How to Build Your Bot

Russ Williams & Charlene Wahl

Table of Contents

Overview

This workshop **IS NOT** about how to build **A BOT**, it's about how to build **YOUR BOT**! Think of it as a "one-stop shop" for building your first enterprise bot. It's part recipe, part master plan, which you can use to accelerate your bot journey from conception to production deployment. You'll use the **Plan** and **Build** links from the master plan to complete this workshop and then use the other links in the master plan as you continue to extend and refine your bot all the way through to production deployment.

How to Build Your Bot

Master Plan for Building a Virtual Assistant using the Microsoft Bot Framework

Plan:
Workshop Prechecks and Prerequisites
Identify Scenarios
Author Dialogs, design cards, visualize
(Post Workshop) Define Actions and Review Design Guidelines

Build:
Create, Deploy, Run your Virtual Assistant
Customize your Virtual Assistant
Update QnA Maker Knowledgebase
Add a Custom Skill to Your Virtual Assistant
Implement Your Skill's Core Scenario

Test:
Testing and debugging guidelines, Trouble Shooting
Bot emulator and Wiki
Conversation Transcript
Unit Testing and Load Testing Bots (coming soon)

Publish:
Publish using Visual Studio
DevOps

Manage:
Manage using Azure Portal and Configure Bot Setting
Enable channels
Integrate with other Cognitive Services

Learn:
Read all our doc on Bot Framework Solutions, Azure Bot Service, LUIS, and QnA Maker
Leverage Learning Journeys to drill in on bot basics
Have questions or need help use one or more of the options discussed here
Report bugs or issues for core SDK here, Bot Builder tools here, Bot Framework Solutions here, and other options here.

Plan

1. Workshop Prechecks and Prerequisites
2. Identify Scenarios
3. Author Dialogs, Design Cards, Visualize
4. (Post Workshop) Define Actions and Supporting Activities
5. (Post Workshop) Review Design Guidelines

In the planning stage you'll take steps to prepare for an effective workshop that's been designed to accelerate the development of an enterprise grade bot. These steps do not require participants to have any prior experience with the Microsoft Bot Framework. Spending quality time preparing for the workshop is the best way to insure a successful outcome. Steps 1 through 3 should be discussed in a pre-call with architect who will be leading the workshop. Steps 4 and 5 can be reviewed for context before the workshop and followed up on post workshop.

Workshop Prechecks and Prerequisites

A critical step in preparing for the workshop is installing the bot framework SDK and tools and making sure you have enough permissions in your Azure Subscription to create all the resources and services required by the Virtual Assistant Template. Normally, these installation steps are done in the workshop itself, but its important to confirm that the attendees have enough permissions to install software on their development PC before the workshop starts (i.e. admin rights to the PC).

There are two sets of prerequisites that must be installed: Virtual Assistant and Skills prerequisites. The instructions for both must be followed exactly or you will experience strange errors later when you try to build your bot and it won't be obvious that the errors you are seeing are a result of improper installation. The prerequisites that need to be installed for the Virtual Assistant can be found here and the second set of installation steps for Skills can be found here. For Skills, you only need to do step 1 where you install the Skills template since the rest of those installation steps are redundant. If you find yourself unsure of what to do in any of the installation steps, you can look at this short screen recording that shows the whole process end-to-end.

Although not exactly necessary, it's never a bad idea to update Visual Studio which you can do by launching it and choosing **Help | Check for Updates**. This can safely be done before workshop begins.

Import: In addition to the above installation prerequisites, developers will also need to have an Azure Subscription and enough permissions in their Azure Subscription to do the following:

- Add apps to App Registration Portal (https://apps.dev.microsoft.com)
  Note: If they have never created an app registration before then they can check this by creating an app registration and then turn right around and delete it
- Azure Portal rights to create:
  - Azure Resource Groups
  - Azure App Service
  - Azure Web App Bot
  - Azure Cosmos DB
  - Azure App Service Plan
  - Azure Cognitive Services (LUIS & QnA Maker)
  - Azure Search
  - Azure Application Insights

- Azure Storage Account

All these services will be created in this workshop and your developers should check to make sure they have permission to create them. One way to confirm they have enough permissions is to create each of those resources/services in the Azure Portal and then turn right around and delete them.

