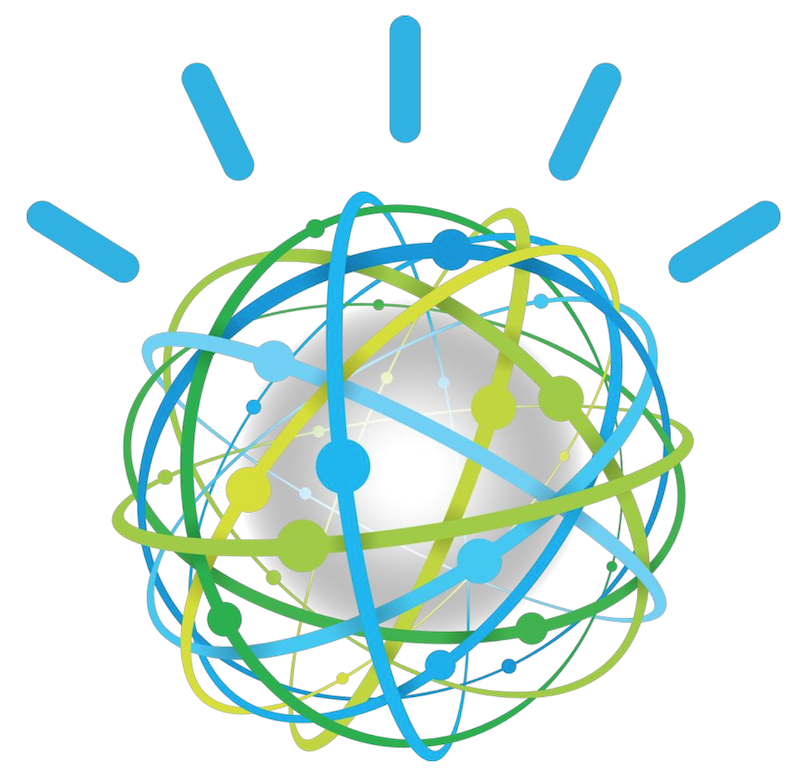
# IBM Watson Solutions

Business and Academic Partners

Building Advanced Dialog in Watson Conversation Service

Prepared by Armen Pischdotchian

Version 1.0 November 2016

# Overview

What is Bluemix you ask? [**Bluemix**](x-msg://450/ibm.biz/bluemixchicago#_blank) is an implementation of IBM's Open Cloud Architecture, leveraging Cloud Foundry to enable developers to rapidly build, deploy, and manage their cloud applications, while tapping a growing ecosystem of available services and runtime frameworks. You can view a short introductory video here: [**http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html**](http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html#_blank)

Additionally, for our academic partners, there are no-charge 12-month licenses for faculty and no-charge 6-month licenses for students - all renewable and NO CREDIT CARD required!

