**Honda Virtual Assistant**

Setup Instruction

**Members: Jason Xu, Jiankai Xu, Jeremy Lin**

**Table of Contents**

[Create a Global Admin Account 3](#_Toc58359320)

[Create a Web App Bot 3](#_Toc58359321)

[Create an App on Teams 3](#_Toc58359322)

[Create a custom App 3](#_Toc58359323)

[Create LUIS Module 4](#_Toc58359324)

[Setting up Azure Blob Storage 6](#_Toc58359325)

[Setting up Azure Cognitive Search 6](#_Toc58359326)

[Source Code Explanation 7](#_Toc58359327)

[File explanation 7](#_Toc58359328)

[Bots/resources/testcard.json 7](#_Toc58359329)

[Bots/resources/welcomeCard.json 7](#_Toc58359330)

[Bots/resources/ticketCard.json 7](#_Toc58359331)

[Bots/dialogAndWelcomeBot.js, Bots/dialogBot.js: 7](#_Toc58359332)

[CognitiveModels/ 7](#_Toc58359333)

[DeploymentScripts/, deploymentTemplates/: 7](#_Toc58359334)

[Dialogs/ 7](#_Toc58359335)

[SearchUtils/ 8](#_Toc58359336)

[.env 8](#_Toc58359337)

[Index.js 8](#_Toc58359338)

# Create a Global Admin Account

To customize a custom App in Microsoft Teams, having access to Azure portal, using Azure bot framework services, etc., one would need to have a **global administration** account. In our project, we created a **Microsoft 365 Business Standard** account (free subscription for one month). This [link](https://www.microsoft.com/en-us/microsoft-365/microsoft-365-business-standard-one-month-trial) will guide you to create a 1-month free account of @(chosen domain name).onmicrosoft.com.

# Create a Web App Bot

This [link](https://docs.microsoft.com/en-us/azure/bot-service/abs-quickstart?view=azure-bot-service-4.0) provides the detail of how to create a Web App Bot service. Note that after the deployment, make sure that you click on the tab **Channels** under Bot management section. To enable the bot on Teams, click Teams icon under **Add a featured channel** section.

# Create an App on Teams

## Create a custom App

There is various way to upload App to Teams. This [link](https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/apps-upload) provides documentation of Microsoft of uploading App. In our project, we used an App in Teams Store called **App Studio** to implement it.



Figure 1. How to install App Studio

Click on the **three-dot icon** on the side bar, on the tab **More apps** (figure 1)**,** one can add Apps into their Teams channel. Search App Studio in the search bar and install App Studio.

Open App Studio and click on the tab **Manifest editor**, it will lead you to the page of creating app. Fill out all the information in part 1 details, this section includes all information about the app as well as the information manifesting on the store page.

To connect the web app bot with the App, click on the **Bots** tab in section 2 capabilities. Click on **Set Up** and choose **Existing Bot**, select the bot which correct name and ID and click save. Make sure to choose the scope of bot. The detail is shown in Figure 2.

