**Microsoft Bot Framework:**

Creating a chatbot:

1. Log in to the [Azure portal](http://portal.azure.com/).
2. Click **Create new resource** link found on the upper left-hand corner of the Azure portal, then select **AI + Machine Learning** > **Web App bot**.
3. A new blade will open with information about the **Web App Bot**.
4. Write the bot name, subscription, and other details, set the **Bot template** to your required need.
5. Click on Create
6. After **Deployment is successful**, you can access the bot in the Resources section. (**Go to resource** button).
7. Don’t test the bot online as it will reduce your free credits, for testing download the code

Downloading the chatbot to local machine:

1. In the **Bot Management** section, click **Build**.
2. Click on **Download Bot source code** link in the right-pane.
3. Follow the prompts to download the code, and then unzip the folder.
4. When downloading your bot, you will be given the option to include the settings (containing the keys and secrets) for your bot in your download, which may be necessary for your bot to work. If you choose **Yes**, the appsettings.json or .env file will have the keys.

Downloading Emulator: <https://github.com/Microsoft/BotFramework->

Emulator/blob/master/README.md

For deployment to Azure:

1. Make sure that your repository root has the correct files in your project. (server.js, app.js, or package.json with a start script)
2. To enable continuous deployment with GitHub, navigate to the **App Service** page for your bot in the Azure portal.
3. Click **Deployment Center** > **GitHub** > **Authorize**.
4. In the browser window that opens up, click **Authorize AzureAppService**.
5. After authorizing the **AzureAppService**, go back to **Deployment Center** in the Azure portal.
6. Click **Continue**.
7. Select **App Service build service**.
8. Click **Continue**.
9. Select **Organization**, **Repository**, and **Branch**.
10. Click **Continue**, and then **Finish** to complete the setup.

Creating a QNA Maker:

1. Clone or copy the samples repo to your computer.
2. Sign into to the [QnA Maker portal](https://qnamaker.ai/) with your Azure credentials and select **Create a knowledge base**.
3. If you have not already created a QnA Maker service, select **Create a QnA service**.
4. Select your Azure tenant, Azure subscription name, and Azure resource name associated with the QnA Maker service from the lists in **Step 2** in the QnA Maker portal. Select the Azure QnA Maker service that will host the Knowledge Base.
5. Enter the name of your knowledge base and the data sources for the new knowledge base.
6. Give your service a **name.** Duplicate names and special characters are supported.
7. Add URLs for data you want extracted. See more information on the types of sources supported [here](https://docs.microsoft.com/en-us/azure/cognitive-services/qnamaker/concepts/data-sources-supported).
8. Upload files for data you want extracted. See the [pricing information](https://aka.ms/qnamaker-pricing) to see how many documents you can add.
9. If you want to manually add QnAs, you can skip **Step 4**
10. Add **Chit-chat** to your KB. Choose to add chit-chat support for your bot, by choosing from one of the personalities.
11. Select **Create your KB**.
12. It takes a few minutes for data to be extracted.
13. When your Knowledge Base is successfully created, you are redirected to the **Knowledge base** page.
14. **Save and train** your knowledge base.
15. **Publish** your knowledge base.
16. Once your QnA Maker app is published, select the SETTINGS Tab, and scroll down to 'Deployment details'. Record the following values from the Postman Sample HTTP request-

POST /knowledgebases/<knowledge-base-id>/generateAnswer

Host: <your-hostname> // NOTE - this is a URL ending in /qnamaker.

Authorization: EndpointKey <qna-maker-resource-key>

1. The full URL string for your hostname will look like "https://< >.azure.net/qnamaker".
2. These values will be used within your appsettings.json or .env file in the next step.
3. The knowledge base is now ready for your bot to use.
4. Add the following values to your .env file:

MicrosoftAppId=""

MicrosoftAppPassword=""

ScmType=None

QnAKnowledgebaseId="<knowledge-base-id>"

QnAAuthKey="<qna-maker-resource-key>"

QnAEndpointHostName="<your-hostname>" // This is a URL ending in /qnamaker

1. Now save your edits.
2. Open a terminal or command prompt to the root directory for your project.
3. Add the **botbuilder-ai** npm package to your project.
4. In **index.js**, following the // Create Adapter section, add the following code to read your .env file configuration information needed to generate the QnA Maker services.

// Map knowledgebase endpoint values from .env file into the required format for `QnAMaker`.

const configuration = {

knowledgeBaseId: process.env.QnAKnowledgebaseId,

endpointKey: process.env.QnAAuthKey,

host: process.env.QnAEndpointHostName

};

1. Update the bot construction to pass in the QnA services configuration information.

// Create the main dialog.

const myBot = new MyBot(configuration, {});

1. In your **bot.js** file, add this require for QnAMaker

const { QnAMaker } = require('botbuilder-ai');

1. Modify the constructor to now receive passed configuration parameters required to create a QnAMaker connector and throw an error if these parameters are not provided.

class MyBot extends ActivityHandler {

constructor(configuration, qnaOptions) {

super();

if (!configuration) throw new Error('[QnaMakerBot]: Missing parameter. configuration is required');

// now create a qnaMaker connector.

this.qnaMaker = new QnAMaker(configuration, qnaOptions);

1. Finally, add the following code to your onMessage( ) call that passes each user input to your QnA Maker knowledgebase and returns the QnA Maker response back to the user to query your knowledge bases for an answer.

// send user input to QnA Maker.

const qnaResults = await this.qnaMaker.getAnswers(turnContext);

// If an answer was received from QnA Maker, send the answer back to the user.

if (qnaResults[0]) {

await turnContext.sendActivity(`QnAMaker returned response: ' ${ qnaResults[0].answer}`);

}

else {

// If no answers were returned from QnA Maker, reply with help.

wait turnContext.sendActivity('No QnA Maker response was returned.'

+ 'This example uses a QnA Maker Knowledge Base that focuses on smart light bulbs. '

+ `Ask the bot questions like "Why won't it turn on?" or "I need help."`);

}