Identify Scenarios

Knowing which scenarios the bot will be expected to handle is as important to the bot as knowing a person's job responsibilities are for a human. Thinking about the bot as a real person can be very helpful in discovering scenarios. When you hire a new employee, you tell them what you expect of them and what job duties they are expected to perform. Think of the bot as a human performing a specific role and job duties. What kinds of specific job duties do you want your bot to perform? The answer to that question will be your scenarios.

If your bot were performing the role of a bank teller, some of the specific job duties might be:

- Open bank account
- Close bank account
- Make deposit
- Make withdrawal
- Provide account balance

Author Dialog, Design Cards, and Visualize

Continuing with this idea of thinking about the bot as a person, and having identified one or two key scenarios, it would be only natural to now wonder how each scenario should be handled. What would a conversation for each scenario look like? What kind of back and forth would be necessary for someone to gather enough information to carry out the goal of the scenario? Essentially, you'll need to create a script, and in Conversational AI parlance, we call that script a "dialog". Dialog in this context means the conversation between the user and the bot, not a rectangle form on a computer screen.

The easiest way to model and design a dialog for a scenario is to simply jot it down and label each sentence with the name of who spoke those words. So, it might be something like:

- Bot: Hello, I'm Angie, your Contoso Bank teller
- Bot: I can help with various things like opening or closing an account, making deposits or withdraws, or getting an account balance.
- Bot: What can I help you with today?
- User: I'd like to open an account
- Bot: Super, let's do that! I'll need to ask a few questions first, but it won't take long.
- Bot: What is your name?
- User: Russ Williams
- Bot: Russ, what is your email?
- etc., etc., etc.
- Bot: Alright, that's all we need to setup your account Russ. You'll receive a confirmation email shortly to keep for your records
- Bot: What else can I help you with?

It is possible to create a rich mockup of a dialog design using the Chatdown tool which is part of Microsoft's bot-builder tools. You can use this tool in cases where you'd like to share a realistic "design comp" to internal business owners and stakeholders to get feedback and approval before spending the effort to build it. For dialogs that only require simple answers (e.g. text answers) for each step in the conversation, the Chatdown tool might not provide any more value than the jot-down technique described previously. But, for dialogs that will flow things like dropdown controls, radio boxes, buttons, and carousel cards, the Chatdown tool is very helpful in capturing that. Note: Not all channels support rich dialogs (i.e. dialogs with non-text conversation) so you'll want to consider that when designing your dialogs.

(Post Workshop) Define Actions and Supporting Activities

Actions carry out the intent of the scenarios. The goal of the dialog for a scenario is to gather enough information to be able to carry out its intent which you can think of as an action. These actions will be integration points between the conversational AI of your bot and the application backend that will execute that task.

Some scenarios will also require additional external context from a database or web service to get the information required to guide the next step in the conversation. These supporting activities are also integration points between the conversation AI of your bot and the application backend that will provide that intermediate information.

Early on in the development of your bot it's wise to simply mock these integration points so that you can take an agile approach and quickly iterate over your scenarios as you tune the conversation. Integration points are always challenging and generally requires a lot of effort to implement so mocking them will allow you to concentrate on getting the conversation right without requiring the integration points to be constantly reworked.

(Post Workshop) Review Design Guidelines

Although it's not necessary for the workshop, reviewing the bot design guidelines here will help beginners learn how to design bots that align with best practices and lessons learned by the Bot Framework team over the past several years. These guidelines could be master after the workshop and used to finalize the bot's user experience.

Build

1. (Preface) Conversational AI Fundamentals and Motivation
2. Create, Deploy, Run your Virtual Assistant
3. Customize your Virtual Assistant
4. Update QnA Maker Knowledgebase
5. Add a Custom Skill to Your Virtual Assistant
6. Implement Your Skill's Core Scenario
7. (Optional) Add Multiturn QnA Prompts to Your Assistant
8. (Optional) Add a Built-in Skill to You Virtual Assistant
9. Wrapping Up Build Stage

In this stage of bot lifecycle, you'll build a custom enterprise baseline for your new Virtual Assistant. When you finish the steps below, you'll have a core scenario working and all the development skills

you'll need to complete your bot. Simply rinse and repeat to incrementally add remaining bot functionality.