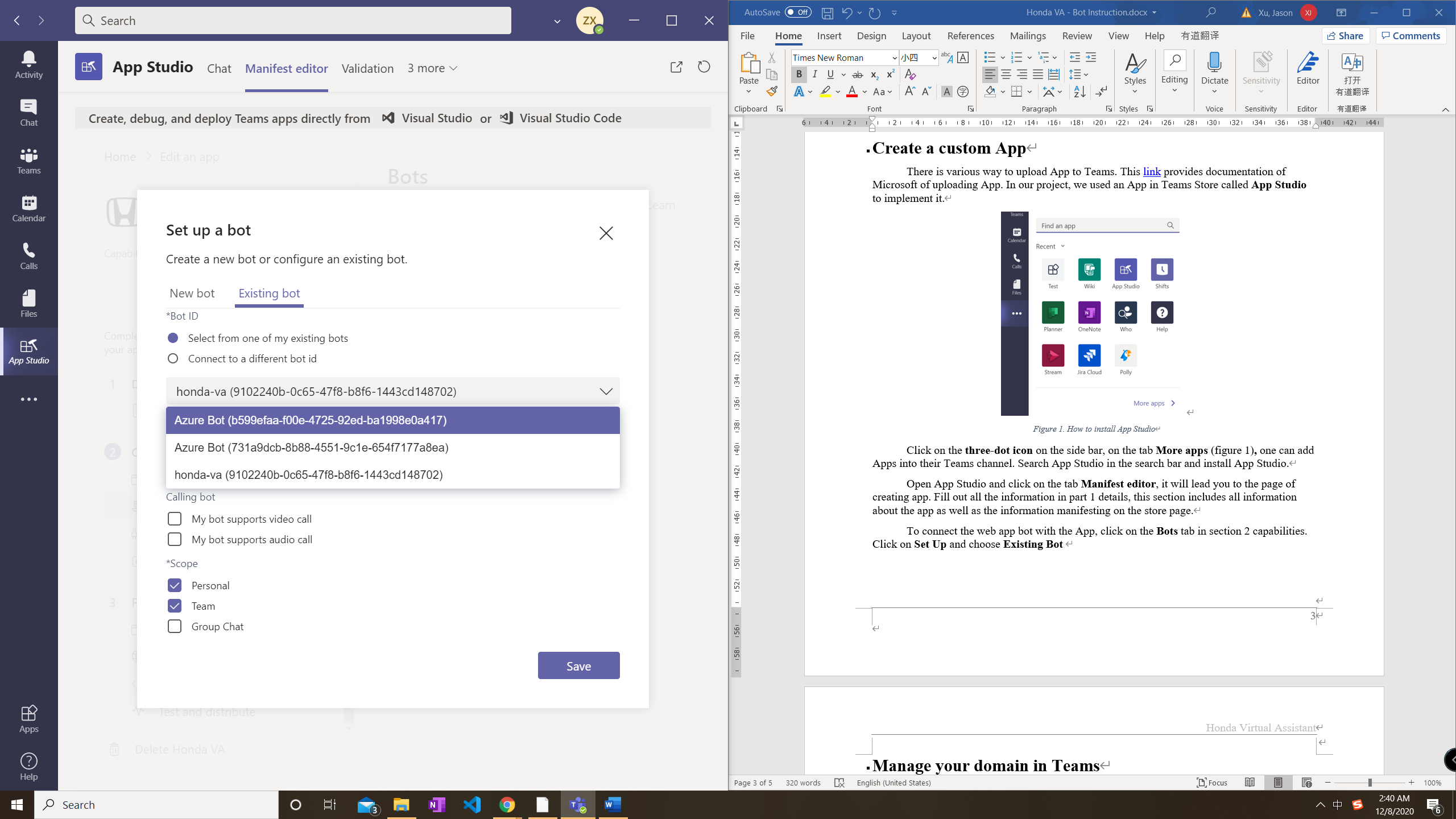
