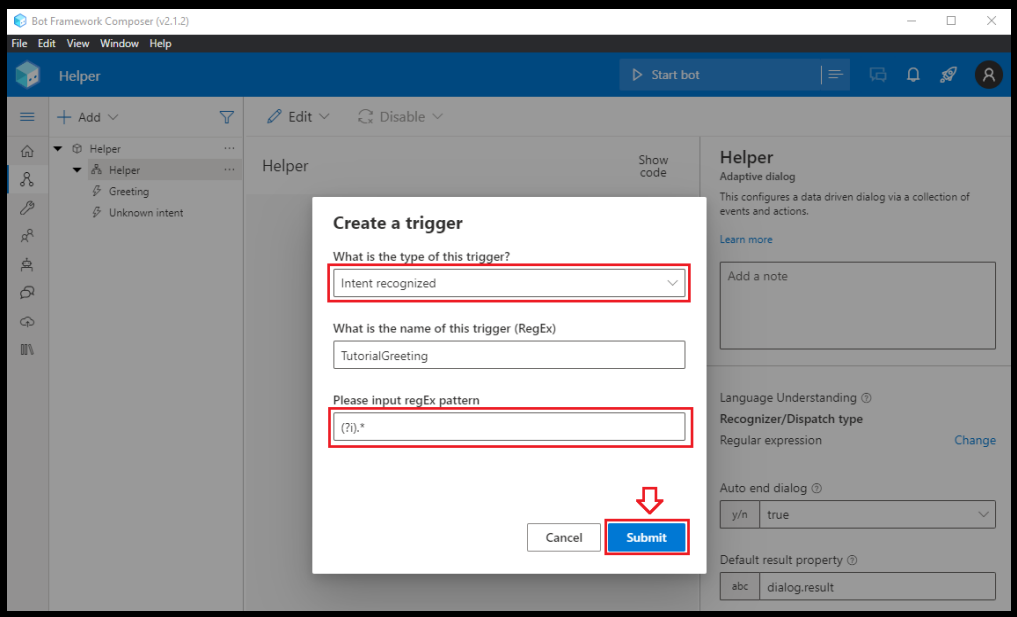
You have now created your bot and set it up for regex. At this stage you may wish to create a github repository for the folder you identified in Step 4.

Part 2. Creating triggers and Adaptive Card dialogs.

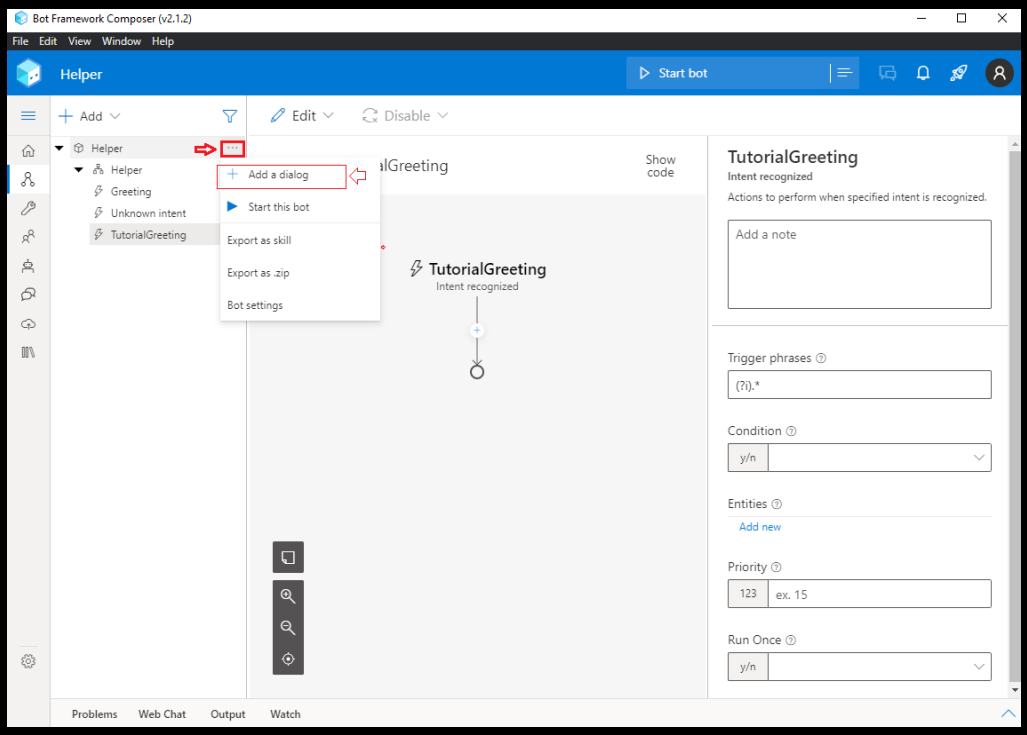
Step 1. In your bot click the "..." icon beside the dialog which shares the same name as your bot. Then, click "Add new trigger".



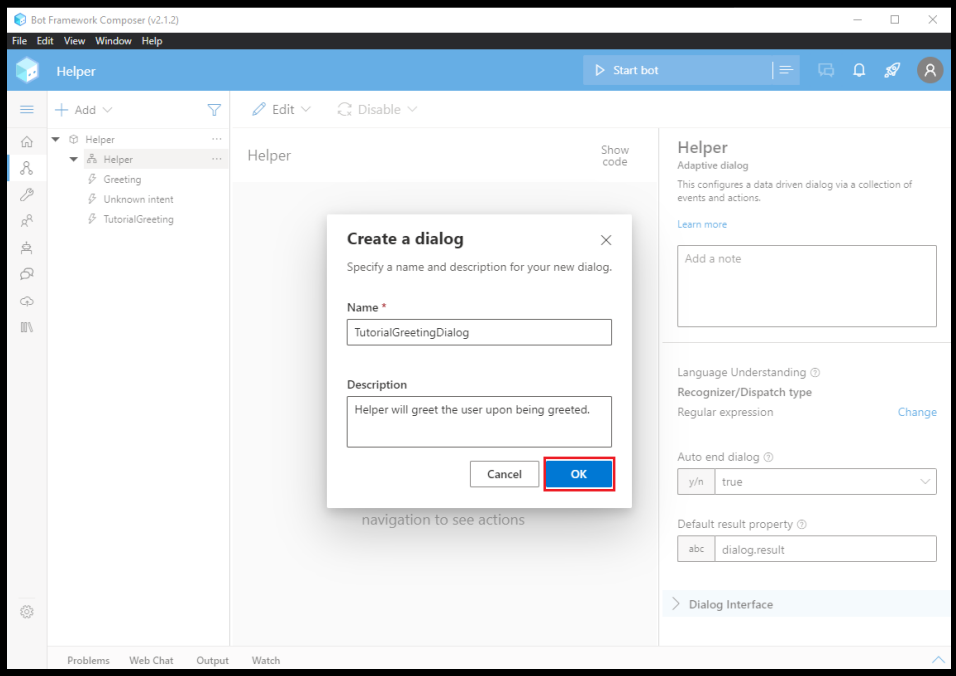
Step 2. Choose “Intent recognized” for trigger type. Enter (?i).* as pictured if you are unsure what to put here, else enter a suitable regex pattern. Regex will be discussed in detail later in the guide.



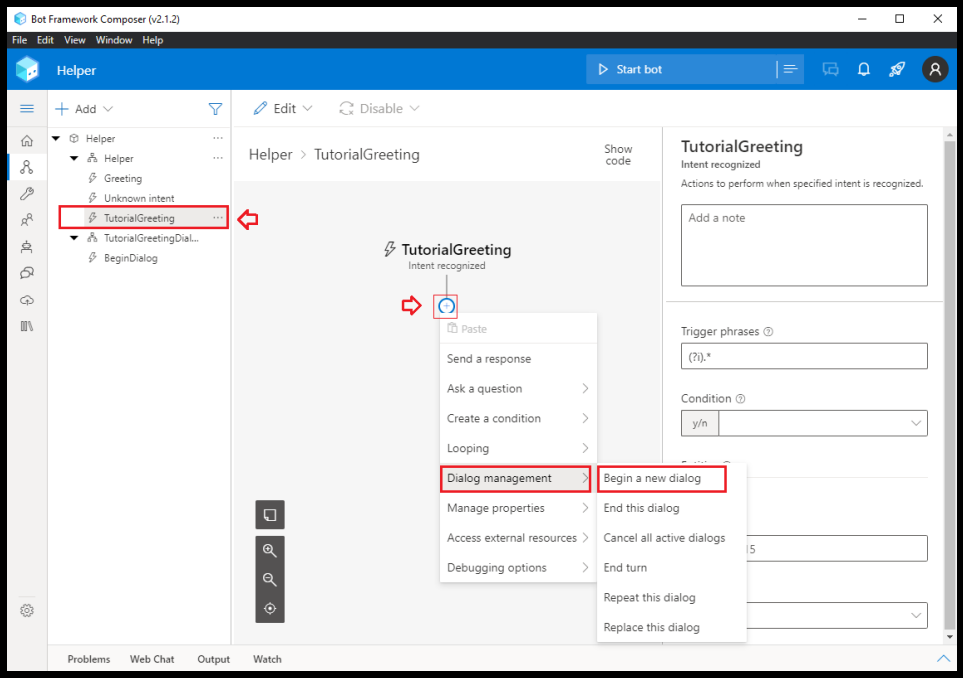
Step 3. Click the “...” icon next to the name of your bot. Select “Add a dialog”.



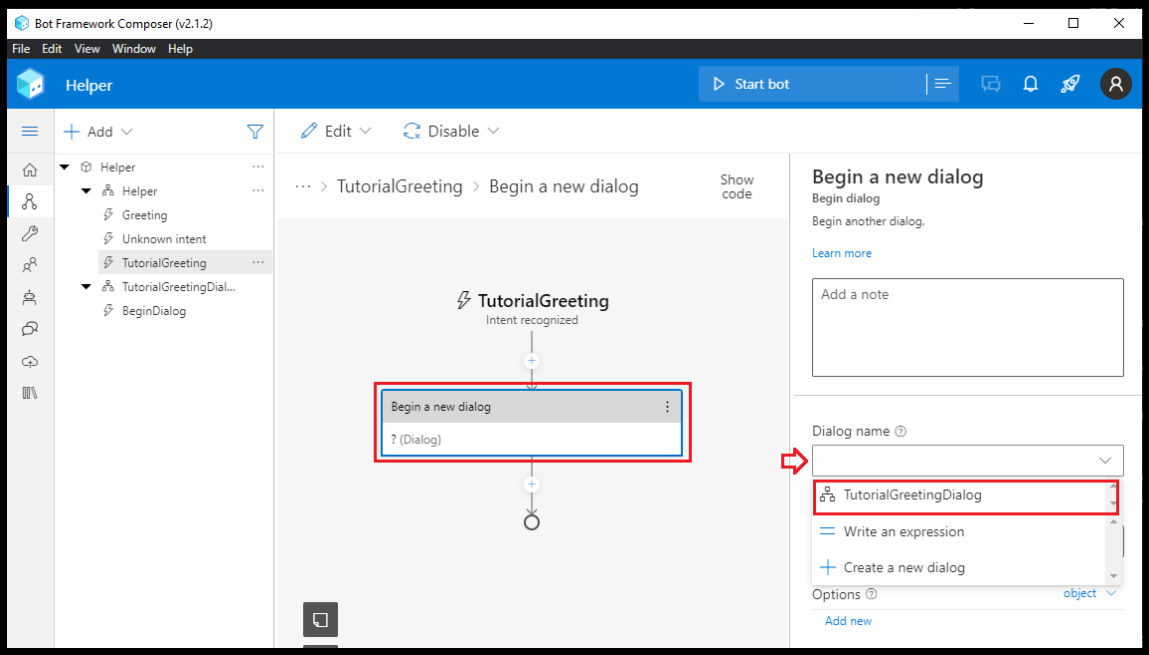
Step 4. Name your dialog, describe its purpose and click OK.



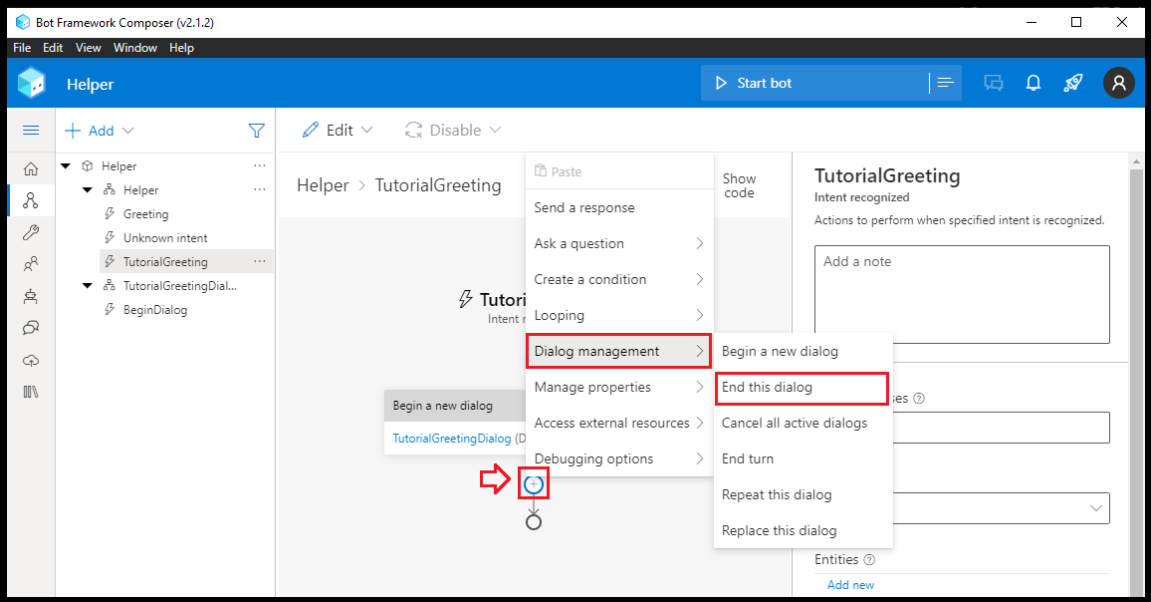
Step 5. Select your trigger and click on the '+' icon in the centre of the screen. Hover over dialog management and select "Begin a new dialog".



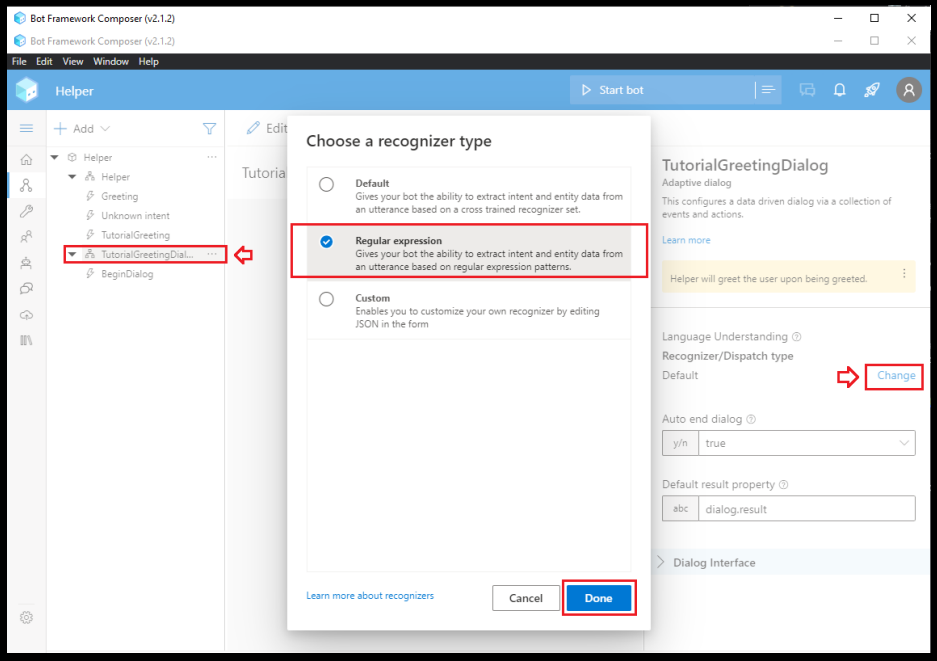
Step 6. Click the “Begin a new dialog” box and in the dropdown menu select your dialog.



