Lab 3: Create interactive conversational bots for Microsoft Teams

Contents

[Exercise 1 - Creating conversational bots for Microsoft Teams 2](#_Toc33099114)

[Register a new bot in Microsoft Azure 2](#_Toc33099115)

[Register the bot with Microsoft Azure's Bot Framework 2](#_Toc33099116)

[Enable the Microsoft Teams channel for the bot 5](#_Toc33099117)

[Retrieve the bot app ID and password 6](#_Toc33099118)

[Create a client secret for the app 7](#_Toc33099119)

[Create Microsoft Teams app 9](#_Toc33099120)

[Add a bot to the project 11](#_Toc33099121)

[Expose the bot as part of the Node.js app's REST API 12](#_Toc33099122)

[Register the bot in the Microsoft Teams app 13](#_Toc33099123)

[Test the conversation bot 18](#_Toc33099124)

[Install the custom app in Microsoft Teams 19](#_Toc33099125)

[Summary 23](#_Toc33099126)

[Exercise 2- Bots in Microsoft Teams channels and group chats 24](#_Toc33099127)

[Add channel support to a conversation bot 24](#_Toc33099128)

[Update the app's configuration 24](#_Toc33099129)

[Update the bot code 25](#_Toc33099130)

[Test the conversation bot in a channel 26](#_Toc33099131)

[Reply to messages with Adaptive cards 28](#_Toc33099132)

[Test the bot updating existing messages 32](#_Toc33099133)

[Reply to message reactions 34](#_Toc33099134)

[Test the bot reacting to message reactions 34](#_Toc33099135)

[Summary 35](#_Toc33099136)

[Exercise 3- Proactive messages from bots 36](#_Toc33099137)

[Initiate a proactive message from the bot 36](#_Toc33099138)

[Test the bot sending new messages 38](#_Toc33099139)

[Summary 39](#_Toc33099140)

# Exercise 1 - Creating conversational bots for Microsoft Teams

In this exercise, you’ll learn how to create and add a new bot to a Microsoft Teams app and interact with it from the Microsoft Teams client.

## Register a new bot in Microsoft Azure

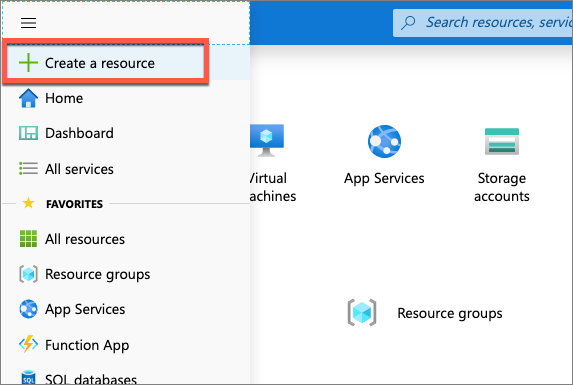
The first step is to create a new Microsoft Teams bot. Adding a bot to the Teams app involves two steps:

1. Register the bot with Microsoft Azure's Bot Framework
2. Add a bot to the project codebase

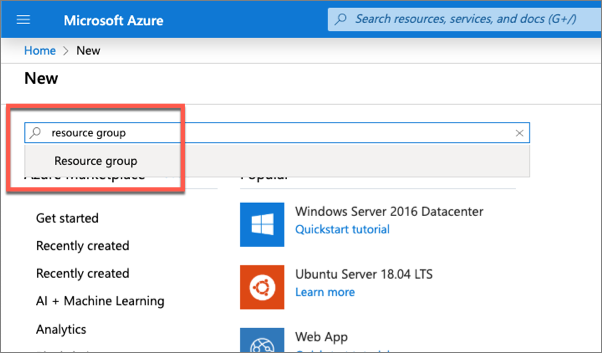
## Register the bot with Microsoft Azure's Bot Framework

Open a browser and navigate to the [Azure portal](https://portal.azure.com/). Sign in using a **Work or School Account** that has rights to create resources in your Azure subscription.

Select **Create a resource** in the left-hand navigation:

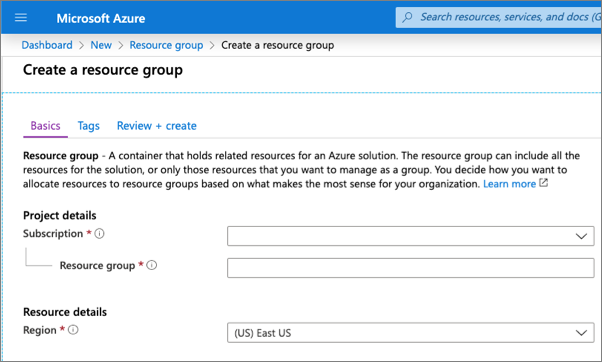


Enter **resource group** in the **Search the marketplace** input box, and select **Resource group**.



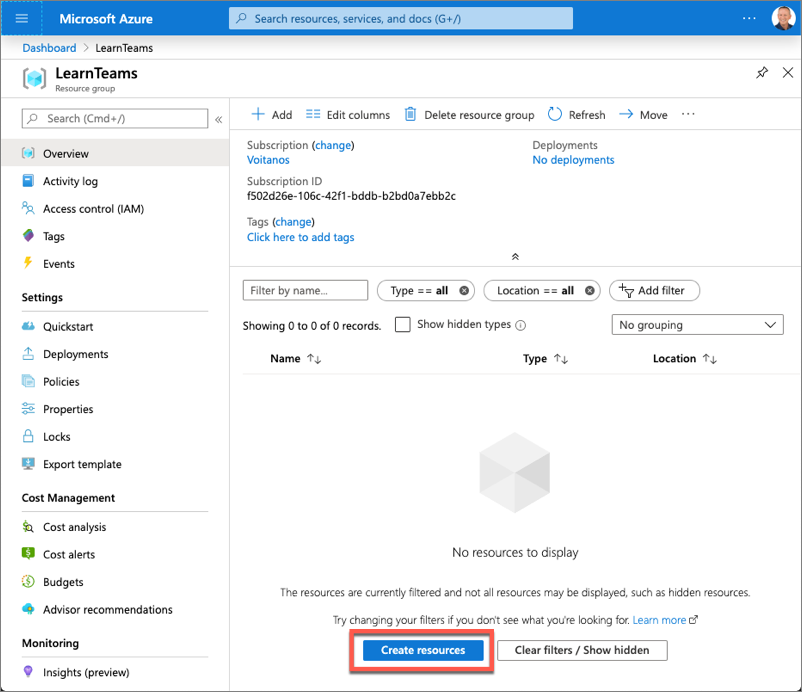
On the **Resource Group** page, select the **Create** button to create a new resource group.

Select a valid subscription, enter a name for the resource group, and select the wanted region. *None of these choices will impact the bot registration and are up to you.*

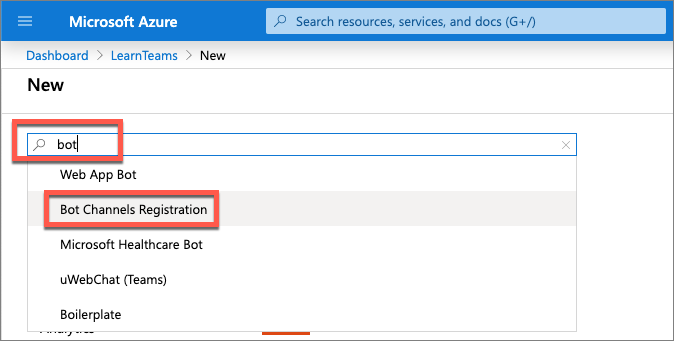


Complete the wizard to create the resource group. Once Azure has completed the resource group creation process, navigate to the resource group.