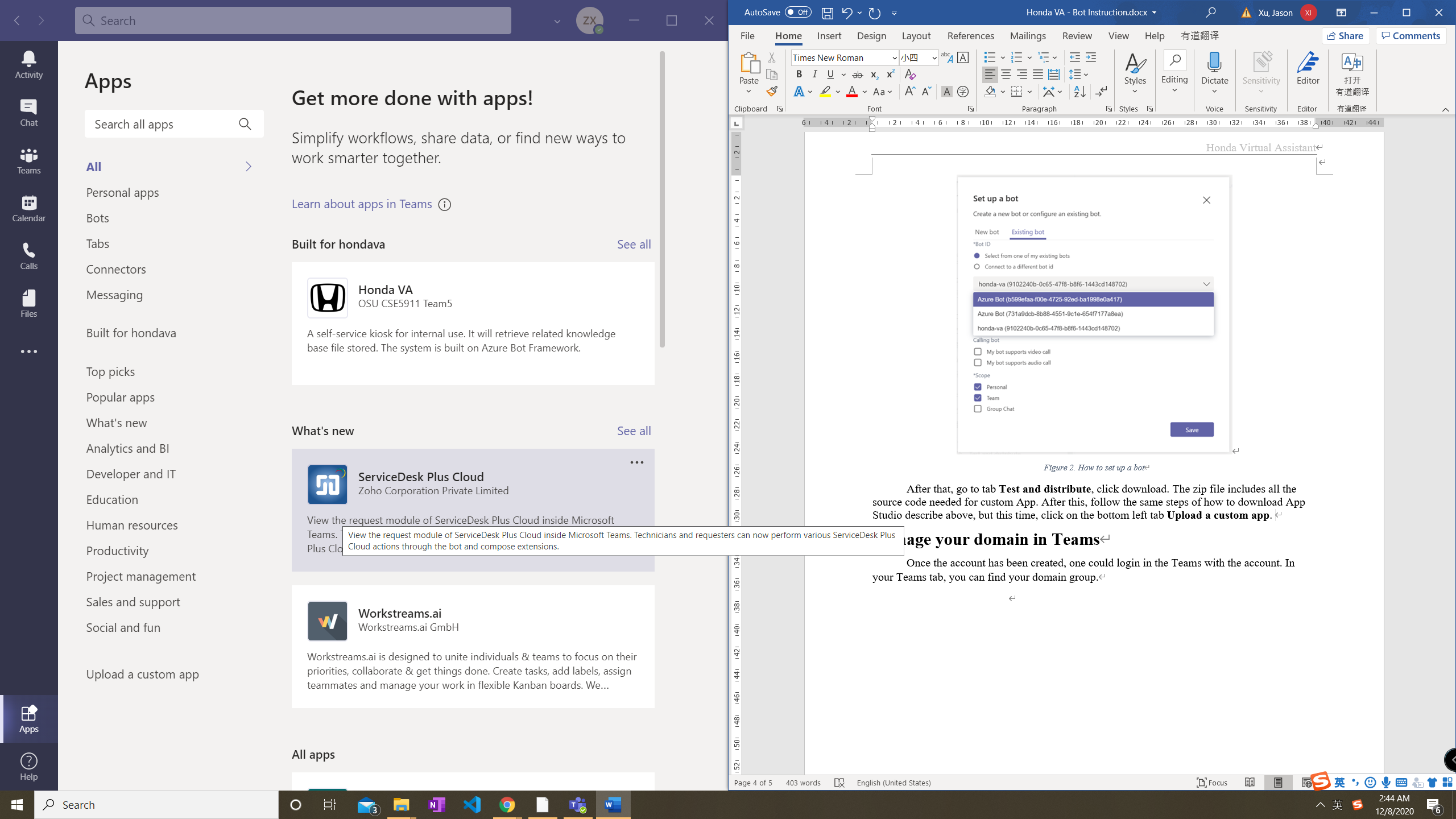
**** 