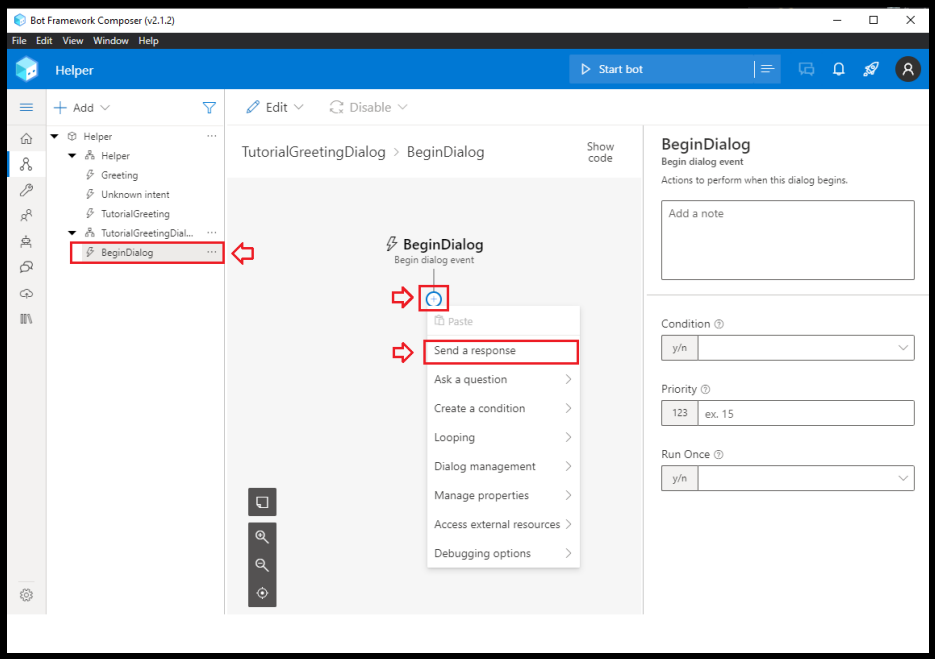
Step 7. Click the next + sign in the chain. Hover over “Dialog management” and select “End this dialog”.



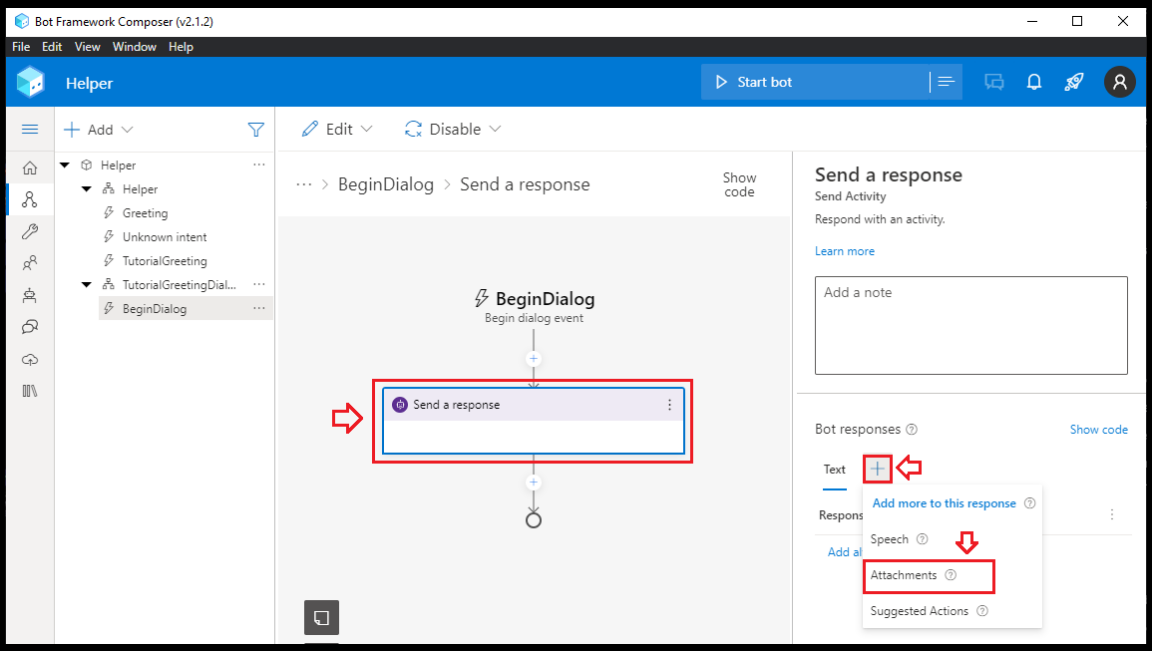
Step 8. Select your dialog and change the Language Understanding to regular expression.



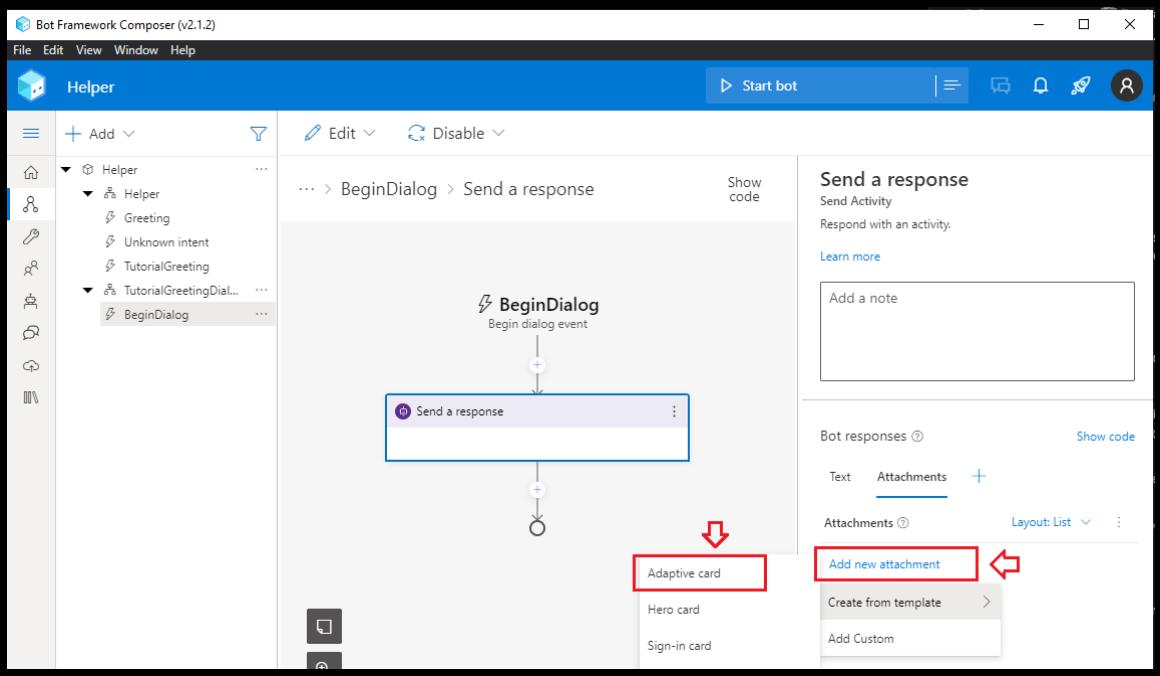
Step 9. Select “BeginDialog” under your dialog and click the first + in the chain. Then, select “Send a response”.



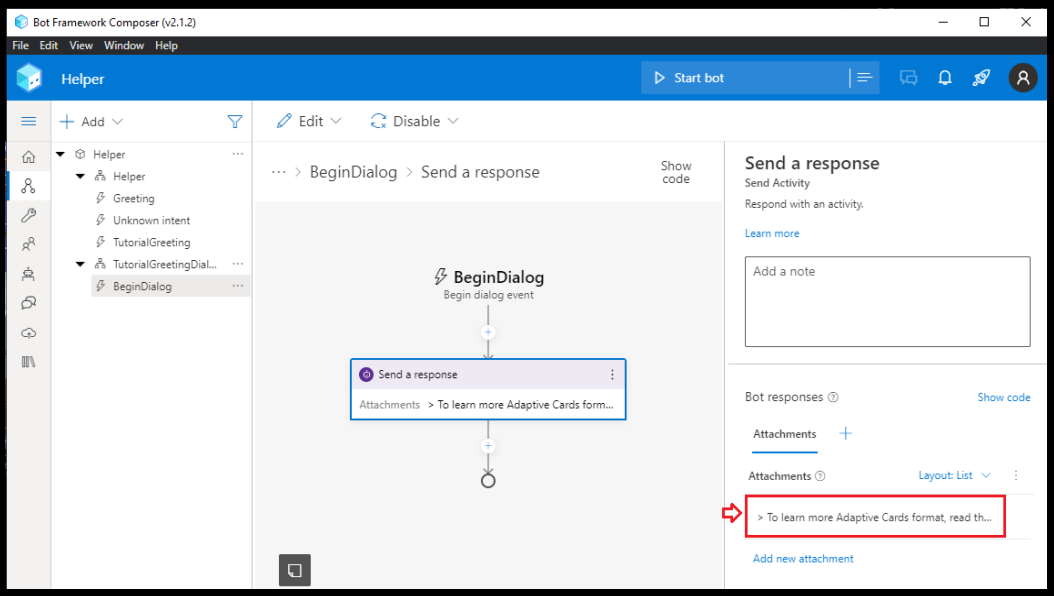
Step 10. Select the dialog box, click the + icon beneath “Bot responses” and select “Attachments”.



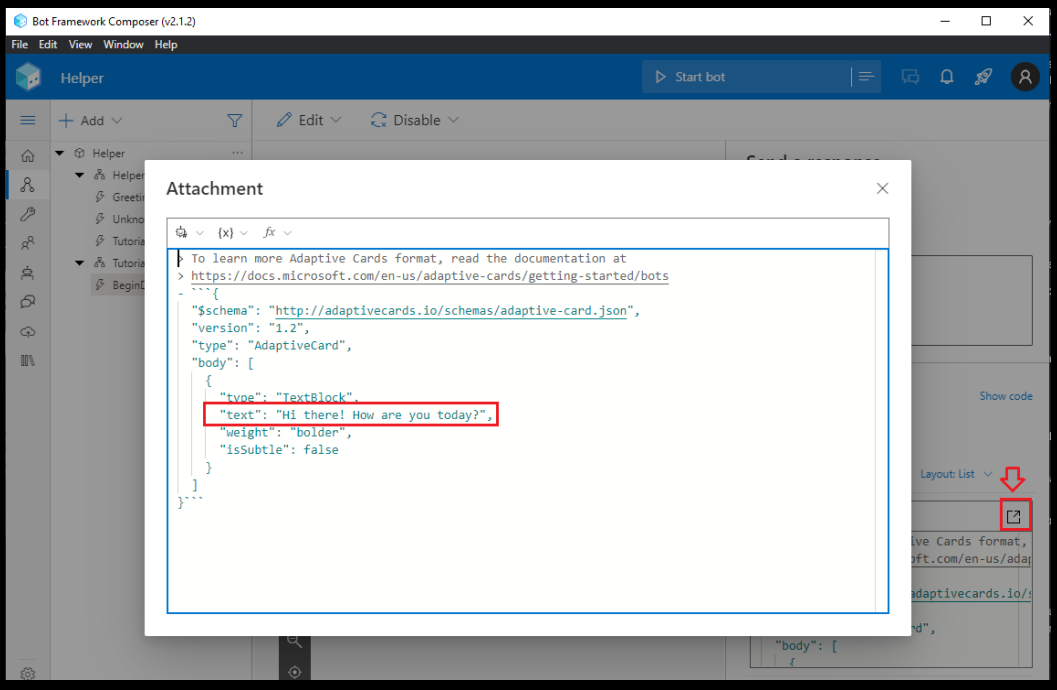
Step 11. Click “Add new attachment” beneath “Attachments”. Hover over “Create from template” and select “Adaptive card”.



Step 12. Click the row with the adaptive card content beneath “Attachments”.



Step 13. Click the expand icon on the right to open up the contents of the attachment. Inside you will see an adaptive card in JSON format. The line identified as “text” holds the current text output content of this Adaptive card.



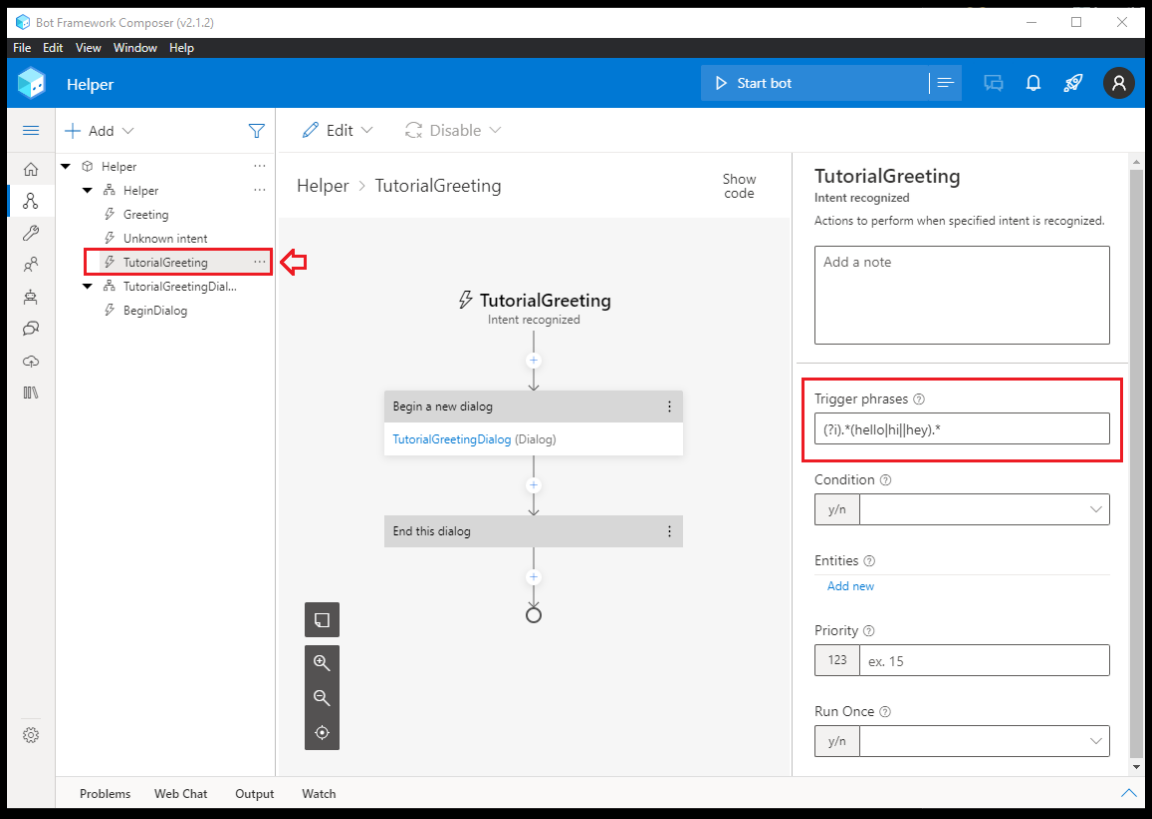
You have now set up your trigger and Adaptive Card dialog.

Part 3. Making a regex pattern.

For more detailed information about regex patterns in the context of this bot please refer to Section {Placeholder}.