(Preface) Conversational AI fundamentals and Motivation

Before jumping into details of building your bot, it's important to first have an understanding of the big picture. Your workshop guide/instructor will take a few minutes to go over the fundamentals of Conversational AI and an overview of Microsoft Bot Framework so you'll understand key concepts and the various components of the Virtual Assistant architecture. You can also take a look at overviews of Conversation AI, Conversational AI Tools, Azure Bot Service, Virtual Assistant, and Microsoft Bot Framework.

Create, Deploy, and Run Your Virtual Assistant

Developing your bot starts by following the instructions here.

Note:
During the process of creating and deploying your bot you'll be asked to run PowerShell Core and if you haven't run that before you might not know how to launch it. PowerShell Core gets installed to $env:ProgramFiles\PowerShell\<version>\pwsh.exe and you can launch it from old PowerShell command line by entering **pwsh.exe**. After you've run PowerShell Core once, you'll be able to more easily launch it using Cortana search and will show up in the term completion from then on.

Update NuGet Packages

Although not exactly necessary, it's a good idea to update the NuGet packages for each project in the solution (including the Skills projects you'll create later). You can update the NuGet packages with these steps:

1. Open NuGet Manager for Assistant project by right-clicking the project and choosing **Manage NuGet packages…**
2. Select the **Updates** tab and then type in "Microsoft.bot" in the **Search** field to filter out all non-Bot Framework packages
3. Update these packages first and in this order to avoid import errors: Schema, Connector, Builder, Configuration
4. Click the Select

Customize your Virtual Assistant

Follow the instructions here

Update QnA Maker Knowledgebase

Follow the instructions here

Note: Over time, as you develop your assistant, you might decide to add additional QnA Knowledgebases (or refactor your current Knowledgebase into multiple Knowledgebases) and the instructions for how to do that are located here

(Optional) Collaborate

Although it not a concern you have to deal with to complete the workshop, many organizations will need to know how to setup QnA Maker so that several people can collaborate on a single QnA Maker KB. Instructions for doing that can be found here.

Add a Custom Skill

A key feature of the Virtual Assistant (VA) is its Skills based architecture. Skills are the heart of a VA bot and they are what give a VA bot its behavior. Skills are the unit of modularization for VA bots.

Typically, a bot will have a single core skill and one or more companion skills that complement the core skill and create a richer, more knowledgeable bot. In this step you'll add a baseline foundation for your bot's core skill which defines it core behavior by following the instructions here. In the next step you'll give this core skill its behavior.

Implement Your Skill's Core Scenario

The previous step added a custom skill to your virtual assistant but that was a baseline skill foundation that doesn't understand how to do anything useful or specific to your business. In this step we'll teach your custom skill to understand what users are saying and then the skill can act on that understanding to take action.

Understanding what users are saying is the job of Azure's Language Understand (LUIS) service. This service allows your skill to understand what users mean (i.e. their *intent*), no matter how they say it. Part of what happened in the previous step when we added the custom skill to the virtual assistant was the creation of your skill's LUIS application.

Add LUIS Intent for Core Scenario

The task now is to add a language model to your skill's LUIS application that can recognize when the user is asking for the core scenario of your bot. To do that, you'll need to add an Intent for the core scenario following the instructions here and optionally add Entities, if applicable, following the instructions here. Now you can test out the LUIS model in the portal to make sure it's recognizing utterances correctly and when that working you can move on to updating the skill to incorporate the new Intent so it can act on it.

Update Your Custom Skill to Incorporate Core Intent

The instructions for incorporating the new Intent created in the LUIS portal, found here and here, are wrong. The following is a workaround until those links and the **botskills update** command is fixed.