From the resource group, select the **Add** or **Create resources** button.



Enter **bot** in the **Search the marketplace** input box, and select **Bot Channels Registration** from the list of resources returned. Then select **Create** on the next page to start the process of registering a new bot resource:



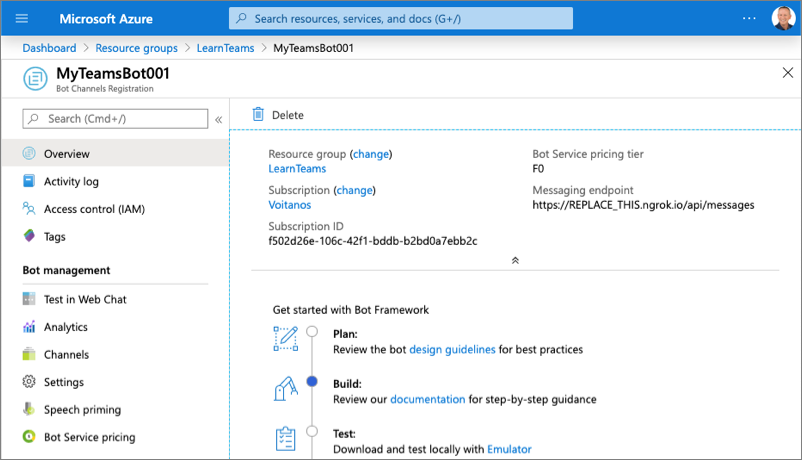
In the **Bot Channels Registration** blade, enter the following values and then select **Create**:

* **Bot handle**: *Enter a globally unique name for the bot*
* **Subscription**: *Select the subscription you selected previously when creating the resource group*
* **Resource group**: *Select the resource group you created previously*
* **Location**: *Select your preferred Azure region*
* **Pricing tier**: *Select a preferred pricing tier; the F0 tier is free*
* **Messaging endpoint**: [https://REPLACE\_THIS.ngrok.io/api/messages](https://replace_this.ngrok.io/api/messages)

The bot registration needs to know the endpoint of the web service where the bot is implemented. This will change each time you start the ngrok utility used in previous exercises.

* **Application Insights**: Off
* **Microsoft App ID and password**: Auto create App ID and password

Azure will start to provision the new resource. This will take a moment or two. Once it is finished, navigate to the bot resource in the resource group.

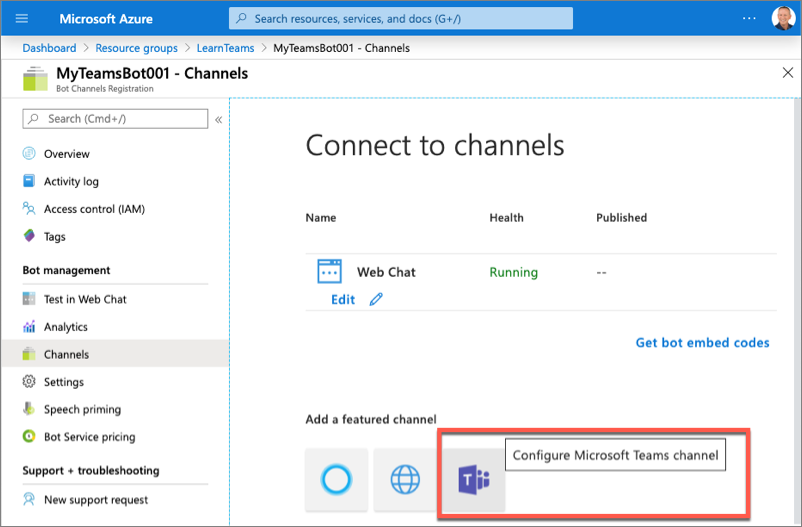


## Enable the Microsoft Teams channel for the bot

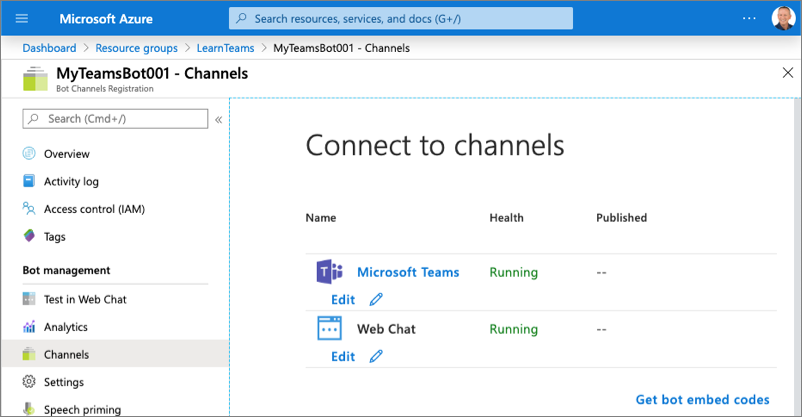
In order for the bot to interact with Microsoft Teams, you must enable the Teams channel.

From the bot resource in Azure, select **Channels** in the left-hand navigation.

On the **Connect to channels** pane, select the Microsoft Teams channel, then select **Save** to confirm the action.



Once this process is complete, you should see both the **Web Chat** and **Microsoft Teams** listed in your enabled channels:



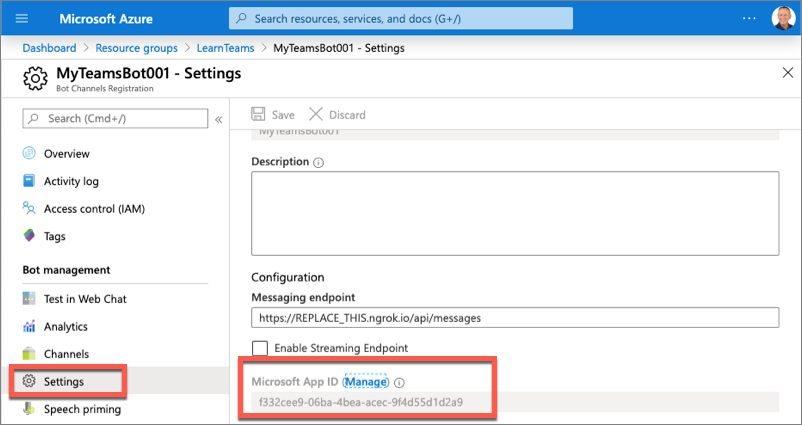
## Retrieve the bot app ID and password

When Azure created the bot, it also registered a new Azure AD app for the bot. You need to generate this new bot app a secret and copy the app's credentials.

Select **Settings** from the left-hand navigation. Scroll down to the **Microsoft App ID** section.

Copy the ID of the bot as you'll need it later.