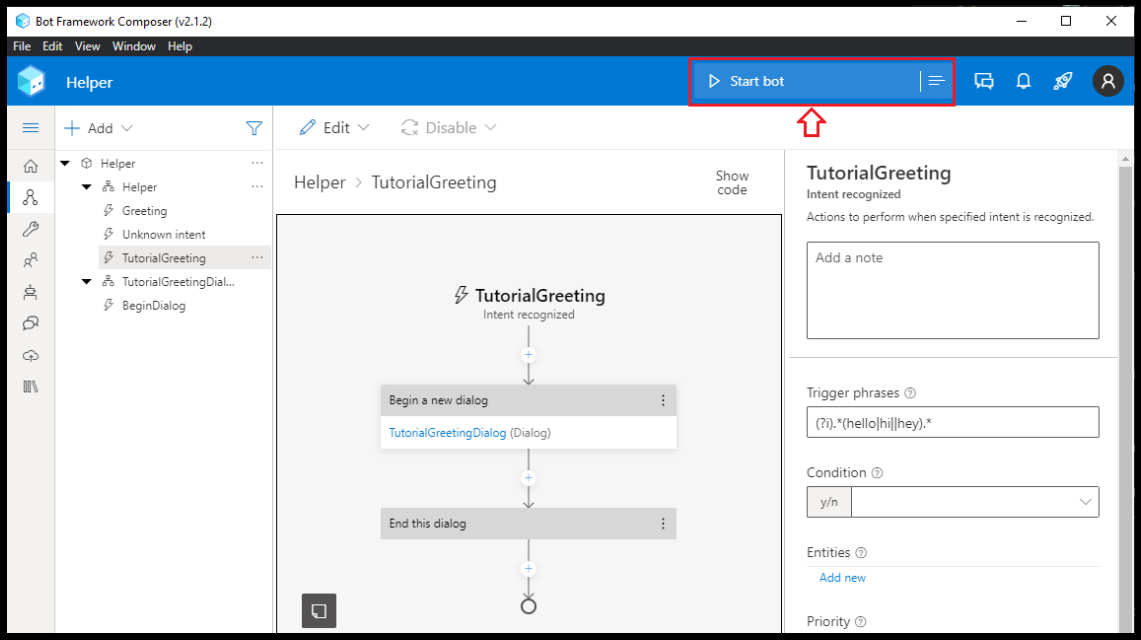
Step 1. Select your trigger. In the “Trigger phrases” text box we can add phrases which will indicate a user's intent to our bot, allowing it to form the correct response.

In our case it is a simple greeting. We start with { (?i) } which makes anything that follows case insensitive (meaning for example that “hi”, “Hi” and “hI” are all considered the same) and { .* } which we will explain later in our guide. Then in brackets we have the various ways a user can say our desired intent, for example “hi”, “hello” and “hey”, separated by the “or” (|) operator. We finish with { .* }.

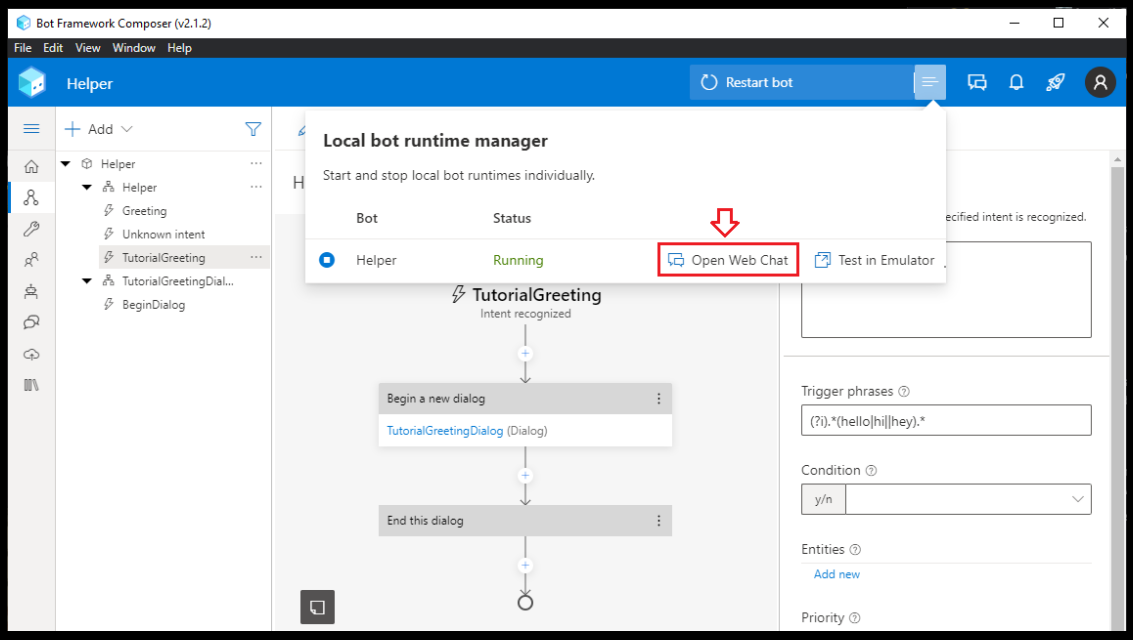


Part 4. Running a bot.

Step 1. Click “Start bot”.

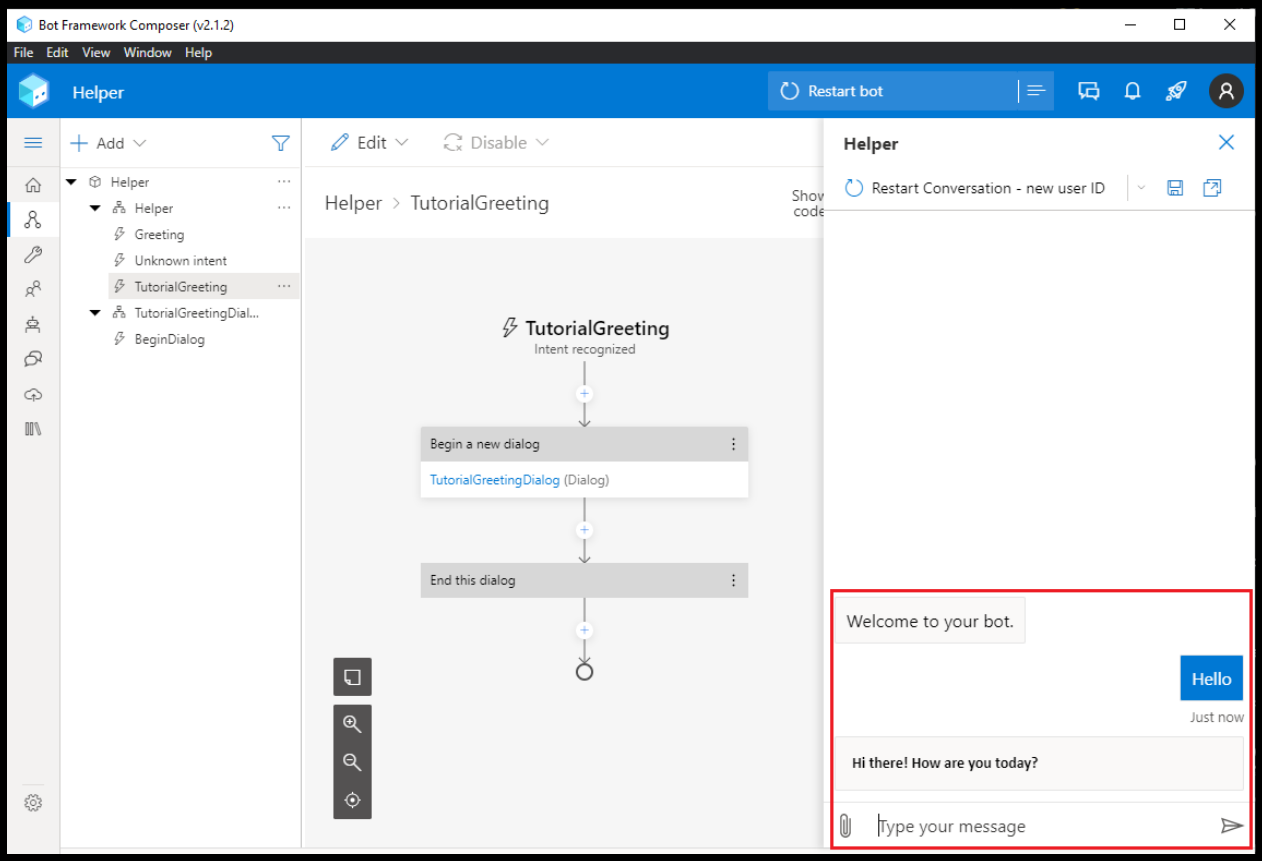


Step 2. Click “Open web chat”.



Step 3. You can now communicate with your bot via the web-app instance.

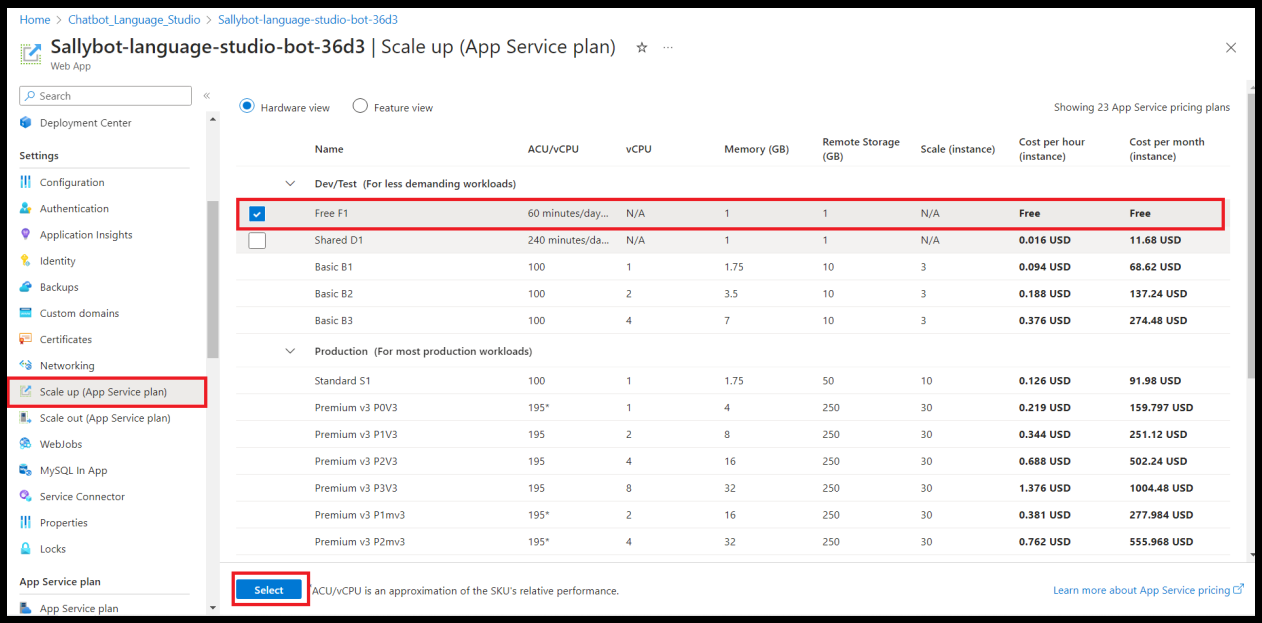
In this example I sent a message saying “Hello”, which was one of the 3 trigger phrases I defined earlier, and I am returned an Adaptive Card with the text content I wrote in my Adaptive Card dialog.



Part 5. Publishing your bot

Before deploying to Teams, you need to publish your bot as a bot application. Follow these steps:

- Open your bot project in Bot Framework Composer.
- In Composer, go to the "Publish" tab.
- Click on "Publish to Azure." This will create a bot registration and publish your bot as an Azure Web App Bot.
- Follow the prompts to configure your Azure resources. You will need to create an Azure Resource Group, an App Service Plan, and an App Name. **VERY IMPORTANT NOTE: When you do this, the default App Service Plan will be set to Standard. This plan costs roughly $5.00 NZD per day and costs begin accumulating straight away, so if you intend to use the free version of the app service plan, please go to this App Service resource or App Service Plan resource inside the Azure console right after provisioning and change to the free plan using the menus pictured below:**



Please also ensure that your Azure Bot resource is on plan “F0” as the free plan is right for a Teams implementation.

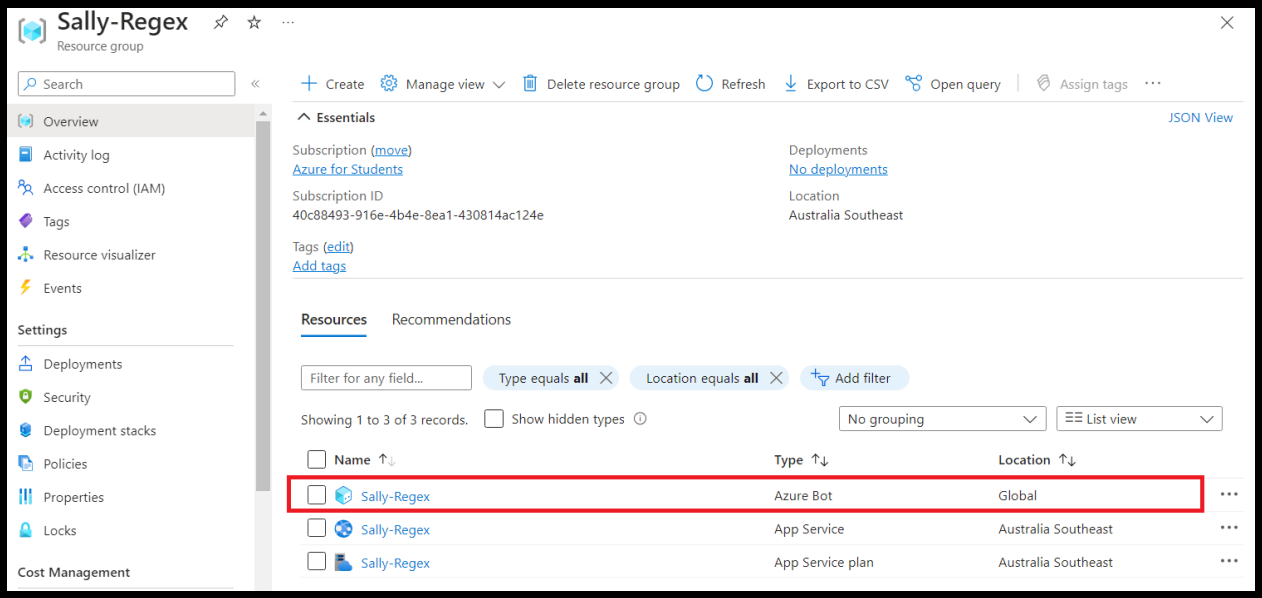
Once the publishing process is complete, Composer will provide you with the bot endpoint URL and the Microsoft App ID. Keep these details handy; you will need them in the next steps.

Part 6. Deploying to Microsoft Teams

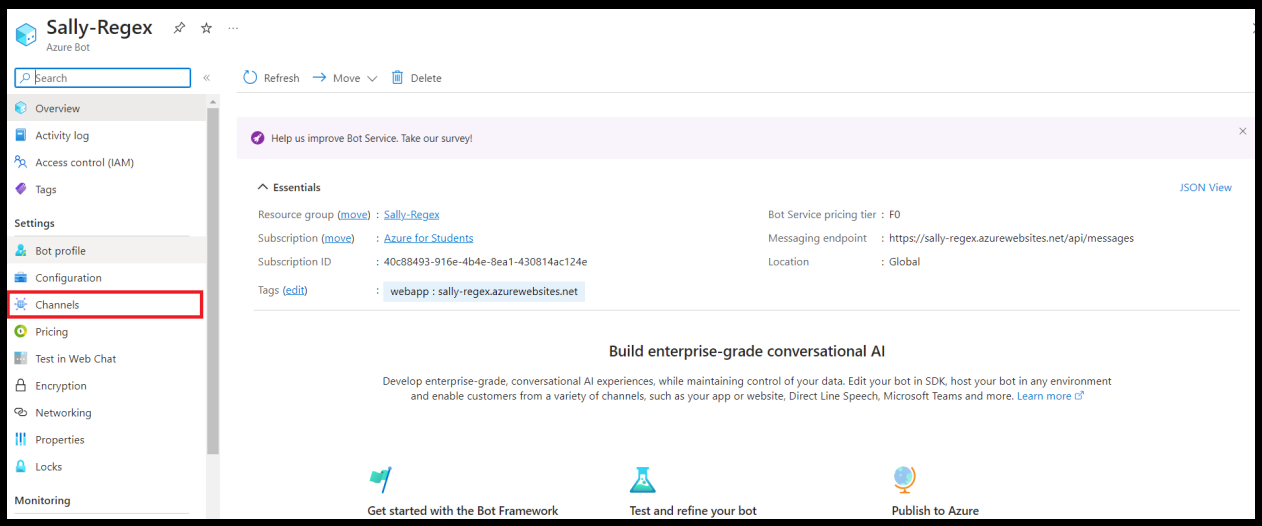
There are two main ways to access your bot in teams: a channel or an app.