Figure 2. How to set up a bot Figure 3. Upload a custom app

After that, go to tab **Test and distribute**, click download. The zip file includes all the source code needed for custom App. After this, follow the same steps of how to download App Studio describe above, but this time, click on the bottom left tab **Upload a custom app** (Figure 3). Then an App connect to your web bot has been installed into your Teams.

# Create LUIS Module

This [link](https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-get-started-create-app) provides the detail of how to build a LUIS App. Since this is a language understanding module, it needs data and structure to train. There are some prebuilt domains for daily uses to import. But to create own intents, one should create **Entities** first. The intents tab can be configured with the sample input sentences. This link provides detail of how to achieve this. After adding training data, click on **Train, Test, and Publish** on top right corner. When you publish, make sure both your **Staging Slot** and **Production Slot** are up to date.

The assets that are needed to connect to the bot framework is the **App ID, App Key, and Location.** You can access to the App ID under Manage section on **Settings** tab. See Figure 4.

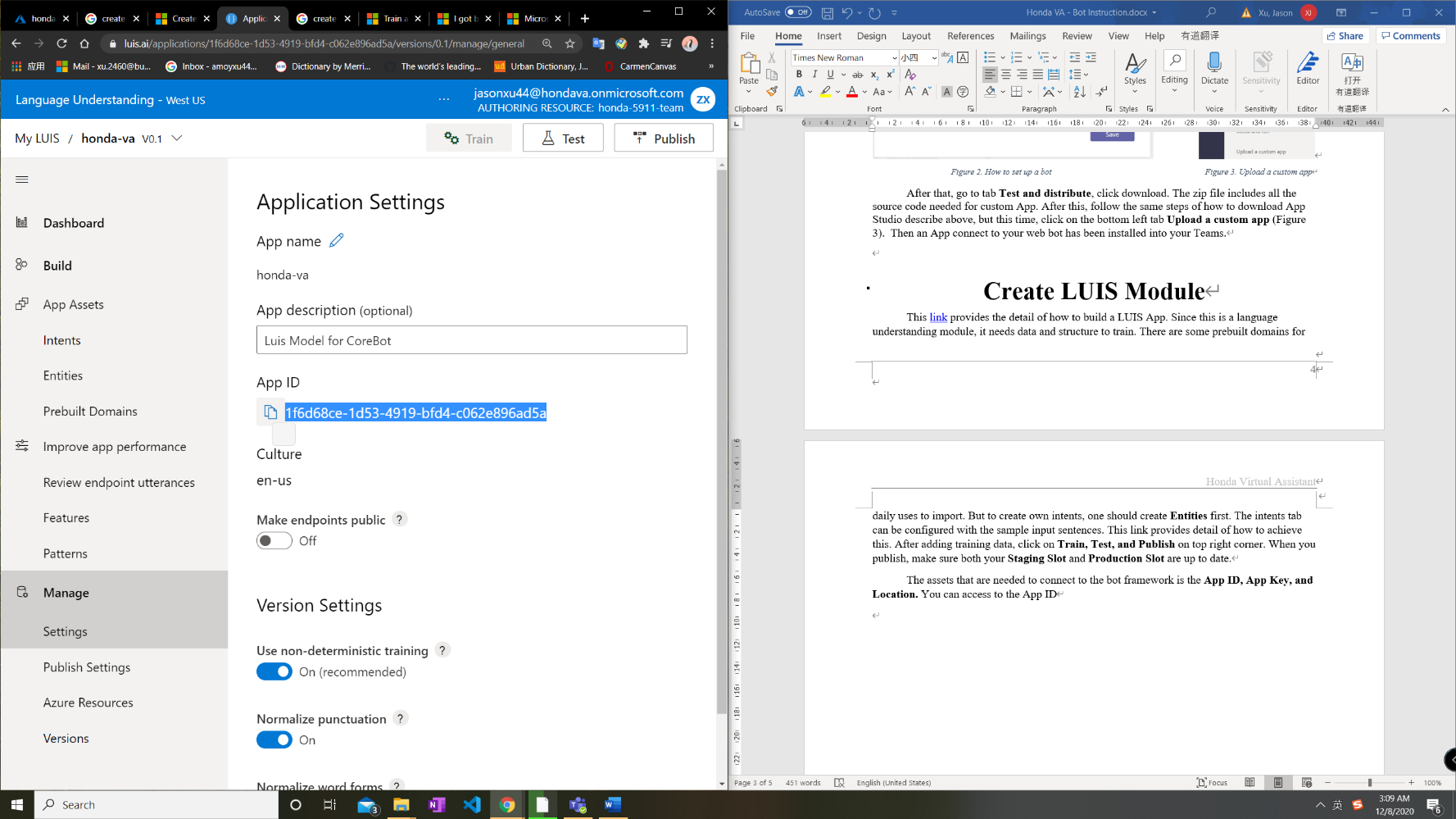


Figure 4. LUIS App ID

The App Key and Location is under **Azure Resourfces – Authoring Resource** tab. See Figure 5 – Location and Primary Key.