Select **Manage** to navigate to the Azure AD app blade:

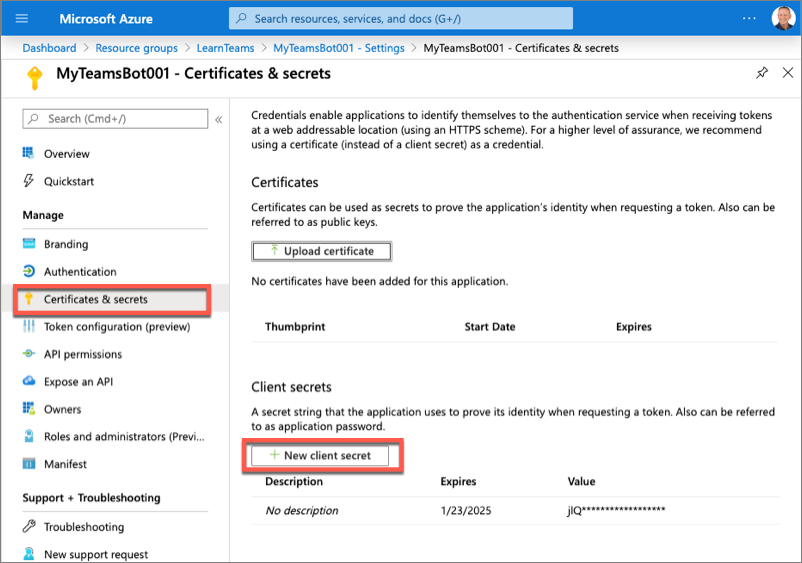


## Create a client secret for the app

In order for the daemon app to run without user involvement, it will sign in to Azure AD with an application ID and either a certificate or secret. In this exercise, you'll use a secret.

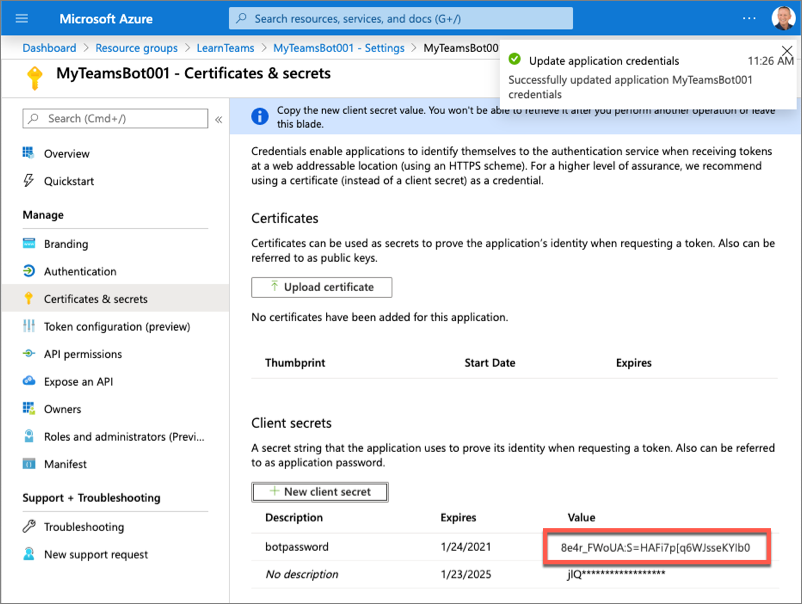
Select **Certificates & secrets** from the left-hand navigation panel.

Select the **New client secret** button:



When prompted, give the secret a description and select one of the expiration duration options provided and select **Add**. *What you enter and select doesn't matter for the exercise.*

The **Certificate & Secrets** page will display the new secret. It's important you copy this value as it's only shown this one time; if you leave the page and come back, it will only show as a masked value.



Copy the value of the secret as you'll need it later.

## Create Microsoft Teams app

In this section, you'll create a new Node.js project.

**Note**

At the time of publication of this module, there are plans to update the Yeoman generator for Microsoft Teams to scaffold new bot projects using the the Bot Framework v4 SDK. However, at the time of publication of this module, the default project uses an older version of the Bot Framework SDK.

Therefore, the steps in this section may change over time because the Yeoman generator may simplify the creation of bots. This exercise will guide you through creating a bot and configuring the project manually to use the Bot Framework v4 SDK because this is the current recommended approach.

Open your command prompt, navigate to a directory where you want to save your work, create a new folder **learn-msteams-bots**, and change directory into that folder.

Run the Yeoman Generator for Microsoft Teams by running the following command:

yo teams

Yeoman will launch and ask you a series of questions. Answer the questions with the following values:

* **What is your solution name?**: ConversationalBot
* **Where do you want to place the files?**: Use the current folder
* **Title of your Microsoft Teams App project?**: Conversational Bot
* **Your (company) name? (max 32 characters)**: Contoso
* **Which manifest version would you like to use?**: 1.5
* **Enter your Microsoft Partner Id, if you have one?**: (Leave blank to skip)
* **What features do you want to add to your project?**: *(uncheck the default option****A Tab****using the space key and press enter)*
* **The URL where you'll host this solution?**: [https://conversationalbot.azurewebsites.net](https://conversationalbot.azurewebsites.net/)
* **Would you like to include Test framework and initial tests?**: No
* **Would you like to use Azure Applications Insights for telemetry?**: No

**Note**

Most of the answers to these questions can be changed after creating the project. For example, the URL where the project will be hosted isn't important at the time of creating or testing the project.

After answering the generator's questions, the generator will create the scaffolding for the project and then execute npm install that downloads all the dependencies required by the project.

**Note**

At the time of publication of this module, the project created by the Yeoman generator for Microsoft Teams includes outdated dependencies to Bot Framework related packages. In the next few steps, you'll update the project's dependencies to use the currently recommended packages.

Remove the older bot SDK-related packages by executing the following command in the command line from the root folder of the project:

npm uninstall botbuilder-dialogs botbuilder-teams botframework-config

Next, upgrade the existing **botbuilder** package and **@microsoft/teams-js** packages to the recommended versions:

npm install botbuilder@4.7.1 @microsoft/teams-js@1.6.0 -SE

## Add a bot to the project

In this section, you'll manually add a bot to the project.

Create a new folder **convoBot** in the **./src/app** folder.

Create a new file **convoBot.ts** in this new folder **./src/app/convoBot/convoBot.ts**.

Add the following code to the **convoBot.ts** file:

import {

TeamsActivityHandler,

TurnContext,

MessageFactory

} from "botbuilder";

import \* as Util from "util";

const TextEncoder = Util.TextEncoder;

import \* as debug from "debug";

const log = debug("msteams");

export class ConvoBot extends TeamsActivityHandler {

constructor() {

super();

this.onMessage(async (context: TurnContext, next: () => Promise<void>) => {

// insert onMessage() handler code here

});

}

}

The first version of this bot will respond to the message **MentionMe** in a 1:1 chat conversation. The response will mention the user who initiated the conversation.

To implement this functionality, add the following method to the ConvoBot class:

private async handleMessageMentionMeOneOnOne(context: TurnContext): Promise<void> {

const mention = {

mentioned: context.activity.from,

text: `<at>${new TextEncoder().encode(context.activity.from.name)}</at>`,

type: "mention"

};

const replyActivity = MessageFactory.text(`Hi ${mention.text} from a 1:1 chat.`);

replyActivity.entities = [mention];

await context.sendActivity(replyActivity);

}

Next, call this method when the bot receives the specific string **MentionMe**. Add the following code to the onMessage() handler, replacing the code comment // insert onMessage() handler code here to implement this:

const botMessageText: string = context.activity.text.trim().toLowerCase();

if (botMessageText === "mentionme") {

await this.handleMessageMentionMeOneOnOne(context);

}

await next();

## Expose the bot as part of the Node.js app's REST API

After creating the bot, the next step is to expose it as part of the app's REST API.

First, add the bot to the **./src/app/TeamsAppsComponents.ts** file by adding the following code to the end of that file:

export \* from "./convoBot/convoBot";

This file is used in the core web server file. This file needs to be updated to expose the bot to the app's API and to configure a bot adapter for the app.

Locate and open the web server file, **./src/app/server.ts**.