Option 1: Channel

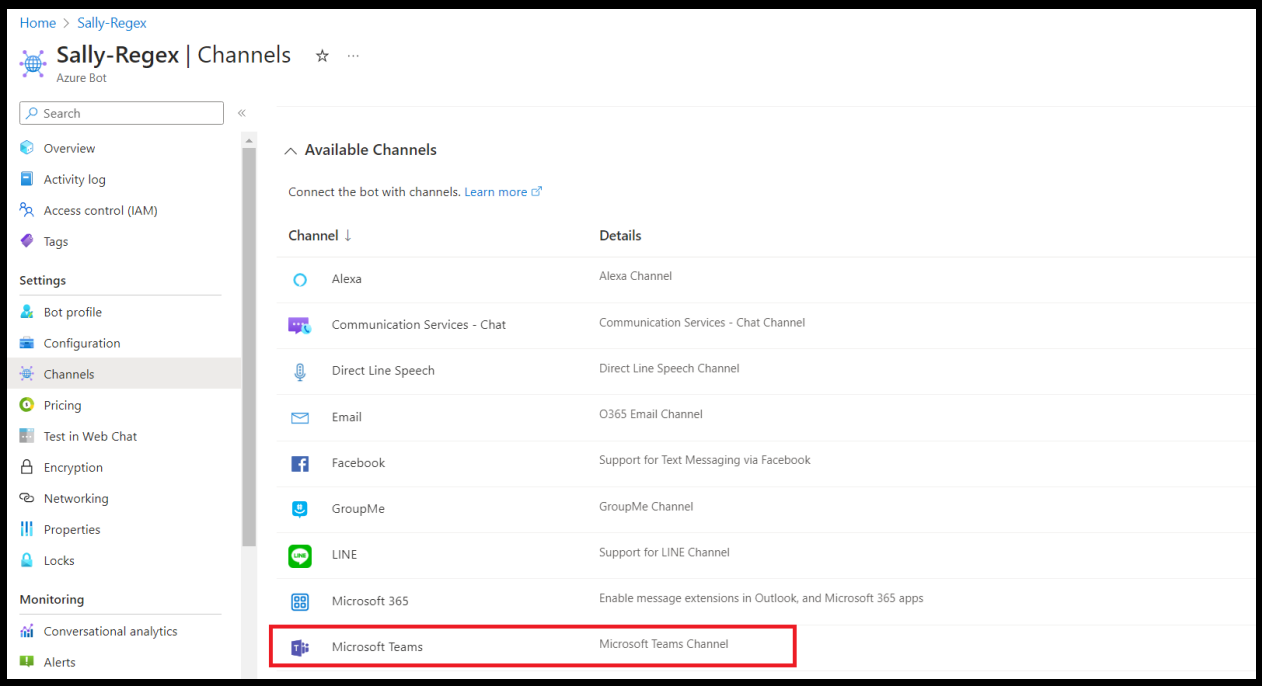
The quickest way to communicate with a bot using Microsoft Teams is to open the Azure Bot resource in your chatbot resource group:



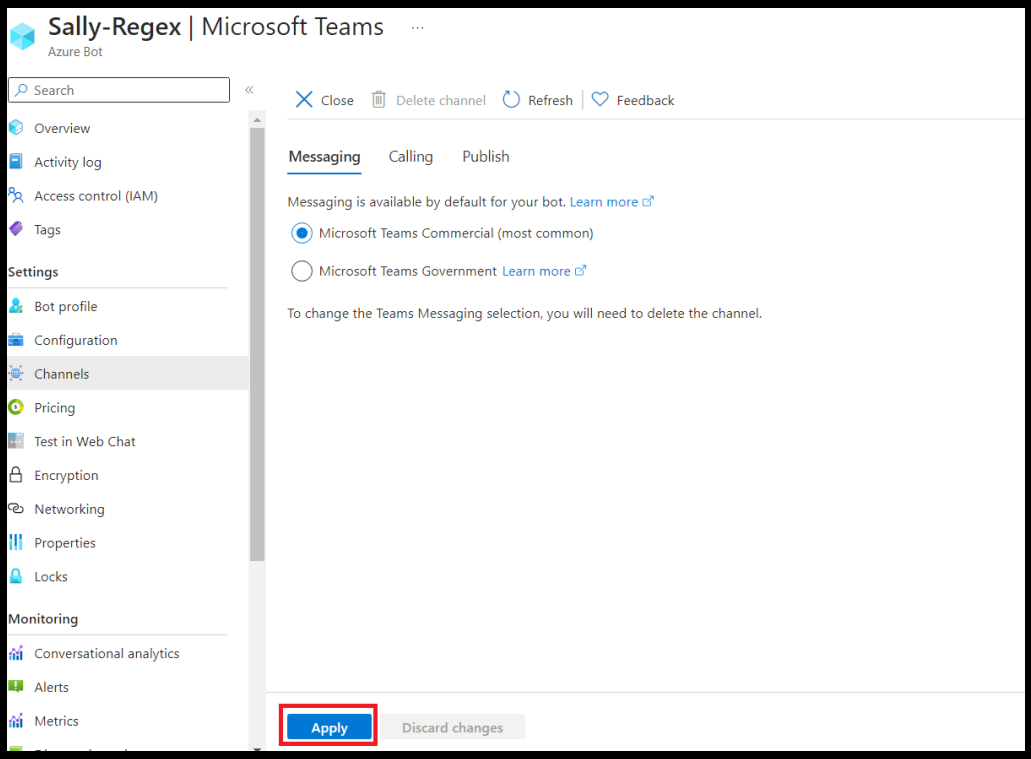
Then click channels:



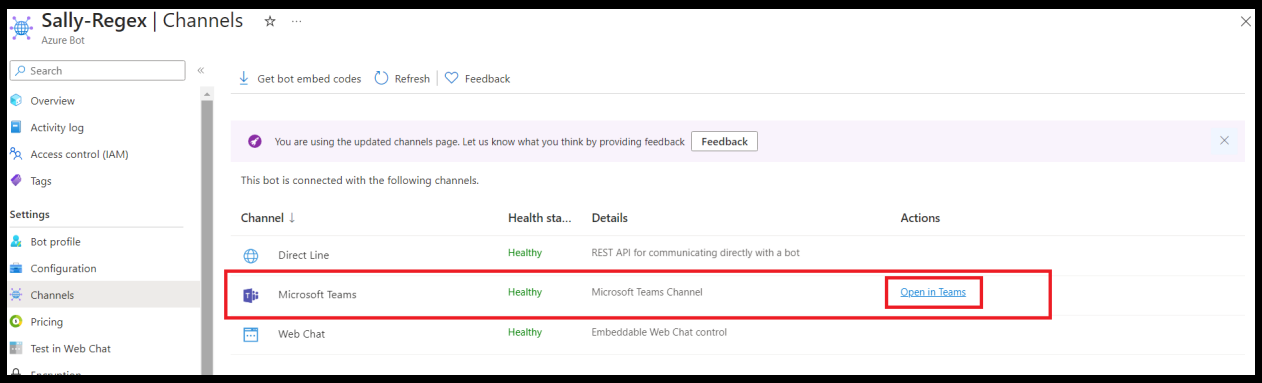
Then select Microsoft Teams under Available Channels:



Then agree to the terms of service and select Apply:



If you re-open the channels page, you should now be able to open your chatbot in teams by clicking this link:



Option 2: App

You can also deploy your chatbot as an app into Teams, which makes it easier to open conversations with it. Deploying a bot created in the Bot Framework Composer to Microsoft Teams as an app involves several steps.

Prerequisites:

- Bot Framework Composer
- Azure Account
- Microsoft Teams Developer Account

Step 1: Configure Bot Settings

After publishing your bot, you need to configure its settings:

- In the Bot Channels Registration page, under "Settings," click on "OAuth & Permissions."
- In the "Token Exchange" section, enable the "Use the same Microsoft App ID for token exchange" option.
- Under "OAuth Connection Settings," click "Add Setting" and configure the settings for your OAuth connection if your bot requires authentication.

Step 2: Add the Bot to Microsoft Teams

Now, you need to add your bot to a Microsoft Teams app:

- Go to the Microsoft Teams Developer Portal.
- Click on "Create a new app."
- Fill in the necessary details about your app, including the app name, short description, and publisher name.
- Under "App details," add an app icon and specify other details.
- In the "Capabilities" section, select "Bots."
- Click on "Set up" to configure the bot settings.
- In the "Bot Endpoint" field, enter the endpoint URL of your bot (the same URL you used during registration).
- In the "Messaging endpoint" field, enter the same bot endpoint URL.
- In the "Microsoft App ID" field, enter the App ID you obtained during registration.
- Click "Save."

Step 3: Package Your Bot

To package your bot for deployment to Teams:

- In the Microsoft Teams Developer Portal, navigate to the "Package" section.
- Click "Create a package."
- Fill in the required information, including the package name and version.
- Upload the manifest file of your bot. This file can be generated using Bot Framework Composer or manually created.
- Click "Create."

Step 4: Install Your Bot in Teams

Now that you have a packaged app, you can install it in your Microsoft Teams environment:

- In the Microsoft Teams Developer Portal, go to the "Test and distribute" section.
- Under "Install your app," click "Install" to add your bot to Teams for testing.
- Review and confirm the installation.
- Your bot should now be available in Teams. You can test it in a chat or channel.

For more information please refer to:
https://microsoft.github.io/botframework-solutions/clients-and-channels/tutorials/enable-teams/4-create-app-manifest/

https://learn.microsoft.com/en-us/microsoftteams/platform/concepts/build-and-test/teams-developer-portal?source=recommendations

https://learn.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/apps-upload

Part 7: Regex and Adaptive cards.

For more information about regular expressions and adaptive cards please refer to these sections in the Appendix.

Implementing via. Language Studio

Language studio is an easy and cost effective way of delivering a question-answering chatbot with natural language processing.

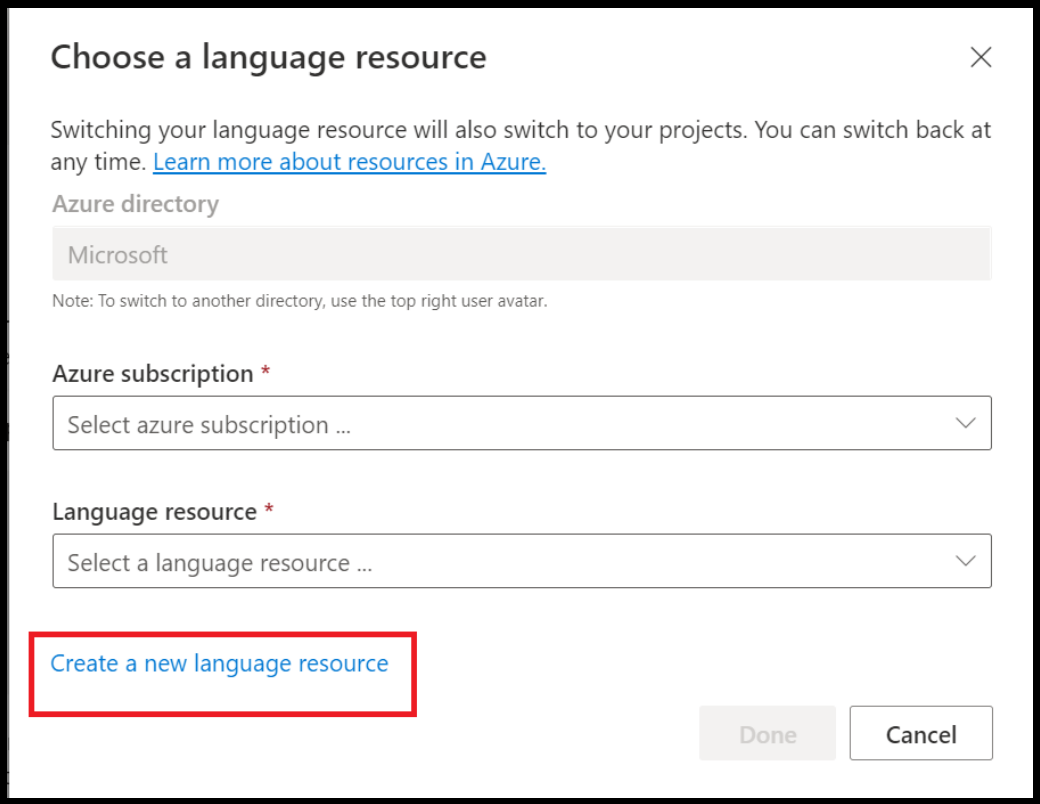
Step-by-step guide to building bots via Language Studio:

Step 1:

Login to Azure and go to https://language.cognitive.azure.com/

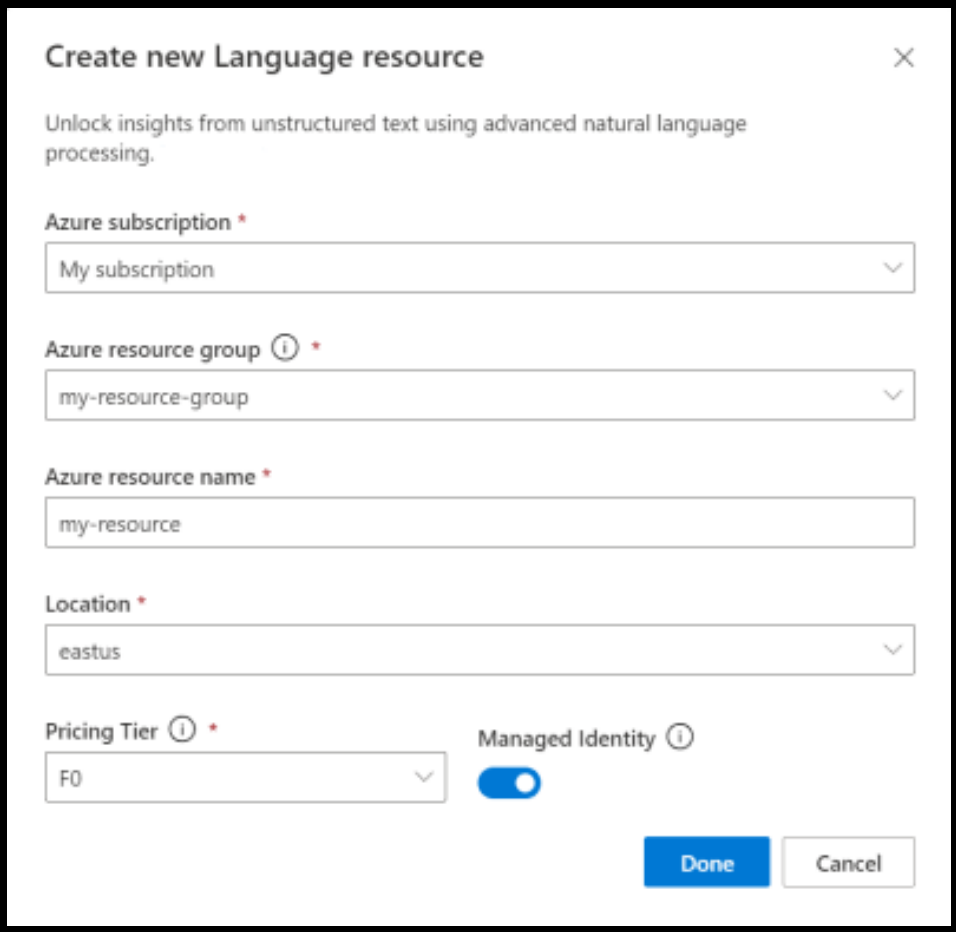
Step 2:

When you first go to this page you will be prompted to choose a language resource. Click “Create a new language resource”.