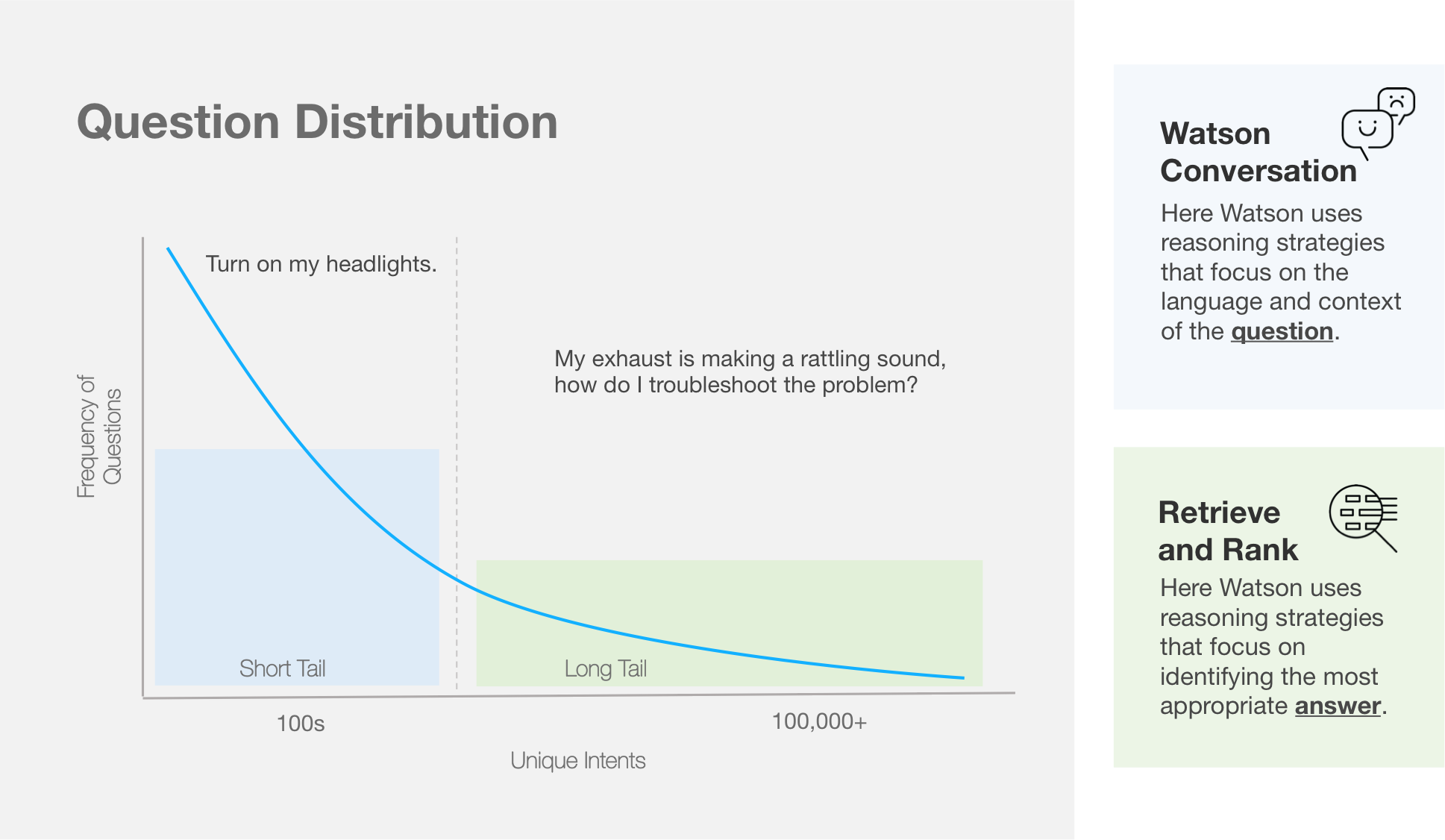
To get started, you will need to become an Academic Initiative member. Refer to this document for details:

<http://it.husc.edu.vn/Media/TaiLieu/IBM_Academic_Initiative_for_Cloud_Process.pdf>

The purpose of this guide is not to introduce you to Bluemix, that foundational knowledge is a prerequisite study on your part and you can obtain it from the links mentioned above. This guide is more of an instructional approach to working with the IBM Watson™ Conversation service where you can create virtual agents and bots that combine machine learning, natural language understanding, and integrated dialog tools to provide automated customer engagements. Watson Conversation provides an easy-to-use graphical environment to create natural conversation flows between your apps and your users. Creating your first conversation using the IBM Watson™ Conversation service entails the following steps:

1. Train Watson to understand your users' input with example utterances: Intents and Examples
2. Identify the terms that may vary in your users' input: Entities
3. Create the responses to your user's questions: Dialog Builder
4. Test and Improve

IBM Watson Conversation service is designed to answer the short tail of typical question distribution per below.



In this workshop, the emphasize is on building a robust dialog from scratch. You will become intimately familiar with the notion of Intent, entity and an in depth dialog tree that uses advanced conditional context within the node.

It is strongly recommended that you watch this 14-minute video: <https://youtu.be/ELwWhJGE2P8>

At the end of this workshop, spend some time and consider what other services you can use to augment a better approach for bringing further cognition to your app; for example, Retrieve and Rank to extract information from specific documents (the long tail), or, you may want to include Speech to Text upfront and Text to Speech for the returned responses. Enjoy the cognitive journey you are about to undertake.

# Prerequisite

For this workshop all you need is a Bluemix account; you will not be building an app, but working within the conversation service in Bluemix to build a detailed dialog tree.