Add the following two import statements after the existing import statements in the file:

import { BotFrameworkAdapter } from "botbuilder";

import { ConvoBot } from "./convoBot/convoBot";

**Tip**

Locate the following lines in the **server.ts** file. These lines load all the components and registers them with the web server's REST API routing.

import \* as allComponents from "./TeamsAppsComponents";

...

express.use(MsTeamsApiRouter(allComponents));

The last step is to configure the bot framework and call the bot when requests are received through the /api/messages path. Add the following code to the end of the **./src/app/server.ts** file:

// register and load the bot

const botAdapter = new BotFrameworkAdapter({

appId: process.env.MICROSOFT\_APP\_ID,

appPassword: process.env.MICROSOFT\_APP\_PASSWORD

});

// configure what happens when there is an unhandled error by the bot

botAdapter.onTurnError = async (context, error) => {

console.error(`\n [bot.onTurnError] unhandled error: ${error}`);

await context.sendTraceActivity("OnTurnError Trace", `${error}`, "https://www.botframework.com/schemas/error", "TurnError");

await context.sendActivity("bot error");

};

// run the bot when messages are received on the specified path

const bot = new ConvoBot();

express.post("/api/messages", (request, response) => {

botAdapter.processActivity(request, response, async (context) => {

await bot.run(context);

});

});

In the code above, the first section initializes the Bot Framework adapter with the Azure AD app credentials created when you registered the bot in the Azure portal. These two properties, the Azure AD app's ID and secret, are pulled from an environment variable. This project contains a file, **./.env** that is used to set environment variables when it runs. You need to set these two values for the bot to work:

Locate and open the file **./.env**.

Locate the following section in the file, and set the values of the two properties that you obtained when registering the bot:

# App Id and App Password ofr the Bot Framework bot

MICROSOFT\_APP\_ID=

MICROSOFT\_APP\_PASSWORD=

## Register the bot in the Microsoft Teams app

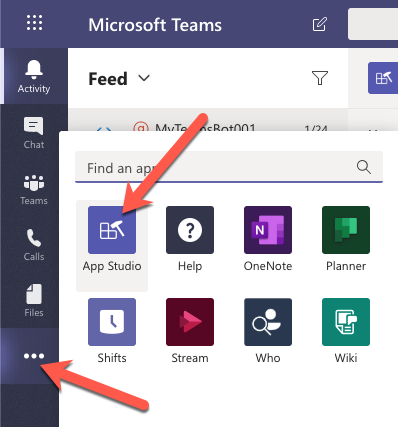
The last step before you can test bot is to add it to the Microsoft Teams app manifest. You can use App Studio to do this.

In the browser, navigate to [**https://teams.microsoft.com**](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

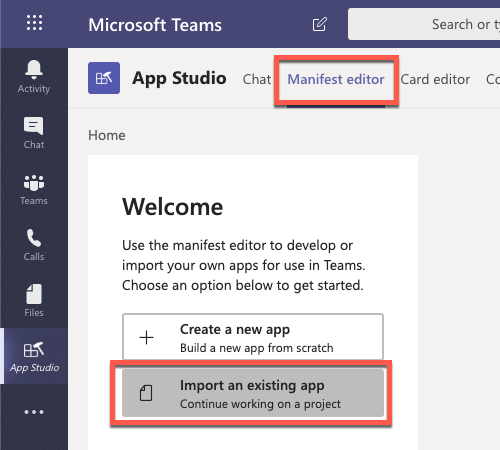
**Note**

Microsoft Teams is available for use as a web client, desktop client and a mobile client. In this module, we will use the web client but any of the clients can be used.

Using the app bar navigation menu, select the **Mode added apps** button. Then select **App Studio**:

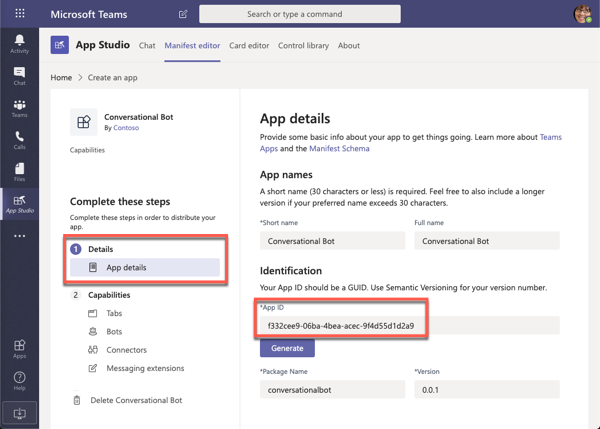


Select the **Manifest editor** tab and then the **Import an existing app** button:



Locate and open the **./src/manifest/manifest.json** from your project. Select the **Conversational Bot** in the **Recently created apps** section of the page.

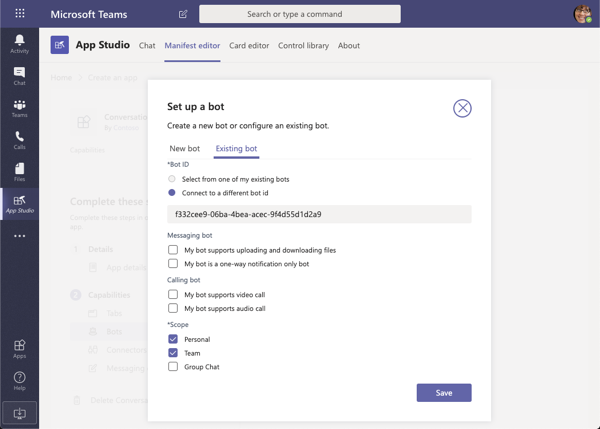
First, update the **App ID** to match the ID of the bot you registered previously in the Azure AD app ID you obtained when registering the bot. You'll find this on the **(1) Details** > **App details** page:



From the **(2) Capabilities** > **Bots** page, select **Set up** to add a bot to the manifest.

Because you previously created a bot using the Microsoft Azure's Bot Framework, select **Existing bot** and set the following values and select **Save**:

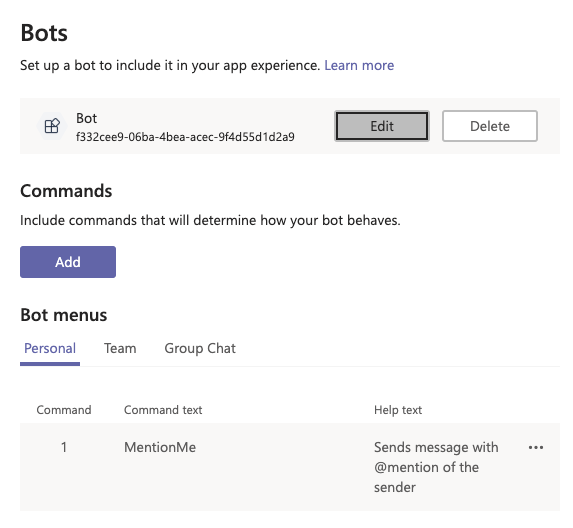
* **Bot ID**
  + Connect to a different bot id: <REPLACE\_WITH\_MICROSOFT\_APP\_ID>
* **Scope**: Personal, Team



Within the **Commands** section, select **Add** to add a new command to the bot.

On the **New command** dialog, enter the following values and select **Save**:

* **Command text**: MentionMe
* **Help text**: Sends message with @mention of the sender
* **Scope**: Personal



With the bot added to the Teams app, you need to update the manifest in your project. Fro the **(3) Finish** > **Test and distribute** section, select the **Download** button from the **Download** section.