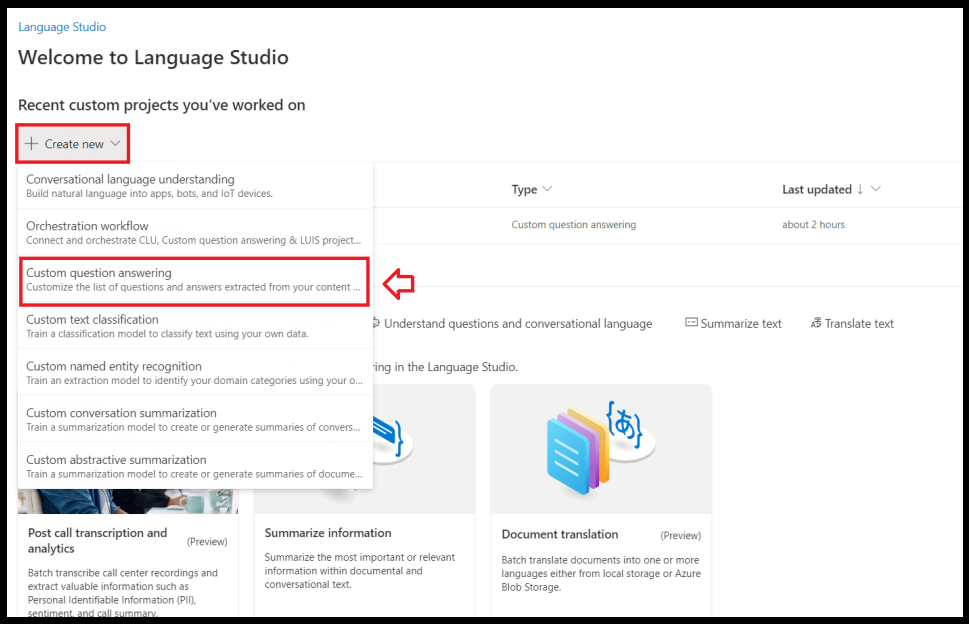
Step 3:

Fill in the following form (Australia southeast for location is recommended):



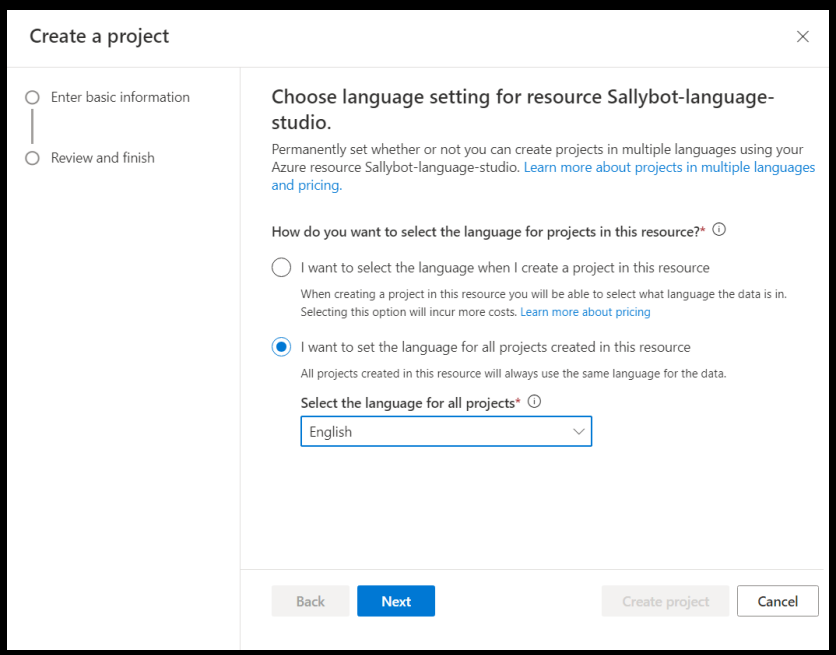
Step 4:

Once you have set up your language resource, go to the language studio page and click “create new” then select “custom question answering”.



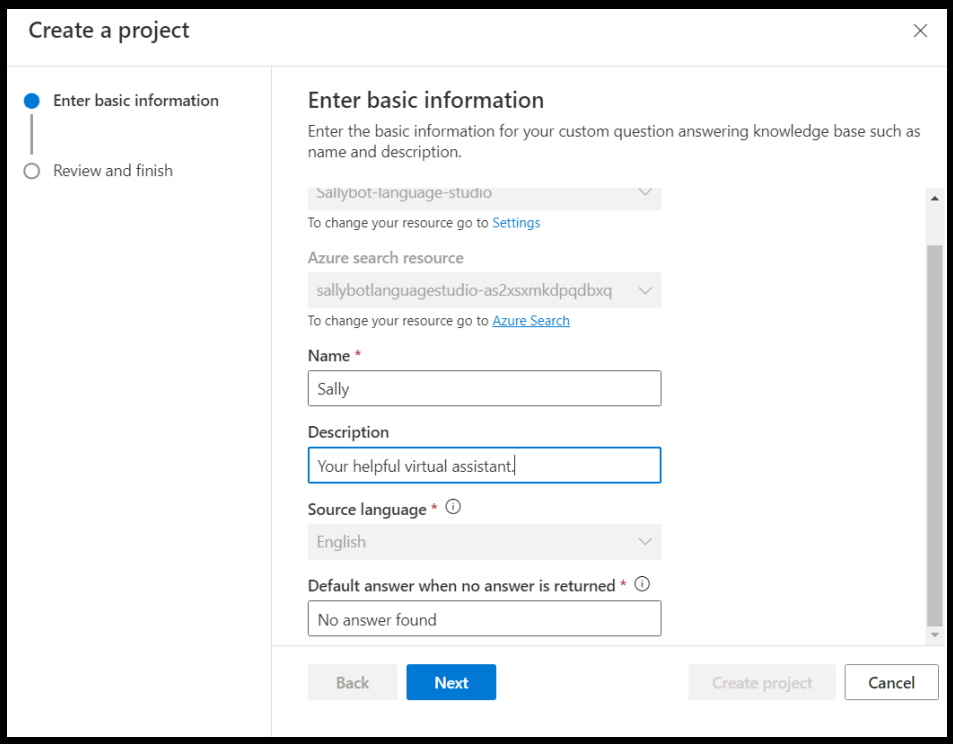
Step 5:

Select English as pictured below and click “Next”.



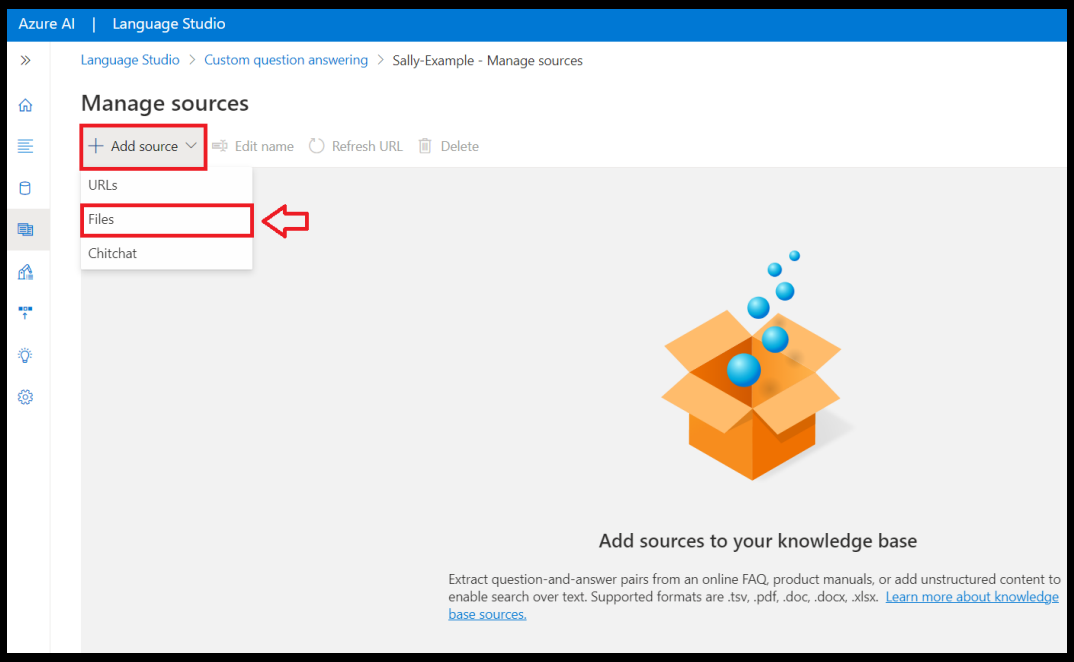
Step 6:

Fill out the name and description (and default error message if desired) and click “Next”. Then, finally click “Create Project”.



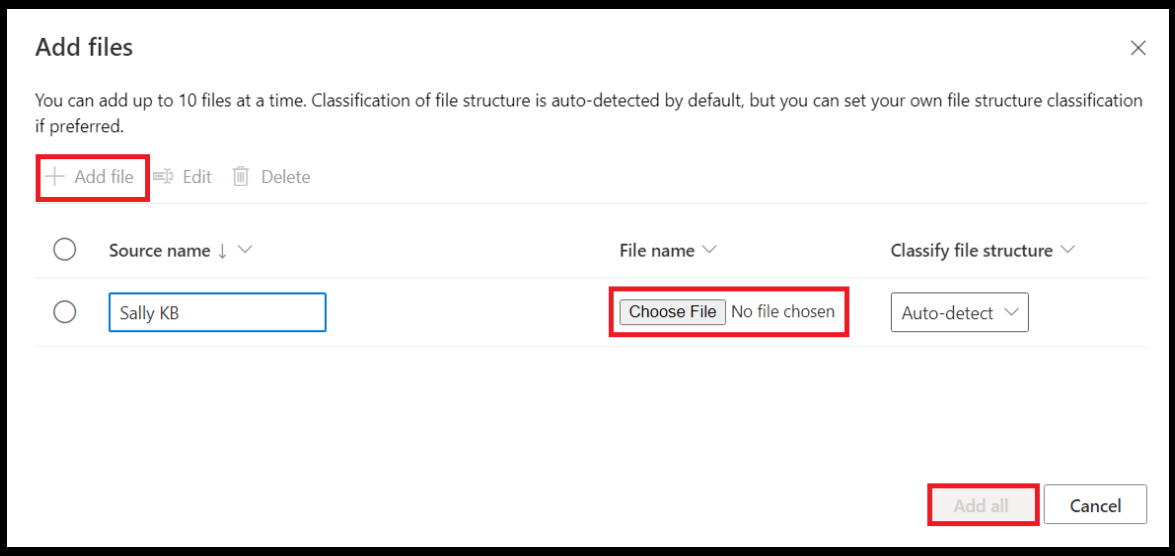
Step 7:

Click into your project and import a knowledge base from a source file as pictured below:



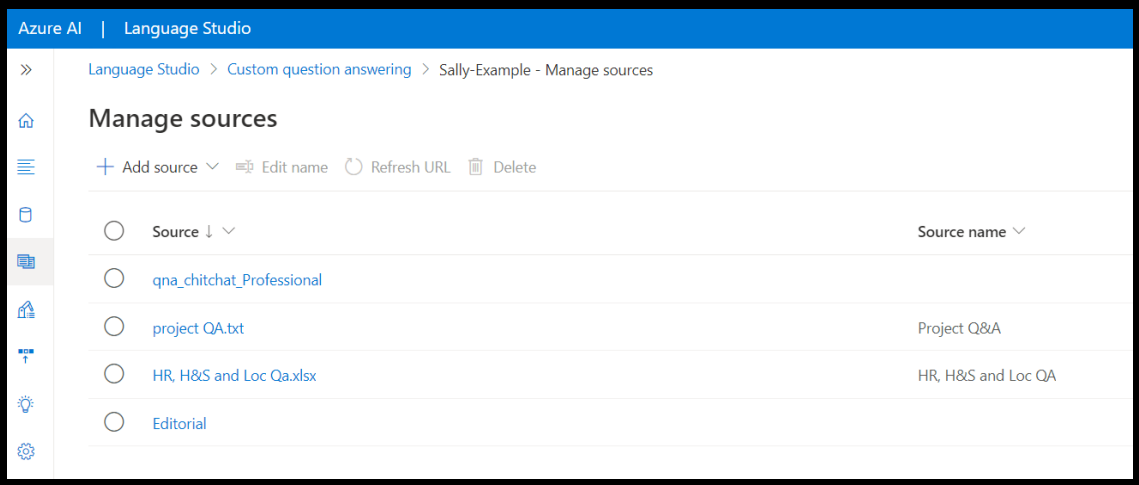
Step 8:

Click “Add file”, name your import, and select the file you would like to import. Several file types can be used such as .txt, .xls, .tsv, .pdf.



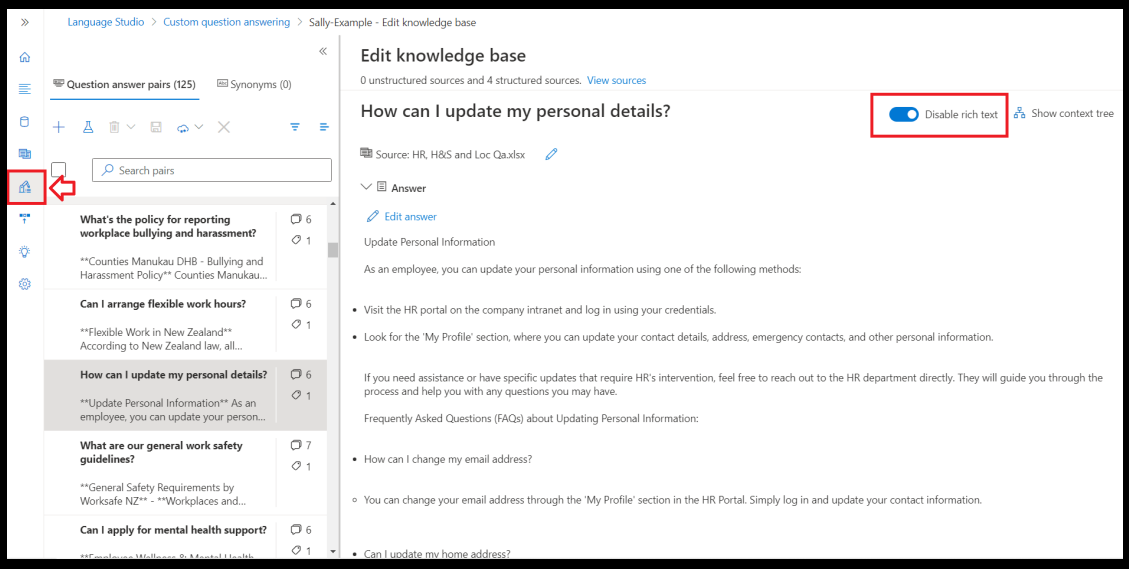
One of the easiest ways to structure new question/answer pairs is to create a spreadsheet in excel with one column for questions and a corresponding column for answers. Language Studio's automated parsing function is able to consistently determine Q&A pairs using this method.

You should now have a number of sources, similar to those pictured below:



Step 9:

By clicking the knowledge base icon identified in the sidebar, or by clicking on a source, you can edit the knowledge base of your bot.



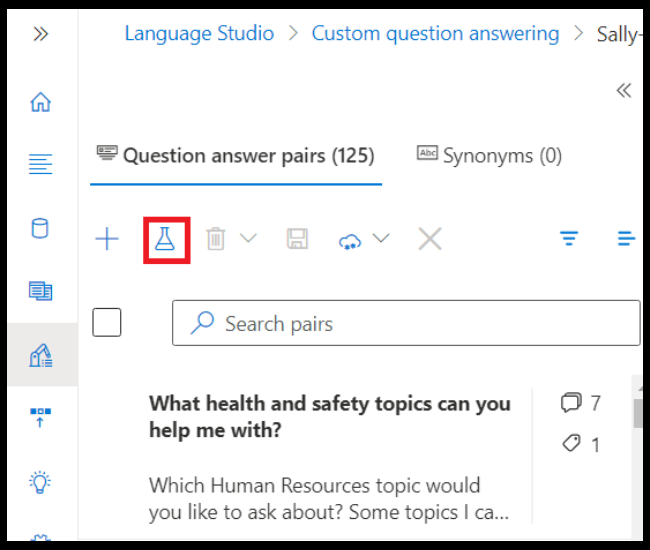
By enabling rich text, you can also format your bots output as desired.