* **Obtain Bluemix credentials:**
  1. Direct your browser to the Bluemix home page: <https://console.ng.bluemix.net/home/>
  2. Click **Sign Up** on the top right. If you are affiliated with a university, use your edu email.
  3. Enter requested information: for **Org**, select the suggestions that are provided for you, for example, your email address; for **Space** name, you can also select the provided suggestions, such as dev; alternatively, you can specify your own values and click **Create Account**.
  4. Look for the email confirmation. You will have to login once again into Bluemix after clicking the link from the email.
* **Download the Conversation\_diaolg.pdf document from Github**

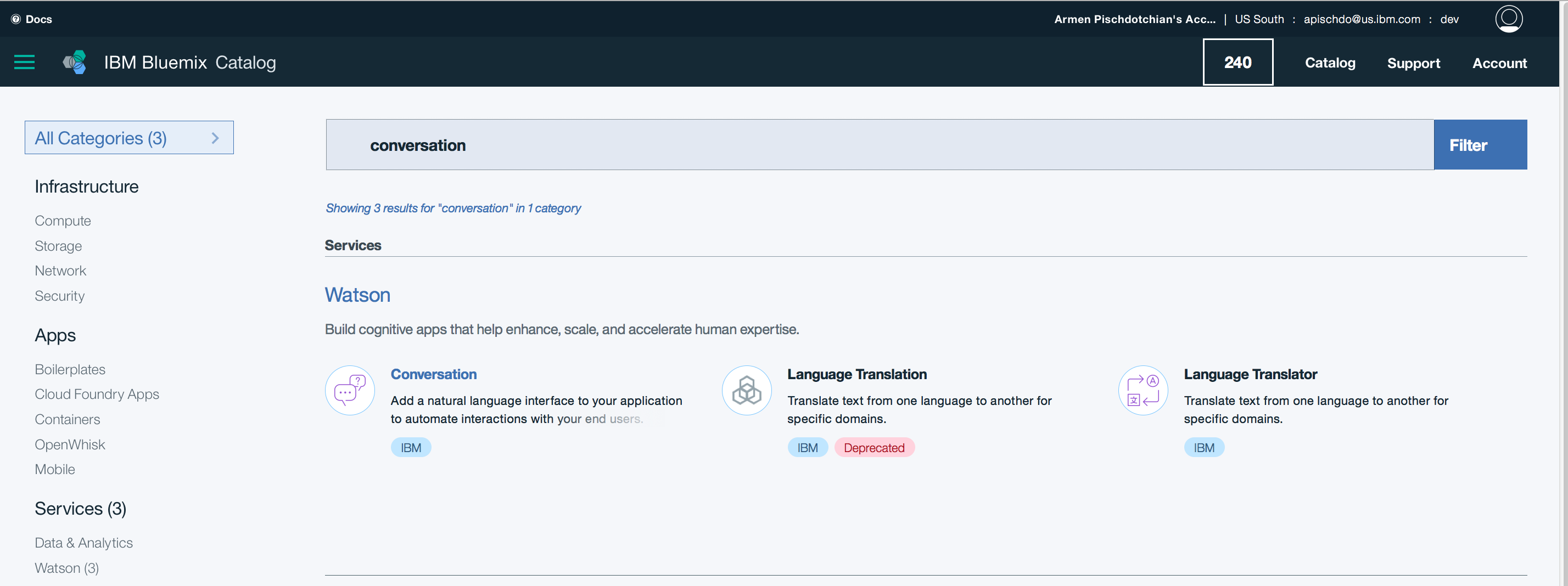
This package contains basic code that you will need to build your app.