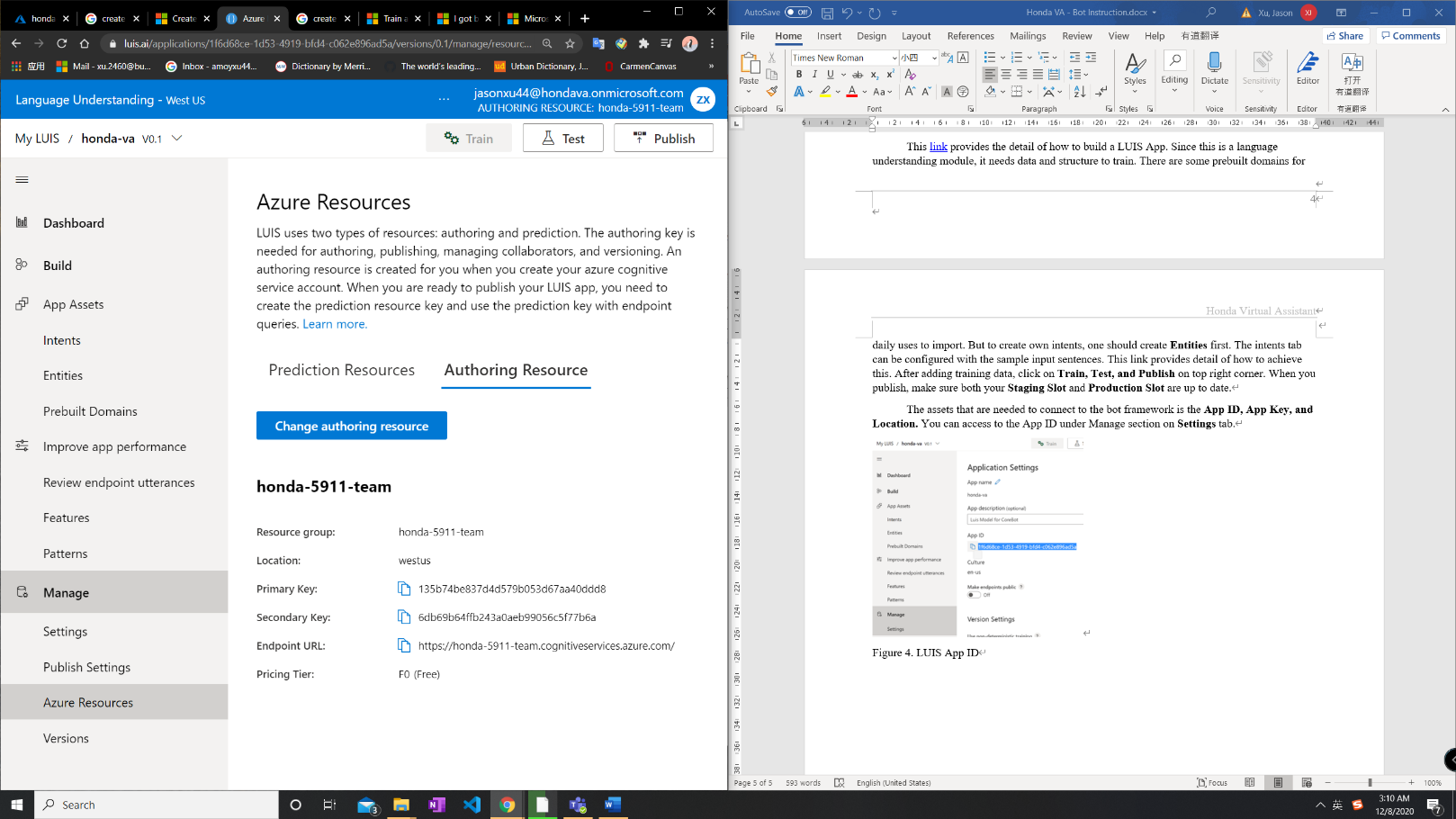


Figure 5. LUIS App Key, Location

These information are needed to be filled out in bot framework source code .env file. Some modification is needed for Location fields. Check source code for more information.

# Setting up Azure Blob Storage

Blob storage allows us to store blobs (Binary Large Objects, e.g. PDF, DOCX, etc) and serves as the location for knowledge files to be stored and be indexed by Azure Cognitive Search. See below to see how to set it up or view Azure’s official instructions [here](https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction).