If you scroll down on a knowledge base entry, you will note that you can also add “alternative questions”. These questions represent the different ways you can ask about the same topic. It is highly recommended that for every knowledge base entry you have at minimum 5 alternative questions, adding more as needed. This is because the first 5 alternative questions in the list for each entry are considered in the core ranking of intents (which is how the bot scores and selects a best response), while any further alternative questions are useful for exact matches. For more information on this topic, please refer to:
https://learn.microsoft.com/en-us/azure/ai-services/language-service/question-answering/concepts/best-practices

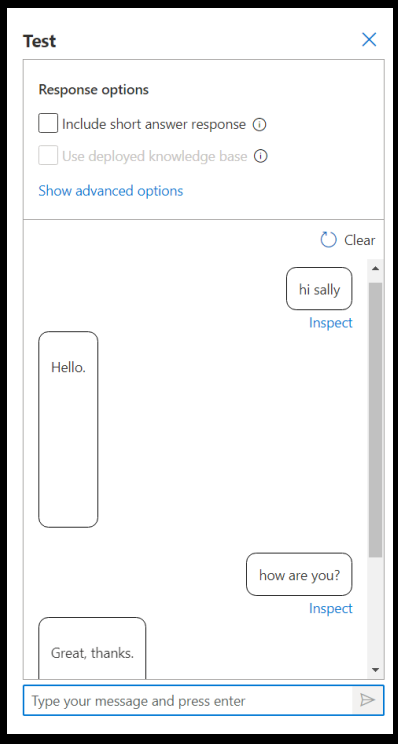
Make sure to save any changes made to the knowledge base by clicking the save icon (floppy disk icon) in the top left of the UI.

Step 10:

Now that we have a knowledge base, we can test it by clicking the vial icon in the top left of the UI.

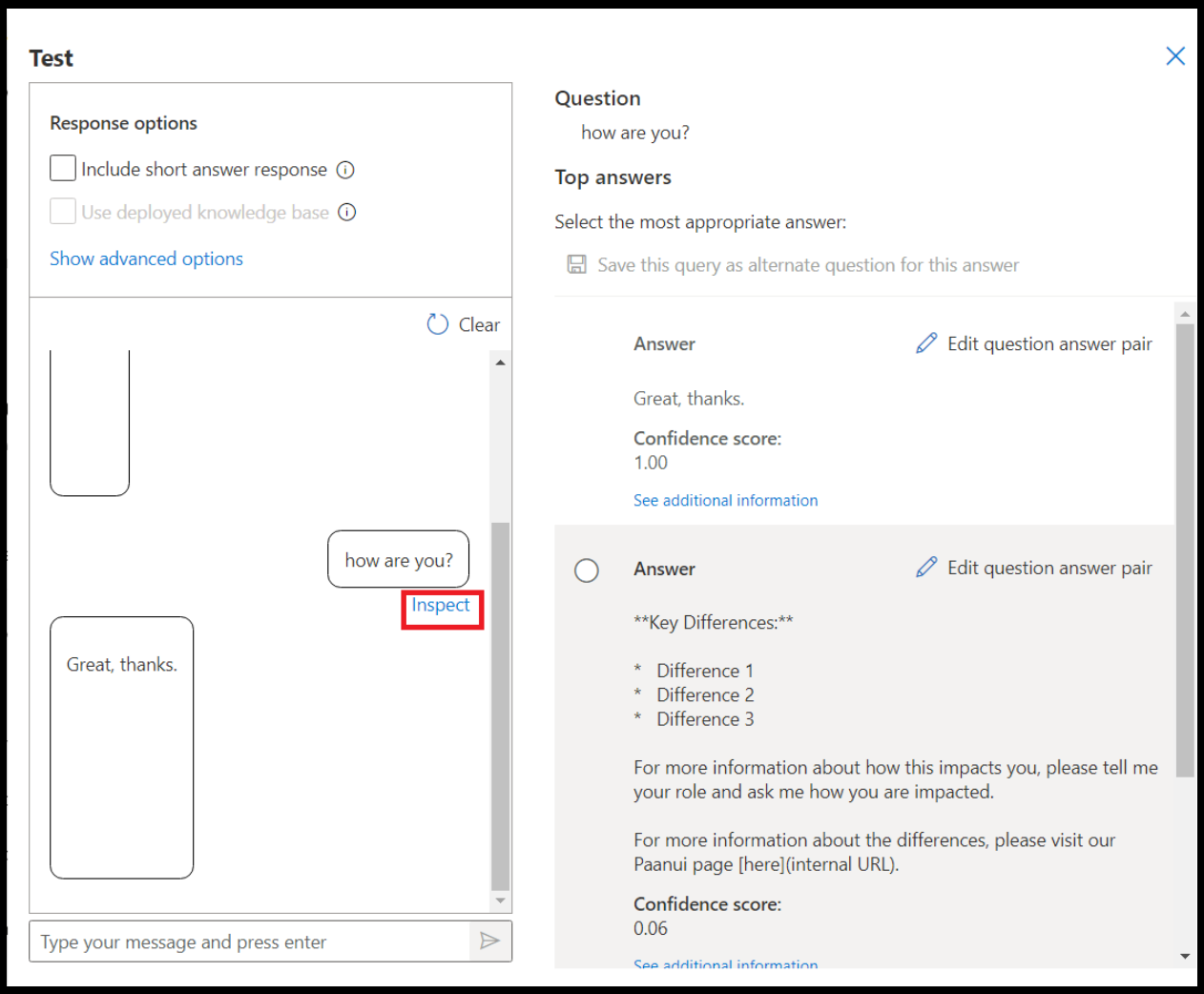


This opens up a chat pane on the right side of the interface.

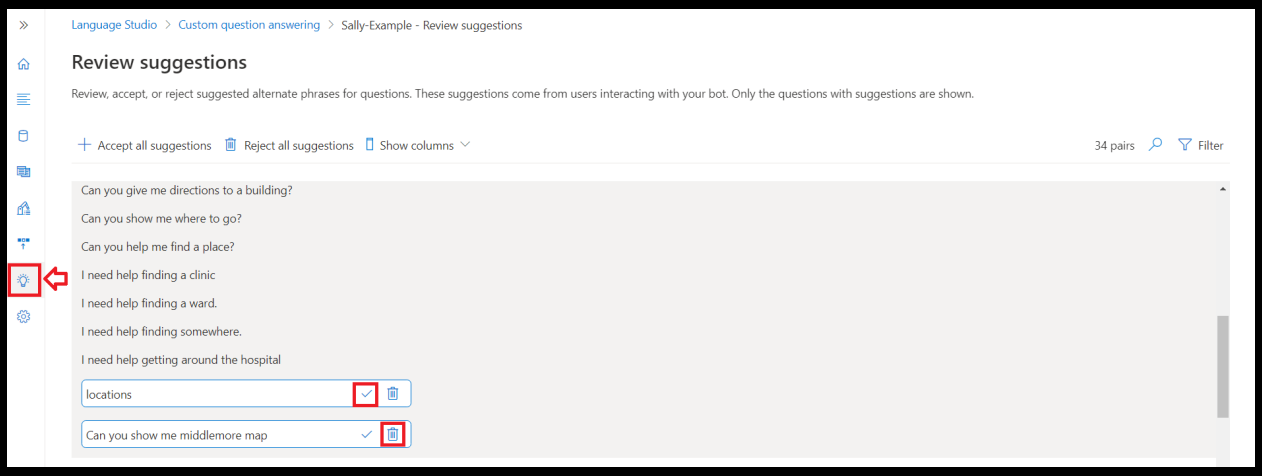


You can use this chat to test the correctness of your bots responses.

If the bot produces an unexpected response, you can click inspect to determine which intent was picked up and the confidence score rating of that intent (0 being the lowest and 1.0 being the highest). This allows us to debug and improve our knowledge base.



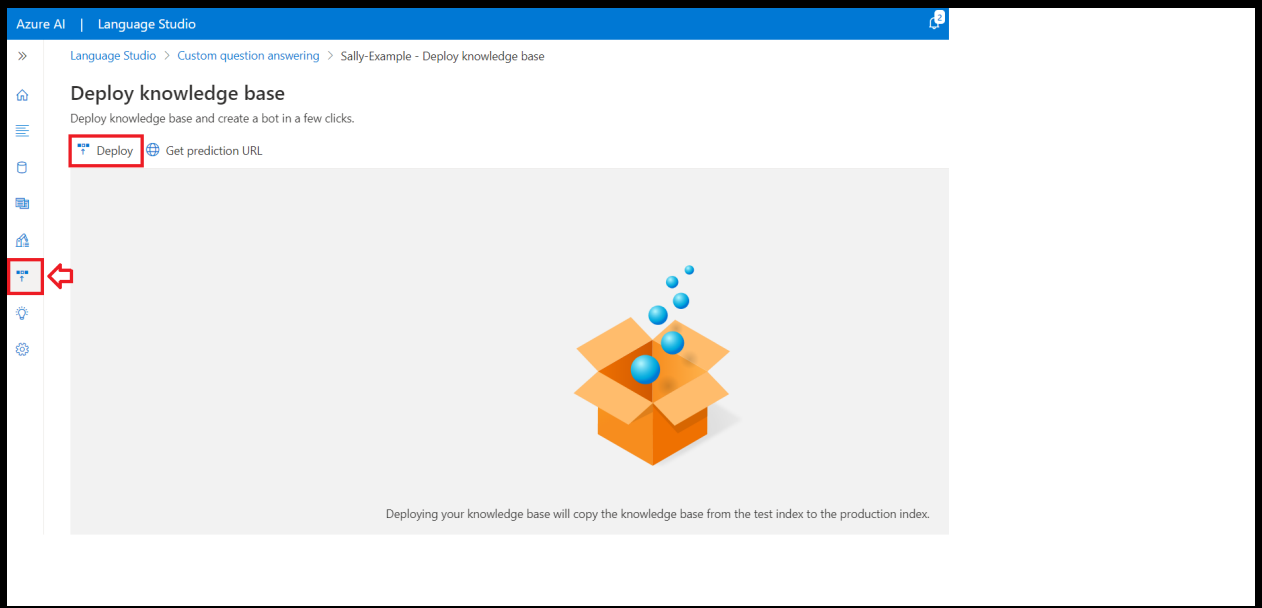
Additionally, if we click the lightbulb icon on the left as pictured below:



This provides us suggestions of alternate questions for our knowledge base entries. This can be a useful tool, as you can quickly confirm or reject adding suggested questions which are generated based on previous bot conversations.

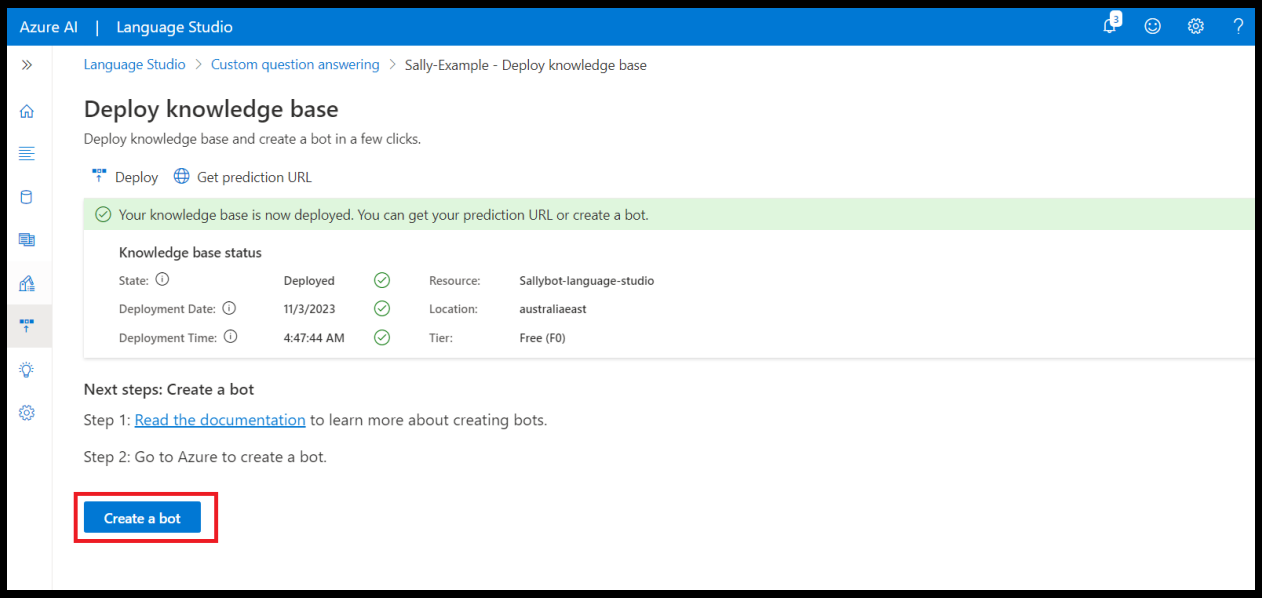
Step 11:

Once you are ready to deploy your app, click the icon pictured below on the left hand side and click “Deploy”.



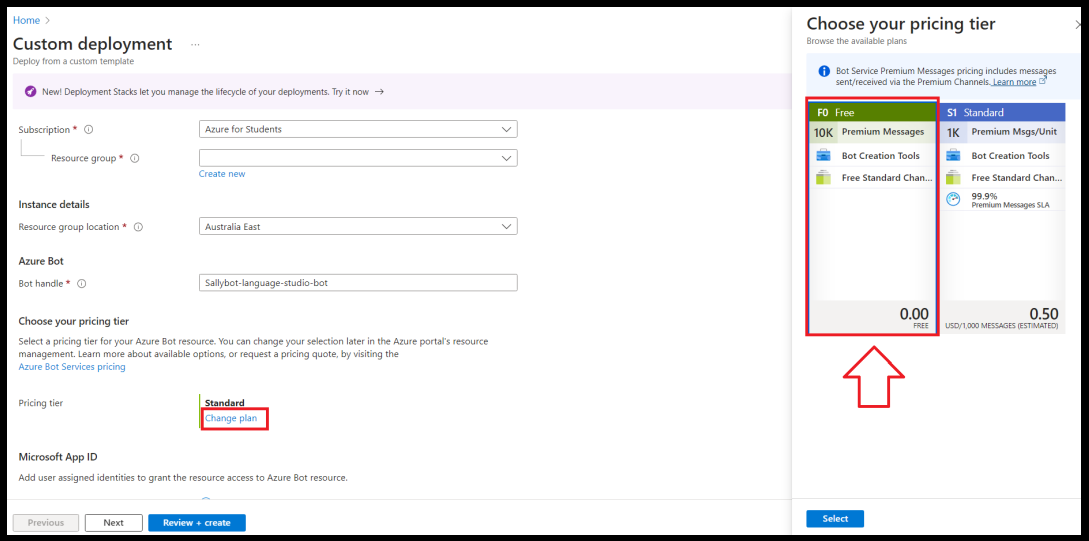
Step 12:

Once your knowledge base is deployed, select “Create a Bot”.