- Run this command from Skills project directory to update the .lu file
  - **.\Deployment\Scripts\update_cognitive_models.ps1 -RemoteToLocal**
- Run this command from the Assistant's project directory to update the assistant's dispatcher
  - **botskills update --botName <assistant's name> --remoteManifest "https://<skill's name>.azurewebsites.net/api/skill/manifest" --cs --luisFolder "<full file path to skill's project folder>\Deployment\Resources\LU\en"**
- Run this command from Skills project directory to update the LibraryBotSkillLuis.cs
  - **luisgen .\Deployment\Resources\LU\en\<skill's name>.luis -cs <skill's name>Luis -o .\Services**

Update Skill to Act on Intent and Begin Dialog with User

We're finally where the rubber meets the road and we're ready to code what action the skill should take in response to understanding the *intent* of the user. In the planning phase you designed a core conversation your skill needs to be able to handle. In this step we're going to create a Component Dialog that uses a WaterfallDialog to handle the conversation flow of your core scenario. To make life easy, we'll implement this core scenario using a dialog accelerator and step-by-step instructions that can be found here.

(Optional) Add Multiturn QnA Prompts to Your Assistant

*[**Important!!!!** The Virtual Assistant Template and bot framework tools have not yet been updated to respect multiturn follow-on prompts so the following is a temporary workaround and some follow-on prompts will not work properly. You can use this topic to explore follow-on prompts in your assistant, but the SDK will eventually be updated and obviate this workaround. Bottom line – this code should be removed before the assistant is deployed in production or once the SDK has been updated. When the SDK is updated, this section will be updated with the proper instructions on how to incorporate follow-on prompts]*

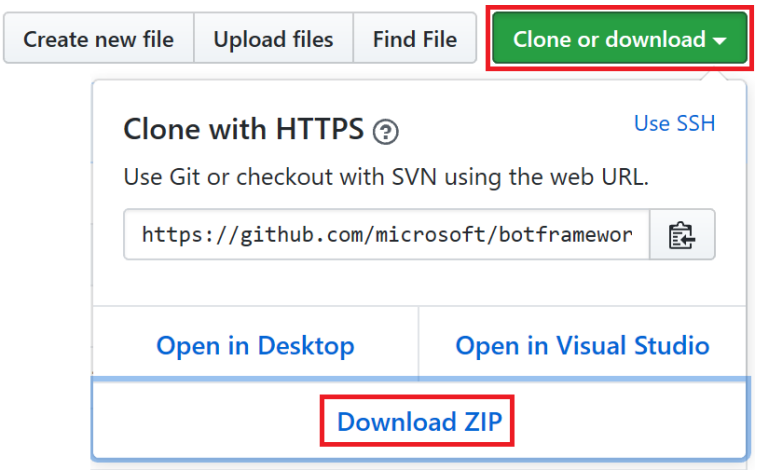
In this next topic we'll use follow-up prompts to create a multiturn QnA conversation as described here. The new QnA Maker Follow-on prompts do not automatically appear in bot clients so you must add code to make that happen. This topic will show you how to add the ability to show the new QnA Maker follow-on prompts in your new assistant. Follow the steps described here skipping the first step related to creating and deploying your assistant since you've already done that.

(Optional) Add a Built-in Skill to Your Virtual Assistant

Adding a built-in skill is an optional part of the workshop since and the built-in skill are not always a good fit for every bot. The Microsoft Bot Framework includes several built-in skills that can be added to your bot which can be found here. If you'd like to get a feel for how built-in skills works you might consider adding the To Do Skill since it's useful and straight forward.

Follow these steps to install the To Do Skill:

1. Browse to the Bot Framework Solutions Repository here and clone it to your development PC by clicking the **Clone or download** button and then choose **Download ZIP**.



2. Extract the ZIP to your local hard drive and then copy the **botframework-solutions-master\skills\src\csharp\todoskill** folder and paste it into the root folder of your Virtual Assistant solution (i.e. the folder that holds the **.sln** file)
3. Open your Virtual Assistant solution in Visual Studio and right-click the solution in the Solution Explorer and choose **Add | Existing Project…** and add **todoskill\todoskill\ToDoSkill.csproj**
4. Add the **todoskill\todoskilltest\ToDoSkillTest.csproj** the same way you did in the last step

5. Now we'll deploy the todoskill.
   Open PowerShell Core 6 and run the following command to temporality set the execution policy to **Bypass** for the current PowerShell session. If this is not done, you'll get an error for attempting to run a script that is not digitally signed. When the PowerShell session ends the setting reverts.