This will download the app package as a ZIP. Unpack the zip and copy the **manifest.json** file in it to your project, updating the existing **./src/manifest/manifest.json** file.

In the **./src/manifest/manifest.json** file, verify icons property's values, and update if necessary, file names to match what's in the project

Locate the property id. Change its value to match the GUID of the Azure AD app that was created when creating the bot in the Azure portal.

Locate the property bots. Add a new bot to the collection of bots registered with this Microsoft Teams app by adding the following JSON to the array. This code will add our bot to the personal scope of the user when its installed. It includes a single help message that will show the command it supports, **MentionMe**.

"bots": [

{

"botId": "<REPLACE\_WITH\_MICROSOFT\_APP\_ID>",

"scopes": ["personal"],

"supportsFiles": false,

"isNotificationOnly": false,

"commandLists": [

{

"scopes": ["personal"],

"commands": [

{

"title": "MentionMe",

"description": "Sends message with @mention of the sender"

}

]

}

]

}

],

**Important**

Ensure you replace the botId property's value with the Azure AD app ID you obtained when registering the bot.

At this point, your bot is ready to test!

## Test the conversation bot

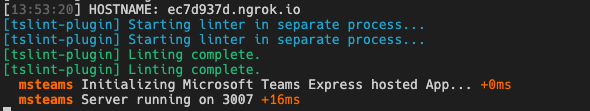
From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

This gulp task will run many other tasks all displayed within the command-line console. The **ngrok-serve** task builds your project and starts a local web server (http://localhost:3007). It then starts ngrok with a random subdomain that creates a secure URL to your local webserver.

**Note**

Microsoft Teams requires all content displayed within a tab be loaded from an HTTPS request. In development, can be done using the tool [**ngrok**](https://www.ngrok.com/) that creates a secure rotatable URL to your local HTTP webserver. Ngrok is included as a dependency within the project so there is nothing to setup or configure.



Note the URL of the Ngrok URL displayed in the console. In the previous screenshot, NGrok has created the temporary URL **ec7d937d.ngrok.io** that will map to our locally running web server. In order for the Bot Framework to route messages from Microsoft Teams to our locally running bot, you need to update the bot's messaging endpoint in the Azure portal.

Open a browser and navigate to the [Azure portal](https://portal.azure.com/) and sign in using a **Work or School Account** that has rights to create resources in your Azure subscription.

Locate the bot by selecting the Azure Resource Group and Bot Channels Registration resource you created at the beginning of this exercise.

Using the left-hand navigation, select **Bot management** > **Settings**.

Locate the property **Configuration** > **Messaging endpoint** and set the domain to the NGrok domain.

Finally, save your changes to the bot configuration using the **Save** button at the top of the page.

**Important**

The free version of Ngrok will create a new URL each time you restart the web server. Make sure you update the **Messaging endpoint** of your URL each time you restart the web server when you are testing the app.

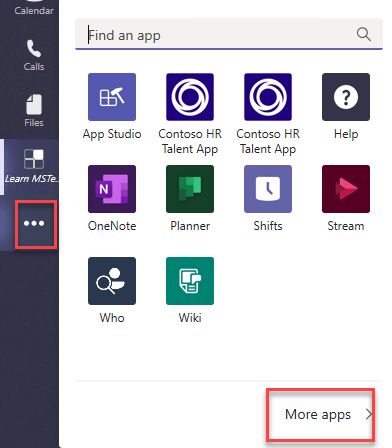
## Install the custom app in Microsoft Teams

Now let's install the app in Microsoft Teams. In the browser, navigate to [**https://teams.microsoft.com**](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

**Note**

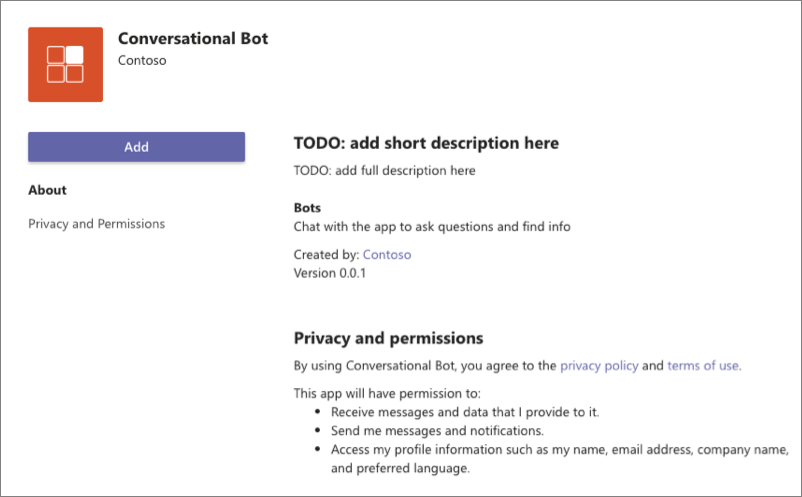
Microsoft Teams is available for use as a web client, desktop client and a mobile client. In this module, we will use the web client but any of the clients can be used.

Using the app bar navigation menu, select the **More added apps** button. Then select **More apps** followed by **Upload a custom app** > **Upload for me or my teams**.

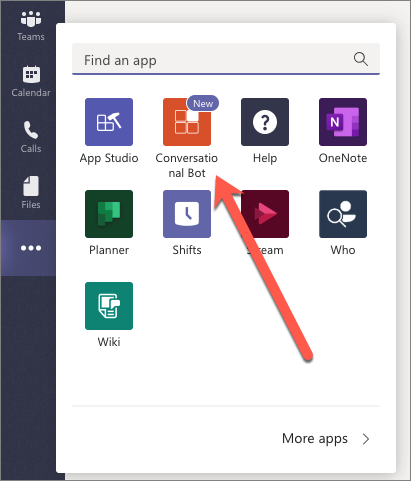


In the file dialog that appears, select the Microsoft Teams package in your project. This app package is a ZIP file that can be found in the project's **./package** folder.

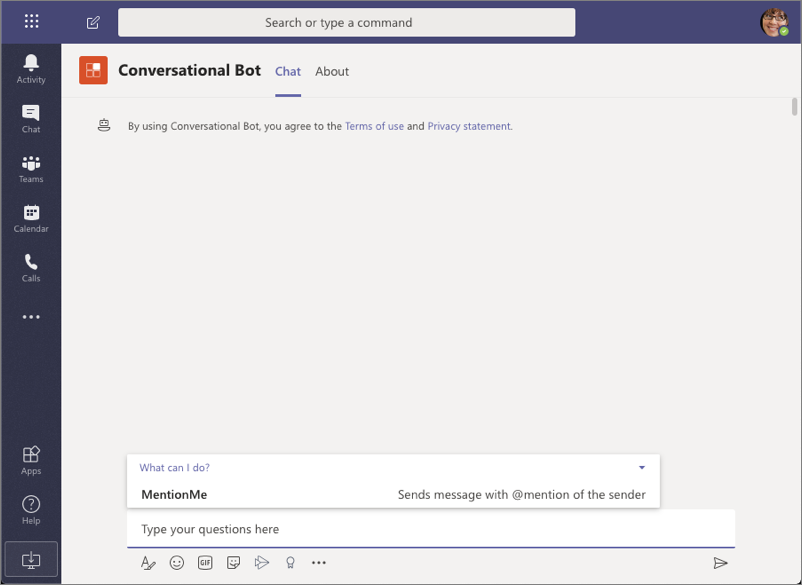
Once the package is uploaded, Microsoft Teams will display a summary of the app. Here you can see some "todo" items to address. *None of these "todo" items are important to this exercise, so you'll leave them as is.*