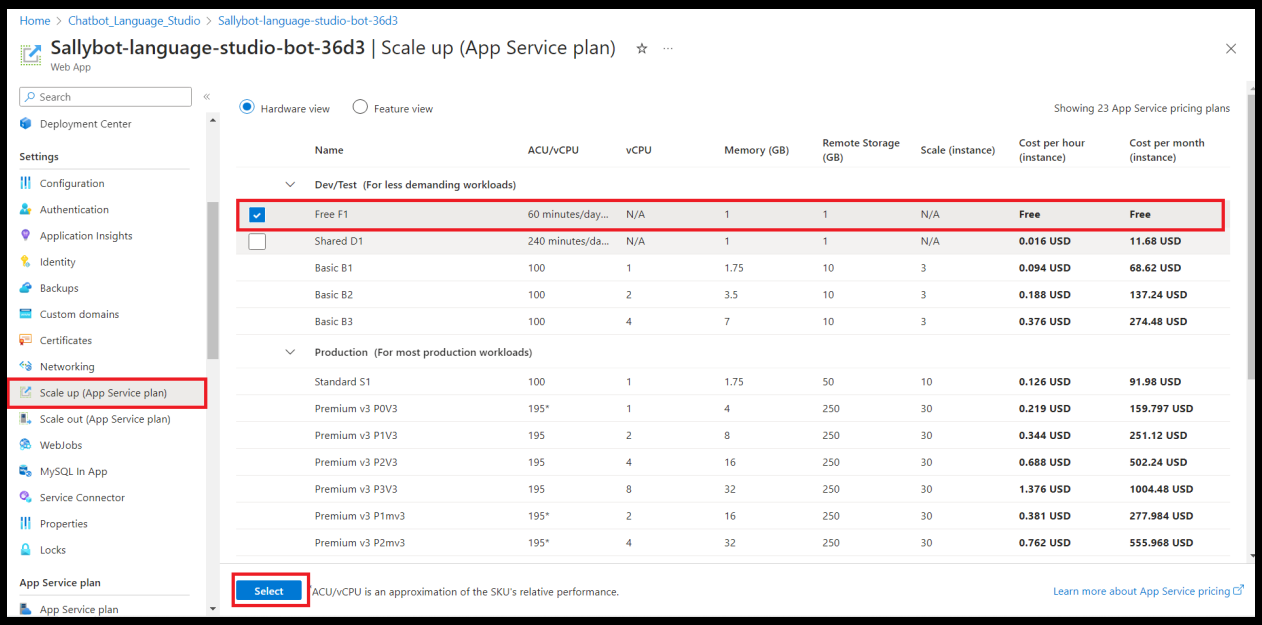
Step 13:

Fill in the form. Create a new resource group and give the bot a handle (name). Make sure to select the free plan as pictured below, as we do not need a standard plan for our implementation, given that Teams is not a premium channel.



Step 14:

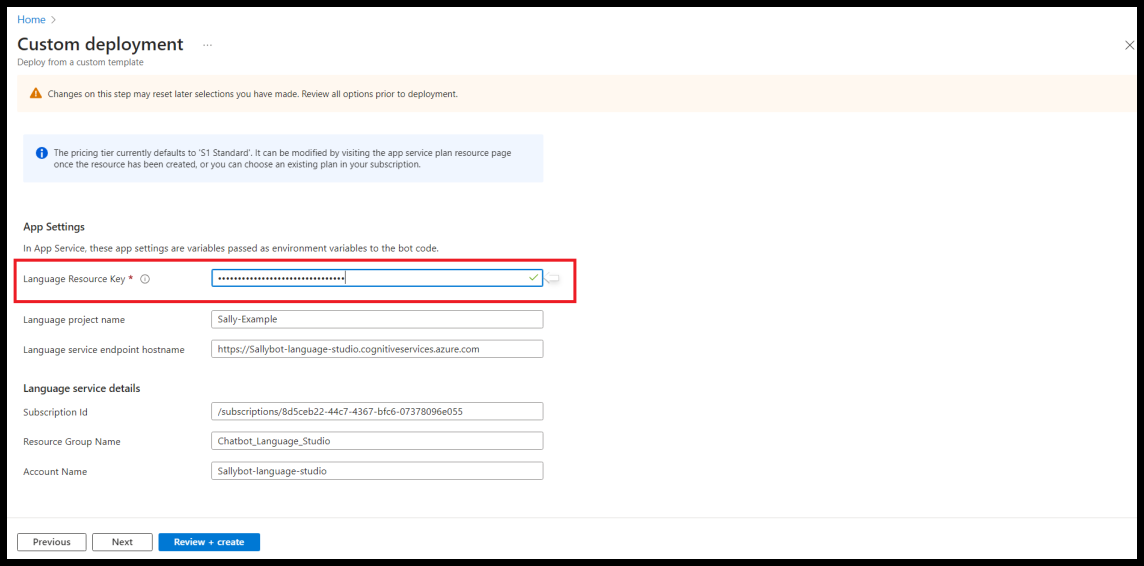
On the next page, you can make an App Service Plan. **VERY IMPORTANT NOTE: When you do this, the default App Service Plan will be set to Standard. This plan costs roughly $5.00 NZD per day and costs begin accumulating straight away, so if you intend to use the free version of the app service plan, please go to this App Service resource or App Service Plan resource inside the Azure console right after provisioning and change to the free plan using the menus pictured below:**



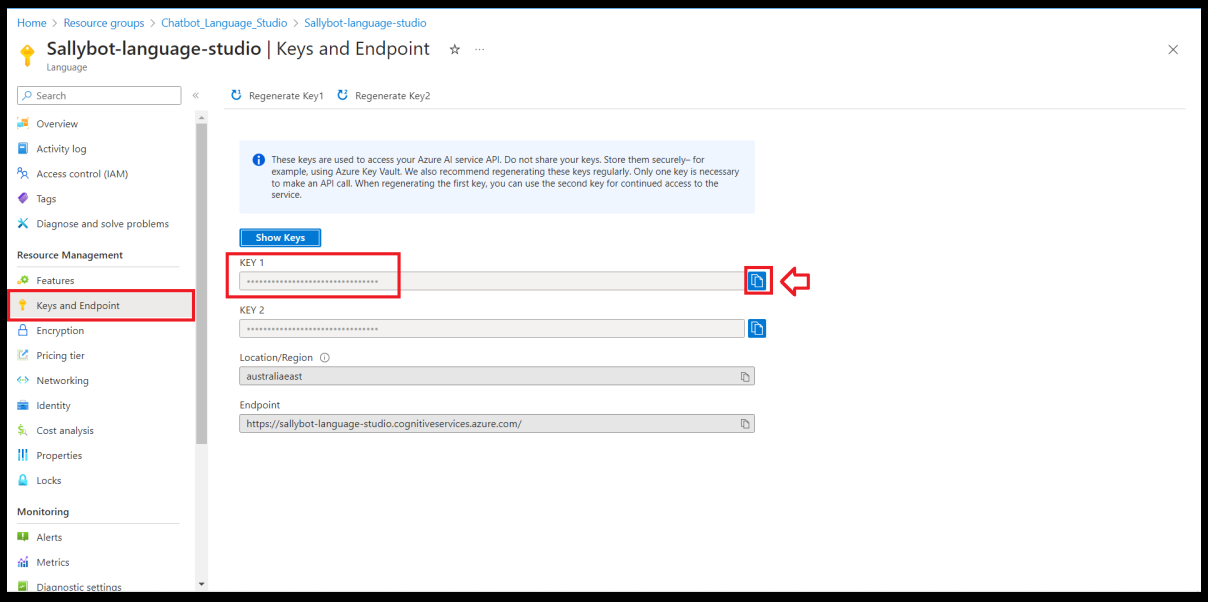
Please also ensure that your Azure Bot resource is on plan “F0” as the free plan is right for a Teams implementation.

Step 15:

After making an app service plan, scroll down the page to see the field for a “Language Resource Key” as seen below.



This field sometimes autofills correctly, but if it doesn’t, you can find and copy your key from inside the Language resource in your azure console as pictured below:



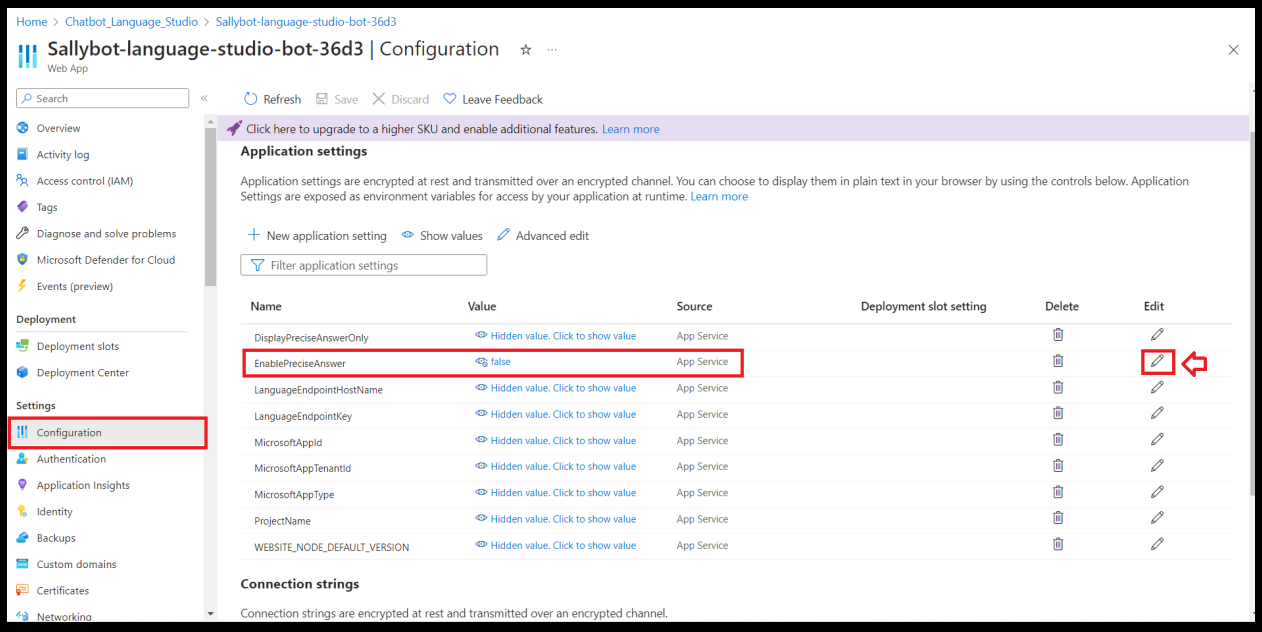
After this, you may review your information and then click “Create”.

Reminder: Check your resource plans and resource group cost analysis regularly to ensure that costs are within the expected range.

Step 16:

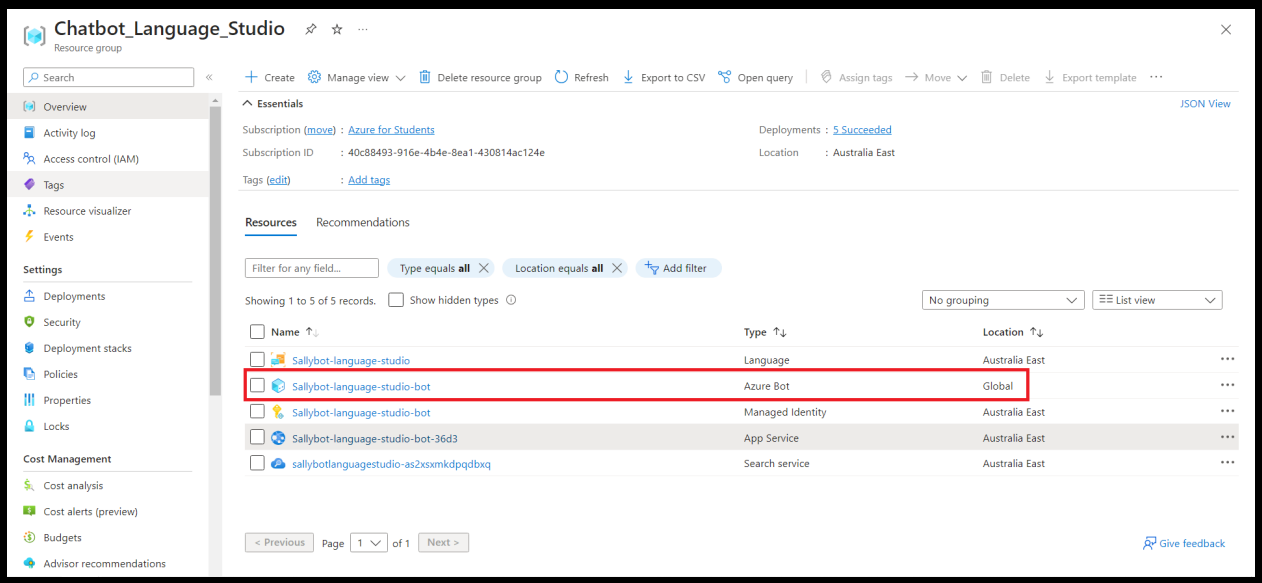
Now that you have created a bot from your deployed knowledge base, any time you make new updates to the knowledge base in Language studio you can simply save the changes and click “Deploy” and your bot will be updated automatically with the new knowledge base.

In order for our bot to output correctly in teams, we need to turn off the “precise answering” function which is enabled by default. This is achieved by going into your App Service inside your resource group and changing the value of “EnablePreciseAnswer” to “false” under the configuration menu as pictured below:

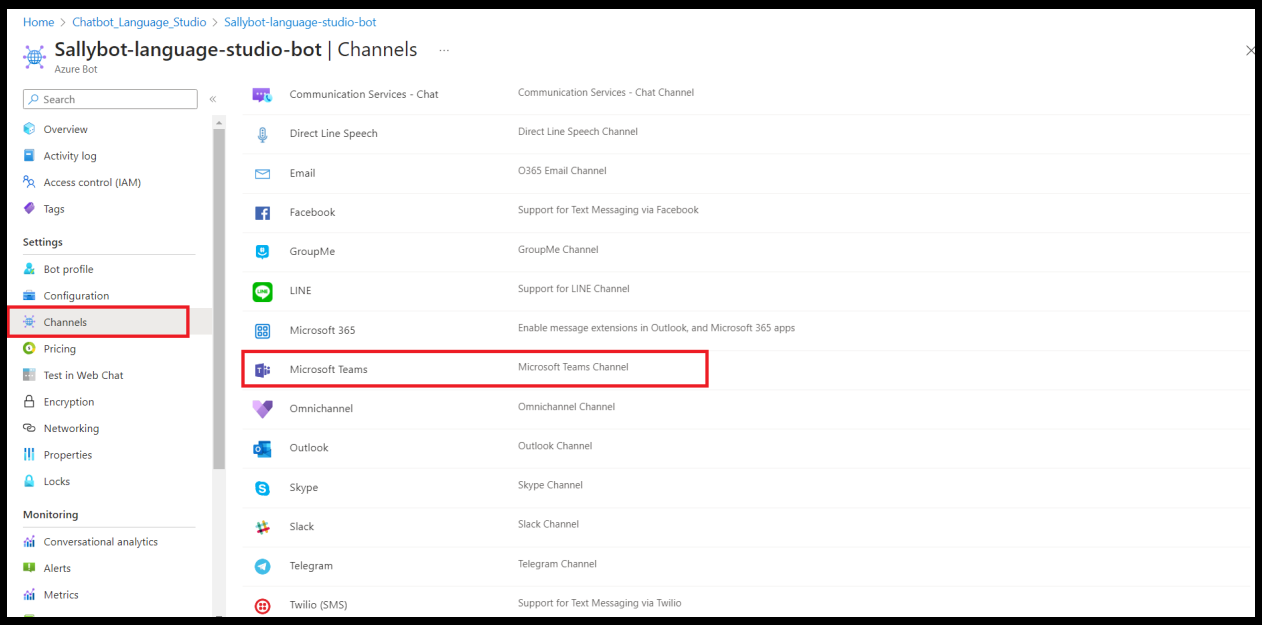


Step 17:

Once you have done this, you can select your Azure bot resource as pictured below:

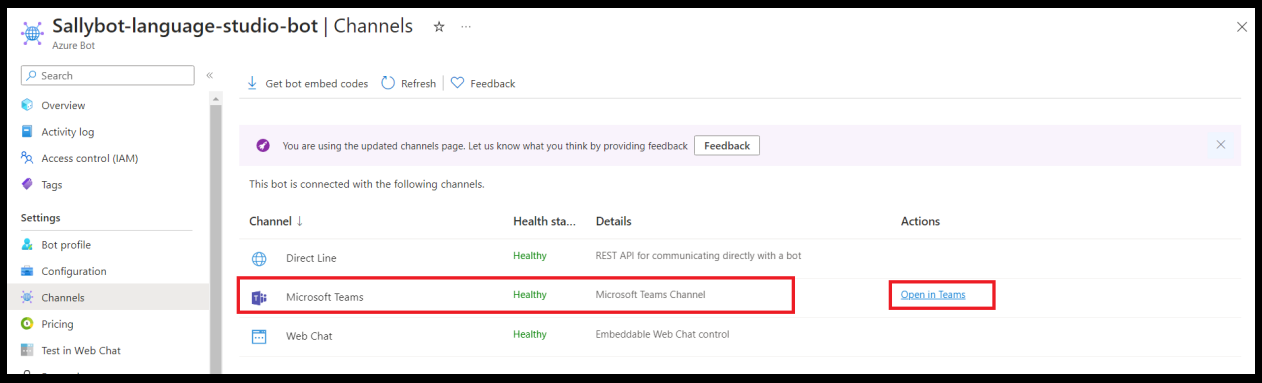


Go into the “Channels” menu and select “Microsoft Teams” under “Available channels”.