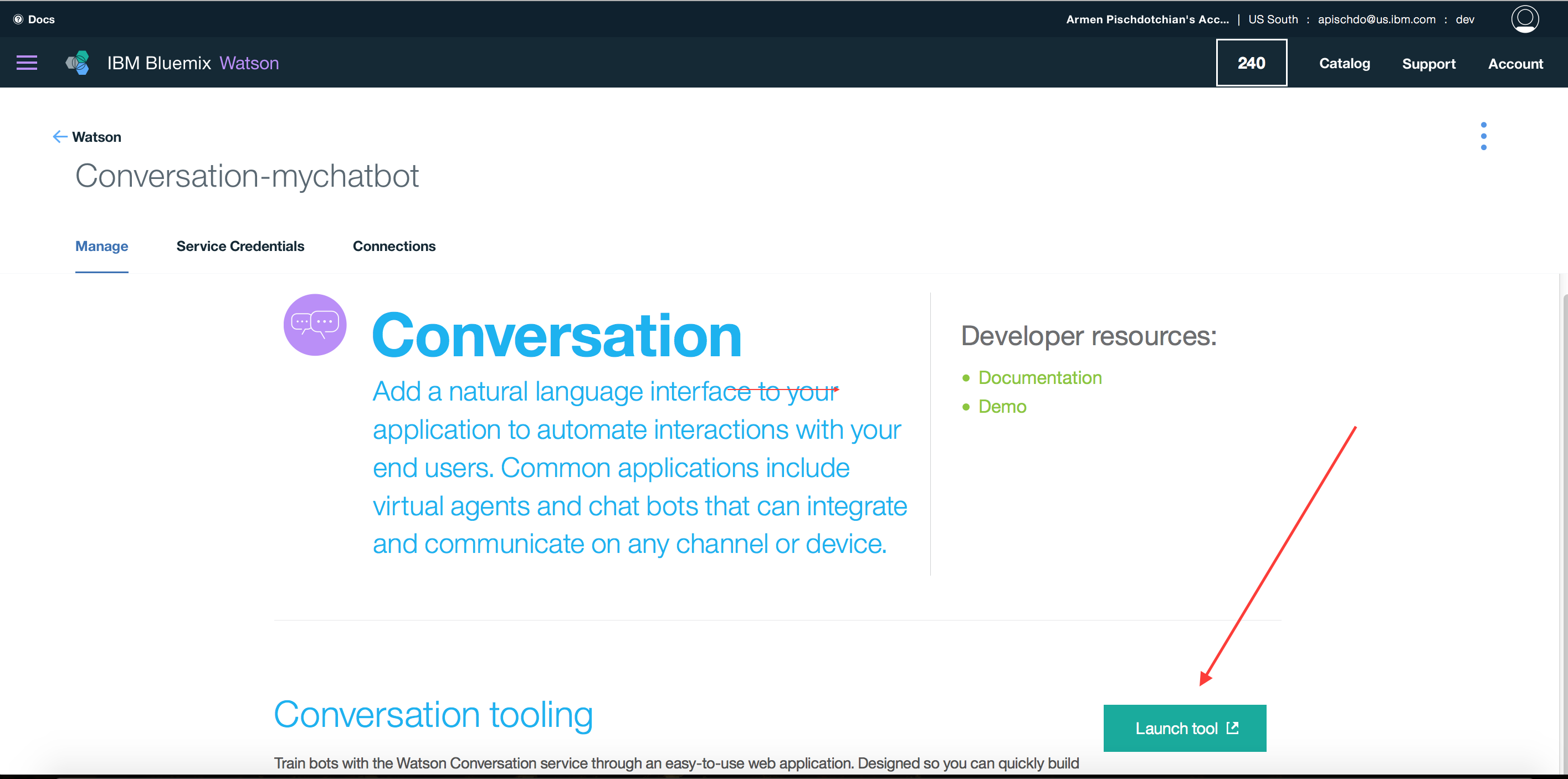
* 1. Direct your browser to a GitHub repository (no need to sign up): <https://github.com/>
  2. Search for **bluemix-workshop**.
  3. Scroll down and select: [apischdo/Bluemix-workshop-assets](https://github.com/apischdo/Bluemix-workshop-assets)
  4. Download just the **Conversation\_dialog.pdf** document.
  5. Follow the instructions in that document.

# Using the Conversation service

Let’s begin our journey:

1. Login into Bluemix: <https://console.ng.bluemix.net>
2. Click the **Catalog** tab.
3. Search for the **Conversation** service and click that tile.

Edit the Service name to something meaningful to you (for example: Conversation-mychatbot) and click **Create** (If you have just created your account and accessed it from the confirmation email, you may need to log into Bluemix once again, then you can see the Create button in the bottom right corner).



1. Click the **Launch Tool** button.
2. Login once again using the same credentials that you use to login Bluemix.
3. Click **Create** to create a new workspace and give it a name, in this example: MyCarDemo
4. Click **Get Started**.
5. Click **Create New**.