Select the **Add** button to install the app, adding a new personal tab to your **More added apps** dialog:



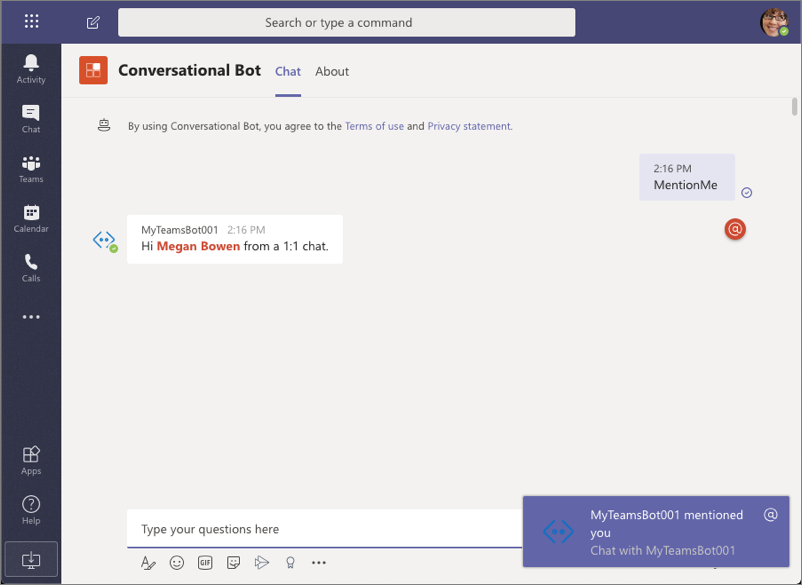
Select the app to navigate to chat with the bot:



Notice the commands that the bot supports are shown in the compose box when the app loads. Let's test the bot!

Select the **MentionMe** command, or manually type **mentionme** in the compose box, then press enter.

After a few seconds, you should see the bot respond mentioning the user you are signed in with:



At this point, we have a working bot that is responding when it's mentioned.

## Summary

In this exercise, you’ll learn how to create and add a new bot to a Microsoft Teams app and interact with it from the Microsoft Teams client.

# Exercise 2- Bots in Microsoft Teams channels and group chats

Conversation bots can do many things within the Microsoft Teams client. They can proactively send a message to a channel or group chat, listen for and act on Microsoft Teams specific events and even update their own messages.

In this exercise, you’ll modify the existing Microsoft Teams app to update your bot to respond to message reactions, and update or delete messages capabilities.

**Important**

This exercise assumes you have created the Microsoft Teams app project with the Yeoman generator that contains a personal tab from the previous exercise in this module. You'll update the project to add a new task module that uses an Adaptive Card.

## Add channel support to a conversation bot

In a previous exercise, you created a bot that could be used in the personal scope that enabled one:one chat. When a bot is used in a channel, you must @mention it to receive messages.

In this section, you'll modify the existing Microsoft Teams app to support being added to a team and respond to messages from the **Conversations** tab.

## Update the app's configuration

First, update app's manifest to add channel support. Locate and open the **./src/manifest/manifest.json** file.

You must increment the version of the app to upgrade an existing installed version. Locate the property version and increment the version to something greater than the default value 0.0.1.

Next, locate the bots.scopes property. Add an addition value teams to the array so it looks like the following code:

"bots": [

{

...

"scopes": [

"personal",

"team"

],

...

}

]

## Update the bot code

The next step is to update the bot's code.

In the previous exercise, our code was looking for the specific message MentionMe in order to respond. This works in a 1:1 personal chat because the bot isn't mentioned in the conversation.

However, in a channel conversation, a user must @mention the bot to trigger it. This results in a message containing a reference to the bot, not just the message submitted.

While there are multiple ways to address this, let's handle it in a simple way: look for MentionMe at the end of the message.

Locate and open the bot in the file **./src/app/convoBot/convoBot.ts**. Locate the existing onMessage() handler in the class constructor. Add the following else if statement to find these new messages sent from a channel conversation:

else if (botMessageText.endsWith("</at> mentionme")) {

await this.handleMessageMentionMeChannelConversation(context); }

The complete onMessage() handler should now look like the following:

this.onMessage(async (context: TurnContext, next: () => Promise<void>) => {

const botMessageText: string = context.activity.text.trim().toLowerCase();

if (botMessageText === "mentionme") {

await this.handleMessageMentionMeOneOnOne(context);

} else if (botMessageText.endsWith("</at> mentionme")) {

await this.handleMessageMentionMeChannelConversation(context);

}

await next();

});

Finally, add the following method to the ConvoBot class to implement the handler for our new scenario:

private async handleMessageMentionMeChannelConversation(context: TurnContext): Promise<void> {

const mention = {

mentioned: context.activity.from,

text: `<at>${new TextEncoder().encode(context.activity.from.name)}</at>`,

type: "mention"

};

const replyActivity = MessageFactory.text(`Hi ${mention.text}!`);

replyActivity.entities = [mention];

const followupActivity = MessageFactory.text(`\*We are in a channel conversation group chat in the !\*`);

await context.sendActivities([replyActivity, followupActivity]);

}

Save your changes, update, and test the installed app.

## Test the conversation bot in a channel

From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

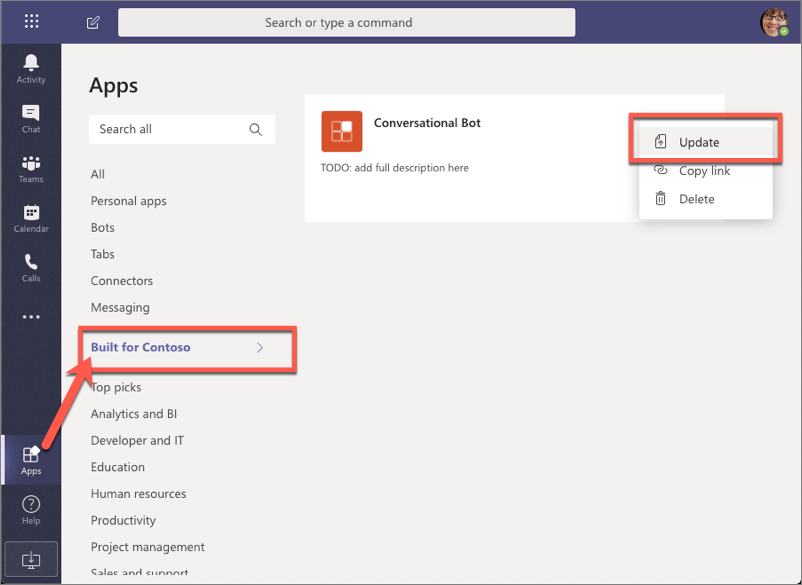
**Important**

Recall from a previous exercise, Ngrok will create a new subdomain. You need to update your bot registration's **Messaging endpoint** in the Azure portal (*shown in a previous exercise*) with this new domain before testing it.

First, update the existing installed version of the bot.

In the browser, navigate to [**https://teams.microsoft.com**](https://teams.microsoft.com/) and sign in with the credentials of a Work and School account.