   ```
   Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
   ```

6. Deploy the todoskill by opening PowerShell Core 6 and change directory to todoskill project folder (**todoskill\todoskill**) and run the following command:

```
.\Deployment\Scripts\deploy.ps1
```

(Optional) What do these parameters mean?

Parameter	Description	Required
`name`	**Unique** name for your bot. By default this name will be used as the base name for all your Azure Resources and must be unique across Azure so ensure you prefix with something unique and **not** *MyAssistant*	**Yes**
`location`	The region for your Azure Resources. By default, this will be the location for all your Azure Resources	**Yes**

Parameter	Description	Required
`appPassword`	The password for the Azure Active Directory App that will be used by your bot. It must be at least 16 characters long, contain at least 1 special character, and contain at least 1 numeric character. If using an existing app, this must be the existing password.	**Yes**
`luisAuthoringKey`	The authoring key for your LUIS account. It can be found at https://www.luis.ai/user/settings or https://eu.luis.ai/user/settings	**Yes**

7. Now follow same instructions for adding a custom skill skipping to the Test Your Skill step found here

Wrapping Up Build Stage

Changing Endpoint

You will want to keep the key limits of the Authoring Key in mind when using it in development. When you use the Authoring Key to access the LUIS prediction endpoint you are limited to 1,000 calls per month. This has bit me more than once where I ran out of monthly quota when using the Authoring Key for development. Once I ran out of quota right in the middle of a customer demo!

Remember, LUIS Authoring keys and not the same thing as LUIS prediction endpoint keys. The LUIS Authoring Key is meant for authoring scenarios like creating and modifying your LUIS app whereas the LUIS prediction endpoint is used for querying your LUIS app for Intent recognition. So two totally separate keys.

The problem here is that the Authoring Key (with its 1,000 query limit) is assigned to the LUIS prediction endpoint by default when a LUIS app is created so Authoring quota is used on every call to the endpoint during development. For the uninitiated bot developer, they are unaware that calls to their newly created bots from the Bot Emulator are actually hitting against this quota right from the get go. On an active project it's not actually hard to hit the 1,000 query Authoring limit (as I mentioned I had at the worst possible time).

To make matter worse, because of the defaulted Authoring Key, developers are able to build bots connected to LUIS apps without ever understanding this Authoring Key dynamic so the first time they see the "out of quota error" in the form of an HTTP 403 or 429, they have no clue what has happened or how to fix it. Have you ever tried to Bing for a solution where you weren't even sure of what the problem is? Not pretty and certainly not a quick resolution.

IMHO, understanding the roles of the Authoring Key and the Endpoint Key and the import of how they are initially assigned is very difficult since its spread over several different document pages and the most important doc that shows how to actually create and endpoint and assign the key is buried in a "Quick Start Deployment" focused page.

The key thing to remember here is that LUIS prediction endpoints are created in the Azure portal and their endpoint keys must then be assigned to the LUIS app in the LUIS portal.

I am no longer using authoring/starter key for projects that will last longer than a single demo. It's a bit of a pain to have add a subscription key but well worth it and DEFINITELY something that needs to be communicated to customers.

Adding Secure Web Chat Control

A very common deployment scenario for bots is to embed a Web Chat Control in a web page. The Azure portal makes this easy by providing an HTML <iframe> snippet that can be copied and pasted into a web app. The problem with that approach is that it exposes your bot secret which would allow any client to connect to it. There is an alternative that allows you to embed the Web Chat Control in a secure fashion and instructions for do that can be found here.

Configure DevOps

Setting up DevOps as early as possible in development is critical to modern development and you can establish a core DevOps Continuous Deployment baseline by following the instructions here. As your bot development progresses, you'll need to mature your DevOps pipeline to target new deployment environments that might not exist early on. Having a core DevOps baseline will allow you to incrementally automate bot specific processes and discipline in a very pragmatic fashion.

https://channel9.msdn.com/Series/DevOps-for-the-Bot-Framework which has pointers to all stages.