1. Create a Blob Storage account
2. Create a container
3. Upload knowledge base files to that created container
4. Be sure to configure access level to appropriate setting. If you are using more than just dummy knowledge base files setting it to public is fine, but if you have confidential knowledge base files be sure to set it to private with no anonymous access.

# Setting up Azure Cognitive Search

Cognitive search provides indexing capabilities to blobs to our knowledge base and makes our document search functionality smart and fast. See below for how to set it up or view Azure’s official instructions [here](https://docs.microsoft.com/en-us/azure/search/).

1. Create a search service
2. Create a search index
3. Create an indexer using the index that was created in step 2 and set the data source to the knowledge base container created in “Setting up Azure Blob Storage”
4. Test search capability using the search explorer.
5. (Optional Step) If you want documents to be searched with leniency to typos, you can configure fuzzy search to compensate for those scenarios by setting the queryType argument to “full” and add a tilde (~) to the search keyword (e.g. “honda~”).
6. (Optional Step) Some organizations may have synonyms referring to the same term. Azure Cognitive also allows configurations of their terms through synonyms map which you can see [here](https://docs.microsoft.com/en-us/azure/search/search-synonyms).

# Source Code Explanation

This is an explanation of files inside of the project folder. We start building our project based on the starter code provided by Microsoft. (<https://github.com/microsoft/BotBuilder-Samples/tree/main/samples/javascript_nodejs/13.core-bot>) The starter code implements the settings of booking a flight. Our team modify its structure to achieve functional requirements of the Honda VA project. However, since we did not build our code from scratch`,

“This conversation” means the use for Honda VA. Some files are useless in this conversation but cannot be simply deleted because the way Microsoft builds its template prevent us from completely transform it into another use. How to safely get rid of redundant codes can be explored by the next team that take over this project.

## File explanation

### Bots/resources/testcard.json

Used as a template for generating result in an adaptive card format.

### Bots/resources/welcomeCard.json

Used as an adaptive card showing in welcome page.

### Bots/resources/ticketCard.json

Used as an adaptive card for submitting a ticket.

### Bots/dialogAndWelcomeBot.js, Bots/dialogBot.js:

Main bot constructor.

### CognitiveModels/

Folder that contains a json file for training a Luis intent. Not useful in this conversation.

### DeploymentScripts/, deploymentTemplates/:

Files that are generated by deploying the bot to certain platform. Not useful in this conversation.

### Dialogs/

A very important folder. This folder contains files that direct conversation flow and Luis integration. All files are from Microsoft’s starter code for booking a flight. We did modification on them.

**BookingDialog.js:** Construct a flight booking dialog. Not useful in this conversation.

**CancelAndHelpDialog.js, dateResolverDialog.js:** Not useful in this conversation.

**FlightBookingRecognizer.js:** The getBehaviorEntities function is created by us. It could recognize the entity resolved by Luis. Other functions are not useful in this conversation, they come from the starter code.

**MainDialog.js:** Important. These files walk through the dialog flow. Although integrated with flight booking related information, you can track the flow in this way:

1. Async function “run” starts the dialog.
2. Async function “introStep” greets the user and prompt for input.
3. Async function “actStep” read the input, send the input to the Luis app for intent recognition, and Luis app returns a result. Then, based on the intent it resolved, go to different cases. Case “Bookflight” is pre-built, not useful in this conversation, and so does intent “GetWeather”. Case “go\_cognitive” is our points of interest. When the Luis app recognize our input’s intent as going to cognitive search, it will then call the cognitive search client. The Luis app will also extract the entity, in this case, the document keyword. The search client will search from the Azure Cognitive Search directory with the extracted keyword. The search results will be in a list, we called it “a” here. After that, adaptive cards will be generated based on the result and rendered to the user. The function “generateCard” simply modify the text content based on the search result.

### SearchUtils/

Codes explaining how to search on Azure cognitive search. Not in use now.

### .env

Save the related app’s id and key. Important. Here we save our Luis APP id, key, hostname, Cognitive Search Key, and endpoint here.

### Index.js

Required packages and dependencies