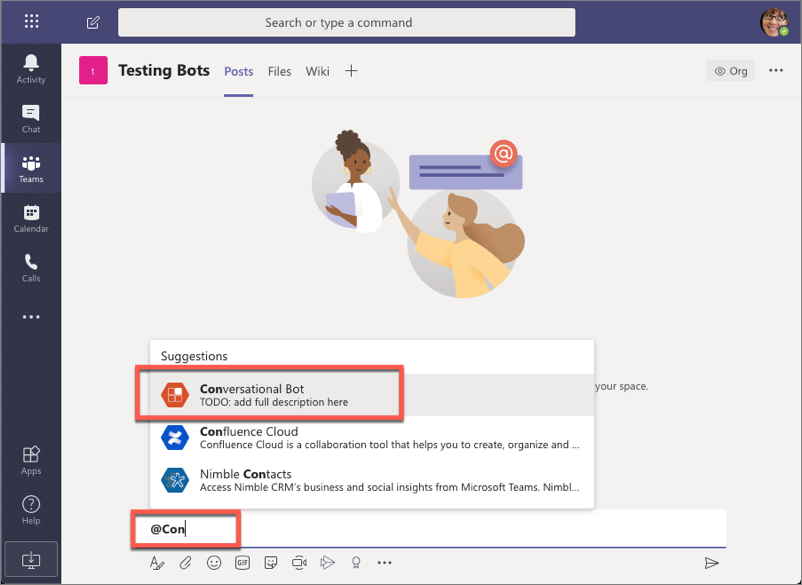
Using the app bar navigation menu, select the **More added apps** button. Then select **More apps**, select the menu in the top-right corner of the **Conversation bot** and select **Update**.



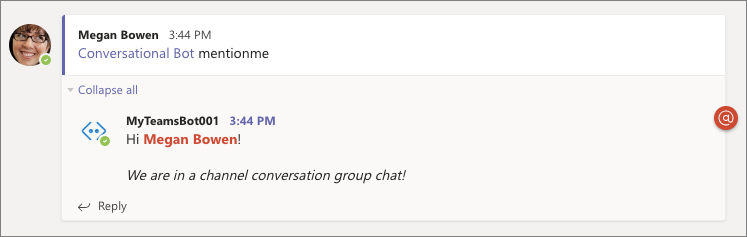
When prompted, select the updated package of the Microsoft Teams app. Microsoft Teams will update the app to the new version.

Navigate to an existing channel in a team.

From the channel's **Conversations** tab, @ the bot. The first time you @mention the bot, you'll be prompted to install it into the team.



After installing the bot, when you @mention it and include the message mentionme, the bot will reply to your message:



## Reply to messages with Adaptive cards

In this section, you'll update the bot to respond to unknown messages using an Adaptive card. The card's single action will trigger the bot to update the existing message with a new Adaptive card. The updated message will include an additional action that will trigger the bot to delete the message.

Locate and open the bot in the file **./src/app/convoBot/convoBot.ts**.

Add the CardFactory and ActionTypes objects to the existing import {...} from "botbuilder"; statement to import two more objects you'll need:

import {

TeamsActivityHandler,

TurnContext,

MessageFactory,

MemoryStorage,

ActionTypes, CardFactory

} from "botbuilder";

Locate the existing onMessage() handler in the class constructor. Add the following else statement to the existing if statement to respond with an adaptive card if the bot receives an unknown command:

else {

const value = { cardAction: "update", count: 0 };

const card = CardFactory.adaptiveCard({

"$schema": "http://adaptivecards.io/schemas/adaptive-card.json",

"type": "AdaptiveCard",

"version": "1.0",

"body": [

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Adaptive card response",

"weight": "bolder",

"size": "large"

}

]

},

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Demonstrates how to respond with a card, update the card & ultimately delete the response.",

"wrap": true

}

]

}

],

"actions": [

{

"type": "Action.Submit",

"title": "Update card",

"data": value

}

]

});

await context.sendActivity({ attachments: [card] });

}

The onMessage() handler should now look like the following:

this.onMessage(async (context: TurnContext, next: () => Promise<void>) => {

const botMessageText: string = context.activity.text.trim().toLowerCase();

if (botMessageText === "mentionme") {

await this.handleMessageMentionMeOneOnOne(context);

} else if (botMessageText.endsWith("</at> mentionme")) {

await this.handleMessageMentionMeChannelConversation(context);

} else {

const value = { cardAction: "update", count: 0 };

const card = CardFactory.adaptiveCard({..});

await context.sendActivity({ attachments: [card] });

}

await next();

});

Notice the else statement will send a card to the conversation that contains a data object in the single actions. This object has a count property & cardAction property. When a user triggers the action, this object will be sent to the bot.

Add the following methods to implement the updateCardActivity() & the deleteCardActivity() handlers:

private async updateCardActivity(context): Promise<void> {

const value = {

cardAction: "update",

count: context.activity.value.count + 1

};

const card = CardFactory.adaptiveCard({

"$schema": "http://adaptivecards.io/schemas/adaptive-card.json",

"type": "AdaptiveCard",

"version": "1.0",

"body": [

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Adaptive card response",

"weight": "bolder",

"size": "large"

}

]

},

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": `Updated count: ${ value.count }`,

"wrap": true

}

]

}

],

"actions": [

{

"type": "Action.Submit",

"title": "Update card",

"data": value

},

{

"type": "Action.Submit",

"title": "Delete card",

"data": { cardAction: "delete"}

}

]

});

await context.updateActivity({ attachments: [card], id: context.activity.replyToId, type: 'message' });

}

private async deleteCardActivity(context): Promise<void> {

await context.deleteActivity(context.activity.replyToId);

}

In the code you've added, notice the updateCardActivity() retrieves and increments the count property it received. It then creates a new card with the same data, but with an additional action to delete the card. Finally, the method uses the updateActivity() method to update an existing message.

The deleteCardActivity() deletes the card using the deleteActivity() method.

The last step is to handle messages that are sent from the adaptive card correctly.

Within the onMessage() method, add the following code at the very start of the method, before the existing code:

// if a value property exists = adaptive card submit action

if (context.activity.value) {

switch (context.activity.value.cardAction) {

case "update":

await this.updateCardActivity(context);

break;

case "delete":

await this.deleteCardActivity(context);

break;

}

} else {

Close the else statement before the last line await next();. The final onMessage() method should look like the following code:

this.onMessage(async (context: TurnContext, next: () => Promise<void>) => {

// if a value property exists = adaptive card submit action

if (context.activity.value) {

switch (context.activity.value.cardAction) {

case "update":

await this.updateCardActivity(context);

break;

case "delete":

await this.deleteCardActivity(context);

break;

}

} else {

const botMessageText: string = context.activity.text.trim().toLowerCase();

if (botMessageText === "mentionme") {

await this.handleMessageMentionMeOneOnOne(context);

} else if (botMessageText.endsWith("</at> mentionme")) {

await this.handleMessageMentionMeChannelConversation(context);

} else {

const value = { cardAction: "update", count: 0 };

const card = CardFactory.adaptiveCard({

/\* card omitted for readability \*/

});

await context.sendActivity({ attachments: [card] });

}

}

await next();

});

## Test the bot updating existing messages

From the command line, navigate to the root folder for the project and execute the following command:

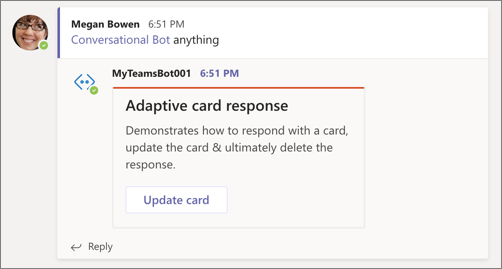
gulp ngrok-serve

**Important**

Recall from a previous exercise, Ngrok will create a new subdomain. You need to update your bot registration's **Messaging endpoint** in the Azure portal (*shown in a previous exercise*) with this new domain before testing it.