Unit Testing Bots (to complete CI/CD pipeline)

https://www.microsoft.com/developerblog/2017/01/20/unit-testing-for-bot-applications/

https://blogs.msdn.microsoft.com/kenakamu/2017/06/20/devops-with-bot-framework-chat-for-business-application-1-en/

https://blogs.msdn.microsoft.com/jamiedalton/2017/03/08/devops-with-bots-cicd-pipeline-with-the-botframework-and-azure-end-to-end-walkthrough/

Analyzing Bot Usage

One of the most critical phases in the life of a bot is the Evaluation stage and a key part of that stage is analyzing bot usage, but because this workshop ends at the Build stage and because analyzing bot usage can be a useful tool during Usability Testing (i.e. testing with representative users or internal project stakeholders and business owners) we include a discussion of it here.

- Bot Analytics

- Overview and usage of the Bot Framework Power BI Template
- Custom Telemetry
- Advanced Custom Insights
- Active Learning for QnA Maker and LUIS

Appendix – Important Links

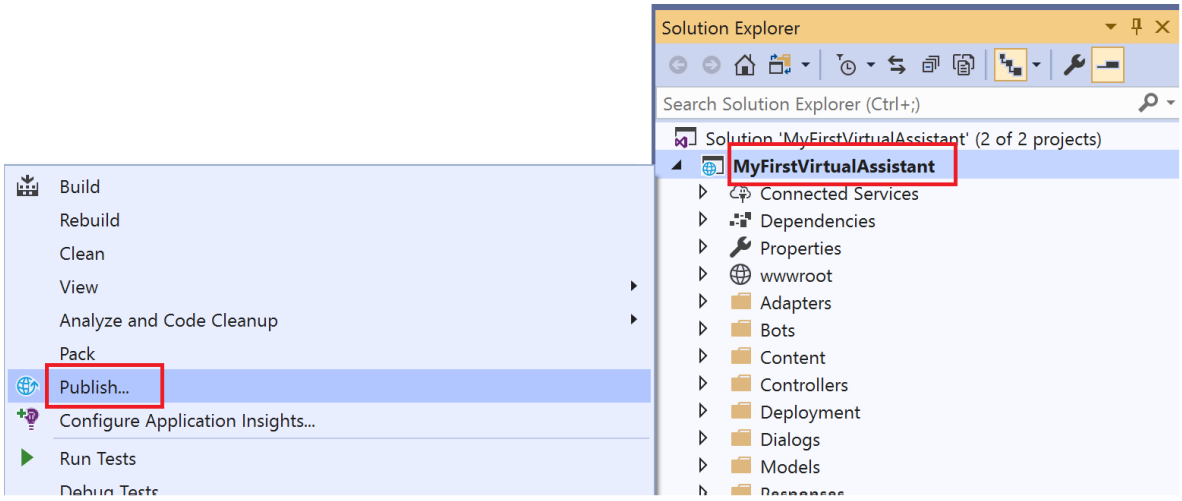
Bot Framework Solutions Repository for Conversational Assistant