Agree to the terms of service and click “Apply”.

Now in the channels menu you can click “Open in Teams” to test your bot in the teams environment.



Step 18:

To deploy this bot as an app in teams, please refer to the following documentation:

https://microsoft.github.io/botframework-solutions/clients-and-channels/tutorials/enable-teams/4-create-app-manifest/

https://learn.microsoft.com/en-us/microsoftteams/platform/concepts/build-and-test/teams-developer-portal?source=recommendations

https://learn.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/apps-upload

APPENDIX

Appendix Section 1

Regex Patterns

Regex explanation:

REGEX, or Regular Expressions, is a sequence of characters that define a search pattern. This search pattern can then be used in string search algorithms, string matching algorithms, for "find" or "find and replace" operations, for input validations, etc.

REGEX is a universal concept used across most of the programming languages like Python, Javascript or C++. It's not specific to Azure Bot Framework SDK or Bot Framework Composer, but is used within them.

In the context of Azure Bot Framework SDK and Bot Framework Composer, REGEX plays a vital role in natural language processing and intent recognition. When we want the bot to understand natural language, we can't manually input every single possible sentence the user might say; it's practically impossible. This is where REGEX comes into play where we can create search patterns to capture the user's intent.

Let's examine a few important examples of symbols which can build REGEX patterns:

1. **"." (Dot) :** This symbol represents any single character except a newline character.

Examples:

- If the REGEX is "a.", It can match "ab", "ac", "ad", etc. But not "a" or "a\n".

- If the REGEX is "do.", It can match "dot", "dog", "dol" etc. Any three-letter word starting with "do" would trigger it.

- In the Azure Bot, if we want to locate any three-letter word in the user's expression, "." will come in handy.

2. **"*" (Asterisk) :** This symbol means zero or more occurrences of the pattern left to it. This can apply to a single character, or an entire section of the expression in parentheses.

Examples:

- If the REGEX is "ba*", It can match "b", "ba", "baa", "baaa" and so on.

- If the REGEX is "(ba)*", It can match "", "ba", "baba", "bababa" etc. Please note, it also matches an empty string because * means 0 or more.

- In Azure Bot, we can utilize "*" to capture repetitive phrases. For example, if we want to capture the sentiment of the user and they keep saying "very" like "I am very very happy", we can use "*".

3. **"?" (Question Mark) :** This means zero or one occurrences of the pattern left to it.

Examples:

- If the REGEX is "ba?", It matches "b" and "ba".

- In Azure Bot, when the user's input comes with optional words, we can use "?".

4. **"(?i)" :** This symbol is used to make the regex match case insensitive.

Examples:

- If the REGEX is "(?i)yes", It matches "Yes", "YES", "yes", etc.

- It can be used in Azure Bot when the response from the user is not case sensitive. For example, if the bot asks for confirmation and the user can reply as "yes" or "YES" or "Yes", the bot can accept any of these.

5. **".*" (Dot Star) :** This can match zero or more of any character. This is quite powerful as it can match almost

anything, and if not used correctly, it can easily lead to over-matching.

Examples:

- If the REGEX is "a.*", It matches "a", "ab", "abcd", etc.

- If the REGEX is ".*", It matches anything or nothing - the entire input, or an empty string.

- Fine-tuning REGEX with ".*" in Azure Bot is extremely necessary as it can catch everything the user says. An excellent application could be an error message or a default message when the bot does not understand the user.

6. **"|" (vertical bar): It's used like a boolean OR and matches the pattern before or the pattern after it.**

Example:

- The REGEX "abc|def" will match either "abc" or "def".

The benefit of using REGEX in Azure Bot Framework SDK or Bot Framework Composer lies in its robustness, flexibility, and the power it gives to process natural language, conduct intent recognition, and design dialogues. However, it is the most challenging way of building a bot with NLP-esque capabilities.

Full list of Regex Patterns:
Regular Expressions List

Short Tutorial
Source: https://gist.github.com/jacksonfdam/3000275

\ // the escape character - used to find an instance of a metacharacter like a period, brackets, etc.
. // match any character except newline
x // match any instance of x
^x // match any character except x
[x] // match any instance of x in the bracketed range - [abxyz] will match any instance of a, b, x, y, or z
| // an OR operator - [x|y] will match an instance of x or y
() // used to group sequences of characters or matches
{} // used to define numeric quantifiers
{x} // match must occur exactly x times
{x,} // match must occur at least x times
{x,y} // match must occur at least x times, but no more than y times
? // preceding match is optional or one only, same as {0,1}
* // find 0 or more of preceding match, same as {0,}
+ // find 1 or more of preceding match, same as {1,}

^ // match the beginning of the line
$ // match the end of a line

[:alpha:] // Represents an alphabetic character. Use [:alpha:]+ to find one of them.
[:digit:] // Represents a decimal digit. Use [:digit:]+ to find one of them.
[:alnum:] // Represents an alphanumeric character ([:alpha:] and [:digit:]).
[:space:] // Represents a space character (but not other whitespace characters).
[:print:] // Represents a printable character.
[:cntrl:] // Represents a nonprinting character.
[:lower:] // Represents a lowercase character if Match case is selected in Options.
[:upper:] // Represents an uppercase character if Match case is selected in Options.

\d // matches a digit, same as [0-9]
\D // matches a non-digit, same as [^0-9]
\s // matches a whitespace character (space, tab, newline, etc.)
\S // matches a non-whitespace character
\w // matches a word character
\W // matches a non-word character
\b // matches a word-boundary (NOTE: within a class, matches a backspace)
\B // matches a non-wordboundary

Sources:

http://regexlib.com

http://net.tutsplus.com/tutorials/other/8-regular-expressions-you-should-know

Email address

```
^[\\w\\-]+(\\.[\\w\\-]+)*@([A-Za-z0-9-]+\\.)+[A-Za-z]{2,4}$
```

```
^[\w-]+(\.[\w-]+)*@([a-z0-9-]+(\.[a-z0-9-]+)*?\.[a-z]{2,6}|(\d{1,3}\.){3}\d{1,3})(:\d{4})?$
```

```
^([\w\.*\-*]+@([\w]\.*\-*)+[a-zA-Z]{2,9}(\s*;\s*[\w\.*\-*]+@([\w]\.*\-*)+[a-zA-Z]{2,9})*)$
```

List of semi-colon seperated email addresses

```
^([a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4})*$
```

IP Address

```
^((?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))*$
```

Credit Cards

^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|6011[0-9]{12}|622((12[6-9]|1[3-9][0-9])|([2-8][0-9][0-9])|(9(([0-1][0-9])|(2[0-5])))[0-9]{10}|64[4-9][0-9]{13}|65[0-9]{14}|3(?:0[0-5]|[68][0-9])[0-9]{11}|3[47][0-9]{13})*$

Username of type test@test

```
[^@/]+@[^@/]+
```

Multiple spaces replacement

```
\\s+
```

Non-alphanumeric replacement

```
[^a-zA-Z0-9]
```

Blank line

```
^$
```

Positive integers

```
^[1-9]+[0-9]*$
```

Positive decimal values

```
(^\d*\.?\d*[0-9]+\d*$)|(^[0-9]+\d*\.\d*$)
```

Percentage (2 decimal places)

^-?[0-9]{0,2}(\.[0-9]{1,2})?$|^-?(100)(\.[0]{1,2})?$

State abbreviation

[A-Z][A-Z] //you may choose to put spaces either before or after the regex.

Phone Numbers

(^\+[0-9]{2}|^\+[0-9]{2}\(0\)|^\(\+[0-9]{2}\)\(0\)|^00[0-9]{2}|^0)([0-9]{9}$|[0-9\-\s]{10}$)

City, State abbreviation

.*, [A-Z][A-Z]

Zip Code

[0-9]\{5\}(-[0-9]\{4\})? //84094 or 84094-1234

Social security number, such as: ###-##-####

[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}

Dollar amounts, specified with a leading $ symbol

\$[0-9]*.[0-9][0-9]

DATE

[0-9]\{4\}-[0-9]\{2\}-[0-9]\{2\} //2003-08-06

[A-Z][a-z][a-z] [0-9][0-9]*, [0-9]\{4\} //Jan 3, 2003

^(\d{1,2})\/(\d{1,2})\/(\d{2}|(19|20)\d{2})$ //DD/MM/YY or DD/MM/YYYY or MM/DD/YY or MM/DD/YYYY

**HTML Tags except **

<(?>/?)(?!p).+?>

Font Tags Replacement

<(FONT|font)([ ]([a-zA-Z]+)=("|')[^"\']+("|'))*[^>]+>([^<]+)(</FONT>|

URL

^http(s)?:\/\/((\d+\.\d+\.\d+\.\d+)|(([\w-]+\.)+([a-z,A-Z][\w-]*)))(:[1-9][0-9]*)?(\/([\w-.\/:%+@&=]+[\w- .\/?:%+@&=]*)?)?(#(.*))?$/i

Appendix section 2

Adaptive Cards

Microsoft Adaptive Cards: An Overview

Microsoft Adaptive Cards are a versatile platform-independent format for presenting information in a user-friendly and customizable way. They provide a standardized way to represent content in a structured format that can be easily rendered and customized across various platforms and devices. Adaptive Cards are particularly popular within the Microsoft ecosystem but can be used in many other applications as well.

Adaptive Cards are typically used for scenarios like:

- **Notifications:** Displaying messages and updates.
- **Conversational Interfaces:** Enhancing chatbots and virtual assistants by presenting information in a customizable format.
- **Task Flows:** Guiding users through multi-step processes by breaking them down into individual card-based steps.

An Adaptive Card consists of several key components:

- **Card Schema:** The card schema defines the structure and content of the card. It includes elements like text, images, buttons, and input fields. The schema is written in JSON format and defines the overall structure of the card.
- **Host Application:** The host application is the platform or application where the Adaptive Card is rendered. This could be a messaging app, a

web page, a mobile app, or any other software that supports Adaptive Cards.

- **Card Renderer:** The card renderer is responsible for interpreting the card schema and rendering it in the host application. Each platform or application may have its own card renderer, which ensures that the card is displayed correctly and with platform-specific styling.
- **Actions:** Actions are interactive elements within the card, such as buttons or input fields, that allow users to perform actions or submit data. Adaptive Cards support a variety of actions, including opening URLs, submitting data, and showing additional cards.
- **Templates:** Templates are reusable card schemas with placeholders for dynamic data. They make it easy to create cards with variable content, such as displaying user-specific information or dynamic data from an external source.

Formatting Options in Adaptive Cards

Adaptive Cards offer a wide range of formatting options to customize the appearance and behavior of the card. Here are some of the key formatting options available:

1. Text Formatting: You can format text in Adaptive Cards using Markdown. This allows you to apply styling to text elements, including bold, italic, underline, and links. For example:

json

"text": "This is *bold* and this is [a link](https://example.com)."

2. Images and Media: You can include images and media in Adaptive Cards to make them more visually appealing. You can specify the image URL, alt text, size, and style.

json
Copy code
"type": "Image",
"url": "https://example.com/image.png",
"altText": "Image Alt Text",
"size": "Medium",
"style": "Person"

3. Lists: Adaptive Cards support both ordered and unordered lists. You can create bullet points or numbered lists to present information in a structured way.

json

```
"type": "TextBlock",
"text": "To-do List:",
"size": "Medium",
"weight": "Bolder"
```

4. Buttons and Actions: Buttons are interactive elements in Adaptive Cards that allow users to trigger actions. You can customize button text, style, and the action it performs.

json

```
"type": "Action.Submit",
"title": "Submit",
"style": "Positive"
```

NOTE: When implementing a bot in MS teams, actions such as Action submit must be formatted differently as there must be an MS teams field to handle the event for intent recognition purposes. It will look something like this:

```
"type": "Action.Submit",
"title": "Human Resources",
"data": {
"msteams": {
  "type": "imBack",
  "value": "Human Resources"
  }
}
```

5. Input Fields: You can add input fields to collect user input. Adaptive Cards support text input, date pickers, and other input types.

json

```
"type": "Input.Text",
"id": "userInput",
"placeholder": "Enter your name",
"maxLength": 50
```

6. Containers: Containers are used to group elements together and control their layout. You can use containers to create columns, organize content, and control spacing.

json

```
"type": "Container",
"items": [
  {
    "type": "TextBlock",
    "text": "Item 1"
  },
  {
    "type": "TextBlock",
    "text": "Item 2"
  }
]
```

7. Conditional Visibility: You can control the visibility of card elements based on conditions. This allows you to show or hide content dynamically.

json

```
"type": "TextBlock",
"text": "This is only visible if a certain condition is met.",
"isVisible": true
```

8. Styles and Themes: Adaptive Cards support different styles and themes to match the branding or design of the host application. You can define custom styles or use pre-defined ones like "Positive," "Warning," or "Accent."

json

```
"style": "Positive"
```

9. Localization: You can make your Adaptive Cards multilingual by providing translations for text elements. This ensures that users in different regions see content in their preferred language.

json

```
"text": {
   "en": "Hello",
   "fr": "Bonjour"
}
```

10. Customization Options for Adaptive Cards:

Customization is a key strength of Adaptive Cards, allowing you to tailor the appearance and behavior of cards to your specific needs. Here are some customization options:

10a. Branding and Styling: You can define custom styles to match your application's branding. This includes setting colors, fonts, and other visual elements. Styles can be defined at the card level or for specific elements.

json

```
"style": {
    "backgroundColor": "#0078D4",
    "fontColors": {
        "default": "#FFFFFF"
    }
}
```

10b. Themes: You can switch between different themes to adapt to different scenarios. Themes can change the overall look and feel of the card, including colors and fonts.

json

```
"theme": "Dark"
```

10c. Templates: Templates allow you to create reusable card layouts with placeholders for dynamic data. This is useful for displaying user-specific information or content from external sources.

json

"type": "TextBlock",
"text": "{{title}}",

10d. Adaptive Layout: Adaptive Cards can adapt to different screen sizes and orientations. You can specify how the card should behave on small screens or in landscape mode, ensuring a responsive user experience.

json

"fallback": "Mobile"

10e. Custom Actions: While Adaptive Cards support common actions like opening URLs or submitting data, you can also define custom actions to implement complex behaviors or integrations with your application.

json

"type": "Action.Custom",
"data": "customActionData"

10f. Data Binding: You can bind data to Adaptive Cards to dynamically update their content. This is useful for displaying real-time data or user-specific information.

json

```
"data": {
    "title": "Dynamic Title",
    "description": "This content is updated dynamically."
}
```

10g. Interactivity: Adaptive Cards support various forms of interactivity, such as showing or hiding elements based on user input, validating user input, and performing conditional actions.

json

```
"type": "Action.ToggleVisibility",
"targetElements": ["elementId"]
```

11. Extensibility: If the standard set of Adaptive Card elements and actions doesn't meet your requirements, you can extend Adaptive Cards by defining custom elements and actions.

json

```
"type": "Custom.MyElement",
"data": "customData"
```

12. Integration: You can seamlessly integrate Adaptive Cards into various platforms and applications, including Microsoft Teams, Outlook, and more. Each platform may offer specific customization options and capabilities.

Using Adaptive Cards in Microsoft Ecosystem

Adaptive Cards are widely used in the Microsoft ecosystem, including Microsoft Teams, Outlook, and Windows applications. When using Adaptive Cards within Microsoft products, you can leverage additional integration and customization options:

Microsoft Teams: You can create Adaptive Cards to enhance the user experience in Microsoft Teams by displaying interactive cards in chat conversations, channels, and as app notifications. Teams offers a range of card-specific capabilities, such as task modules for collecting user input.

Outlook: Adaptive Cards can be used to create visually engaging email messages in Outlook. You can use Adaptive Cards in Outlook messages to present information in a structured and interactive format.

Power Automate: With Power Automate (formerly Microsoft Flow), you can automate workflows that involve Adaptive Cards. For example, you can send Adaptive Cards as notifications or approval requests and respond to user interactions with the cards.

Power Apps: In Power Apps, you can create custom applications that include Adaptive Cards for data display and user interaction. This allows you to build rich user interfaces with low-code or no-code development.