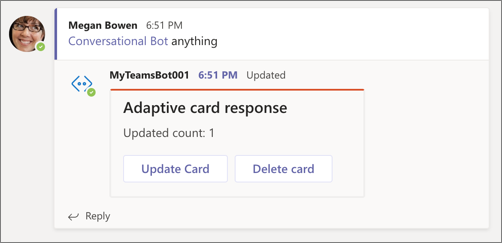
In the Microsoft Teams client, go to the channel you installed the bot in the previous section. From the **Conversations** tab, @mention the bot with a random string to trigger the else condition.

The bot will reply to the message with a card:



Notice the value in the card is set to **0**.

Select the button **Update card**. After a few seconds, the card should be updated with a new card containing an incremented counter value and a new button:



Select the **Update card** button a few more times to see the counter get updated.

Finally, select the **Delete card** button. After a few seconds, the card will be removed by the bot.

## Reply to message reactions

In this section, you'll update the bot to respond when someone likes a message from the bot.

Locate and open the bot in the file **./src/app/convoBot/convoBot.ts**.

Add the following handler to the existing class constructor method:

this.onReactionsAdded(async (context: TurnContext, next: () => Promise<void>) => {

if (context.activity.reactionsAdded) {

context.activity.reactionsAdded.forEach(async (reaction) => {

if (reaction.type === "like") {

await context.sendActivity("Thank you!");

}

});

}

await next();

});

This code will execute when a user adds a reaction to a message from the bot. If the reaction is a *like*, the bot will reply with a *"Thank you!"* message

## Test the bot reacting to message reactions

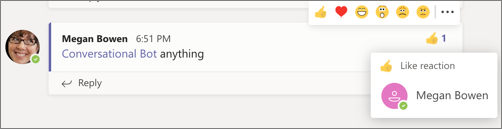
From the command line, navigate to the root folder for the project and execute the following command:

gulp ngrok-serve

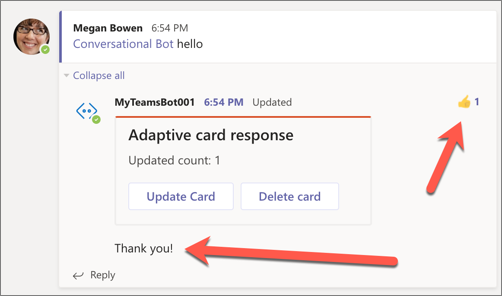
**Important**

Recall from a previous exercise, Ngrok will create a new subdomain. You need to update your bot registration's **Messaging endpoint** in the Azure portal (*shown in a previous exercise*) with this new domain before testing it.

In the Microsoft Teams client, go to the channel you installed the bot in the previous section. From the **Conversations** tab, find a message from the bot and apply a *like* reaction to it:



After a few seconds, the bot will reply with a message, thanking them for liking the reaction:



## Summary

In this exercise, you modified the existing Microsoft Teams app to update your bot to respond to message reactions, and update or delete messages capabilities.

# Exercise 3- Proactive messages from bots

In this exercise, you’ll update the existing Teams app to send a proactive message from your bot.

**Important**

This exercise assumes you have created the Microsoft Teams app project with the Yeoman generator that contains a personal tab from the previous exercise in this module. You'll update the project to add a new task module that uses an Adaptive Card.

## Initiate a proactive message from the bot

Locate and open the bot in the file **./src/app/convoBot/convoBot.ts**.

Add the following objects to the existing import {...} from "botbuilder"; statement you'll need:

import {

ChannelInfo, TeamsChannelData, ConversationParameters, teamsGetChannelId,

Activity, BotFrameworkAdapter, ConversationReference, ConversationResourceResponse

} from "botbuilder";

Locate the card in the else statement in the onMessage() handler you added in the previous section. Add a second action button to the card that will trigger the creation of a new message:

{

"type": "Action.Submit",

"title": "Create new thread in this channel",

"data": { cardAction: "newconversation" }

}

The card should now look like the following:

const card = CardFactory.adaptiveCard({

"$schema": "http://adaptivecards.io/schemas/adaptive-card.json",

"type": "AdaptiveCard",

"version": "1.0",

"body": [

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Adaptive card response",

"weight": "bolder",

"size": "large"

}

]

},

{

"type": "Container",

"items": [

{

"type": "TextBlock",

"text": "Demonstrates how to respond with a card, update the card & ultimately delete the response.",

"wrap": true

}

]

}

],

"actions": [

{

"type": "Action.Submit",

"title": "Update card",

"data": value

},

{

"type": "Action.Submit",

"title": "Create new thread in this channel",

"data": { cardAction: "newconversation" }

}

]

});

Next, add another case statement to the switch statement the onMessage() handler to detect this new action:

case "newconversation":

const channelId = teamsGetChannelId(context.activity);

const message = MessageFactory.text("This will be the first message in a new thread");

const newConversation = await this.createConversationInChannel(context, channelId, message);

break;

The last step is to add the createConversationInChannel() method that will create the new conversation. Add the following method to the ConvoBot class:

private async createConversationInChannel(context: TurnContext, teamsChannelId: string, message: Partial<Activity>): Promise<[ConversationReference, string]> {

// create parameters for the new conversation

const conversationParameters = <ConversationParameters>{

isGroup: true,

channelData: <TeamsChannelData>{

channel: <ChannelInfo>{

id: teamsChannelId

}

},

activity: message

};

// get a reference to the bot adapter & create a connection to the Teams API

const adapter = <BotFrameworkAdapter>context.adapter;

const connectorClient = adapter.createConnectorClient(context.activity.serviceUrl);

// create a new conversation and get a reference to it

const conversationResourceResponse: ConversationResourceResponse = await connectorClient.conversations.createConversation(conversationParameters);

const conversationReference = <ConversationReference>TurnContext.getConversationReference(context.activity);

conversationReference.conversation.id = conversationResourceResponse.id;

return [conversationReference, conversationResourceResponse.activityId];

}

## Test the bot sending new messages

From the command line, navigate to the root folder for the project and execute the following command:

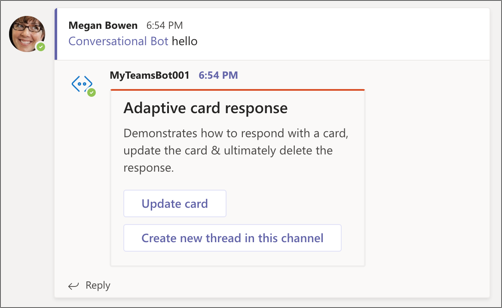
gulp ngrok-serve

**Important**

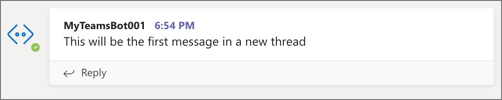
Recall from a previous exercise, Ngrok will create a new subdomain. You need to update your bot registration's **Messaging endpoint** in the Azure portal (*shown in a previous exercise*) with this new domain before testing it.

In the Microsoft Teams client, go to the channel you installed the bot in the previous section. From the **Conversations** tab, @mention the bot with a random string to trigger the else condition.

The bot will reply to the message with the updated card that contains two buttons:



Select the second button, **Create new thread in this channel**. Within a few seconds, you should see a new conversation appear in the channel:



## Summary

In this exercise, you’ll modify the existing Microsoft Teams app update the bot to send a proactive message from your bot.