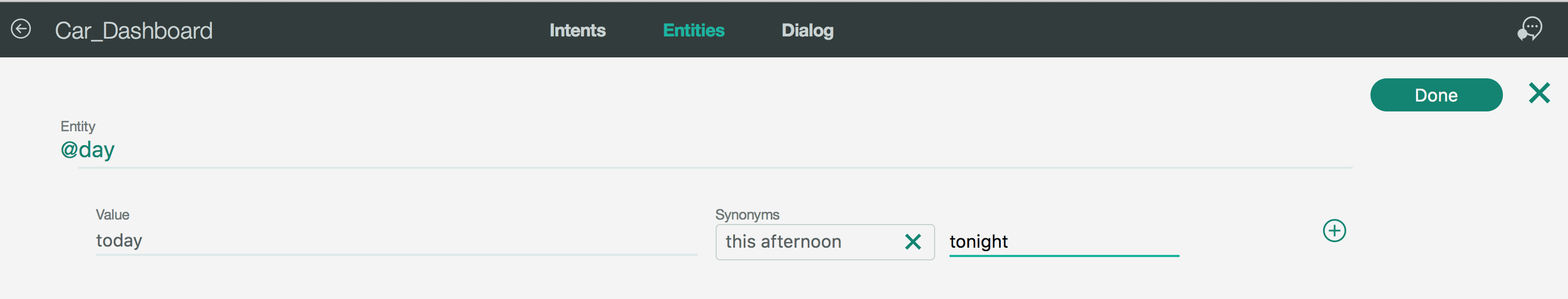
Let’s use a table format now to capture the steps.

|  |  |
| --- | --- |
| **Steps** | **Example screen capture** |
| 1. Add a new Intent and name it **capabilities**. 2. Add a few examples of how you might ask your car what are its capabilities; for example, help me, what can you do? And so forth. Minimum of 5. The more the better. 3. Click **Create.**   Next you will create two new intents of just what it is that the car can do; for example, turning things on and off is a good place to start. |  |
| 1. Add a new Intent and name it **turn\_off** (no spaces please). 2. Add a few examples of how you might ask your car to turn off certain things; for example: off, power off, and so forth. Minimum of 5. The more the better. 3. Click **Create.** 4. Add another new Intent and name it **turn\_on** (no spaces please). 5. Add a few examples of how you might ask your car to turn on certain things; for example: begin, on, play, power on, and so forth. Minimum of 5. The more the better. 6. Click **Create.**   You are now ready to create Intents. Intents are about “what can the system do,” Entities are about what “things” can be done now that the system knows what to do. If the system, knows that your intention is to turn on something, then the entity specifies what those things are: heater, engine, air conditioner, music, all sorts of things that you can turn on (or off) in a car. |  |
| 1. Click the **Entities** tab and click **Create new**. 2. Type **appliance**. 3. Add new values for air conditioner, such as *ac*, *cooler*, *fan*. 4. Add another value of **heater** and add synonyms such as *heat*, *hot air* and so forth. 5. Repeat the above steps and add another value of **music** with synonyms such as *radio*, *song*, *songs*, track and so forth. 6. Click **Create**. |  |
| 1. You are now ready to create the Dialog tree. Click the **Dialog** tab. 2. Click **Create**. This is a Dialog node, the top half is merely a Boolean condition: true or false. The bottom half is the response that has a simple and an advanced view (where you can include JSON objects to add dynamic output and contextual variables for multi-turn conversations). 3. Specify a new condition (the third option from the drop down) after you type **conversation\_start**. Anything else appears always by default. Close that description box. 4. In the context section, type: Hi, welcome to the simple car demo. 5. Let’s test the node. In this lab, each time you add a node, habitually click the chat icon in the top right corner 6. Type something, just bunch of letters to see what happens. 7. So not much happens, because you don’t have any other dialog nodes. But the Anything else box opens. That is a catch all node and is always set to true incase the input is not understood. Enter the following text in the Anything else node: **Sorry, I am having trouble understanding you**.   OK, so let’s add more Dialog nodes. You will now add sibling nodes (sits underneath) and children nodes (goes across). |  |
| 1. Create a sibling node (click the plus sign underneath) and type **capabilities**. If you just type *cap*…the rest appears in the drop down and select the first. 2. In the context section, type: **I can turn on and turn off appliances in the car.** 3. So when the response came in, the first node *conversation\_start* was false so it jumped to the next node; the *capabilities* node had a condition true, so it executed that condition, hence the response that you got. 4. Clear the chat box and close it. |  |
| 1. Let’s build on this dialog by clicking the plus sign under the capabilities node and specify the **turn\_on** condition (the second choice from the drop down). 2. Click the plus sign to the left a child node, because you want to specify turn on which appliance. Because it might have an intent but not an entity, so let’s condition it on appliance. Pick the first one (just appliance) Notice you can be more granular and pick one that does have an entity such as appliance:music. 3. So if there is an intent on an appliance, then type in the context box: **OK turning on the @appliance**. | the screen capture is missing the @ in front of appliance. Include it so it knows the value of the appliance. |
| 1. Test the dialog by invoking the chat window. Type: **turn on the ac**. Notice it gets the intent and the entity right, but no response.   The reason is that there is an implicit *wait for user input* (see the little chat icon in between the two nodes). This means, once the turn\_on node is evaluated for true, then it is going to wait for the user input before moving to the child node (@appliance). You overcome this problem by adding a *continue from*. This is super useful where you can move to different location in the dialog tree.   1. Click the three dots inside the #turn\_on node and click **Continue from…**   Ok let’s take a moment here: you can continue from the condition part of a dialog node (the top half) or the response part of the dialog node (the bottom half) as well as continuing from a user input. This means you can first evaluate the condition of appliance and give a response from that, or bypass the condition and just give the response that is typed on the bottom half.   1. Let’s try one scenario here: click the three dots in the @appliance node and select **Go to condition**. |  |
| 1. Let’s test it. Click the chat icon and type: **turn on the ac**. 2. Notice the response comes back as expected.   So consider this: What will happen if there is no appliance entity? It will evaluate the *turn\_on* node, then be forced to fall back to the root node and if nothing else matches the turn\_on condition, then the *Anything else* node will capture it.  So let’s add a default dialog of our own as a sibling to the @appliance dialog, such that if nothing matches, it will capture it right there and not go back to the root. |  |
| 1. Add a sibling node to the appliance node, set the condition to true (type it and select the third option of condition) and type the following in the response section: **I understood you intent was to turn on something. Try saying turn on the ac**.   This is just some text that the car provides when the user does not provide an appliance to do something with.   1. Create the turn\_off condition node underneath the turn\_on node    1. Create a child node of @appliance    2. Ensure it is *Continued from* the *condition* part of the parent node (Steps 39 and 40).    3. Add a true sibling condition node.    4. Copy paste the responses from the turn on node, except have it say turn **off** instead of turn on. 2. Test the dialog and ask to **turn off the ac**. Expect to see *OK, turning off the air conditioner*.   Stopped at 54:40 in the video |  |