- https://github.com/Microsoft/botframework-solutions

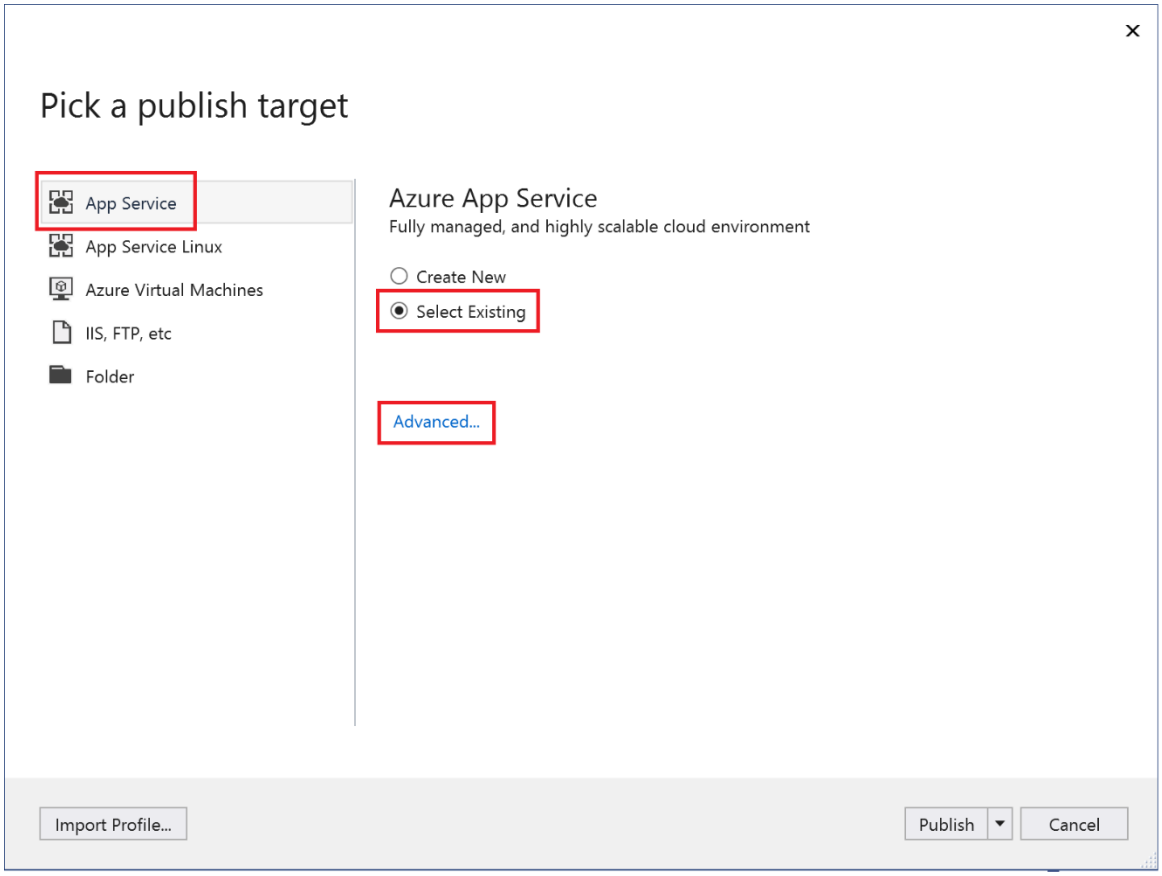
Appendix – Publish Virtual Assistant or Skill using Visual Studio

Publishing a Virtual Assistant using Visual Studio is slightly different that what most developers are used to doing when they use the Visual Studio Publishing Wizard since the App Service has already been created. This means the flow changes to allow for the existing App Service to be selected in the publishing wizard rather than created as is normally the case. Here are the steps.

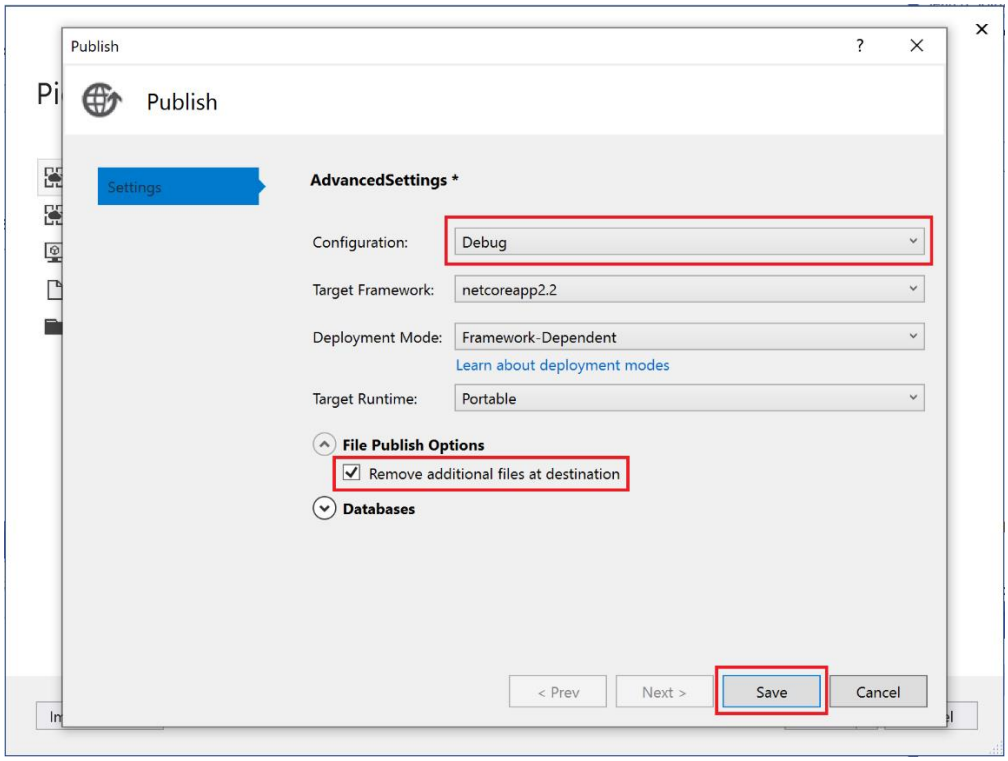
1. In Visual Studio, right-click the Virtual Assistant or Skill in the Solution Explorer and choose **Publish**



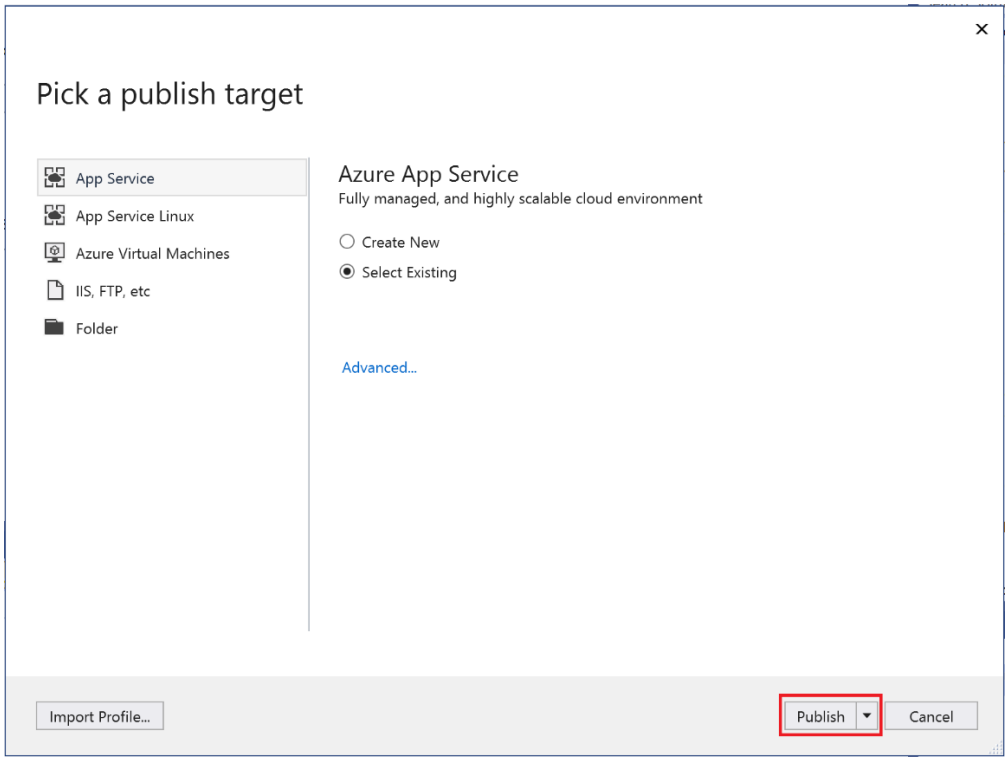
2. Make sure **App Service** is selected then choose **Select Existing** and then click the **Advanced**... link



3. Select **Debug** for the *Configuration* setting and expand *File Publish Options* and check **Remove additional files at destination** to insure there are no pesky leftover after any future deployment and then click **Save**.



4. Now click **Publish**



5. Select your Azure account and subscription then expand the App Service folder that contains the Virtual Assistant App Service you want to publish to and then click **OK** and the publishing process will begin. You can open the View | Output window to watch the progress and check to make sure it completes successfully.