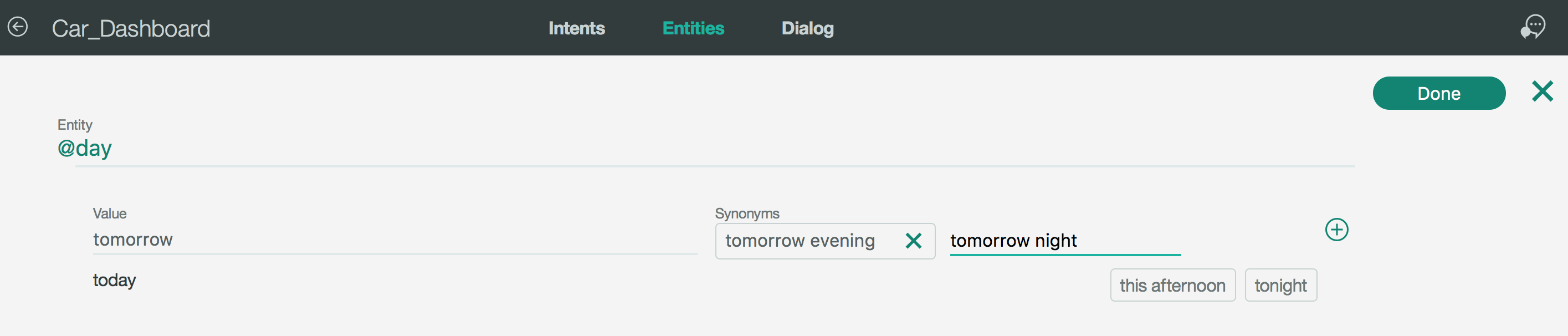
# Updating Dialog

Working with Intents, Entities and Dialog is covered in a separate workshop that elaborates on all facets of creating a robust dialog, for the purposes of this lab, you will edit the Entities and the Dialog.

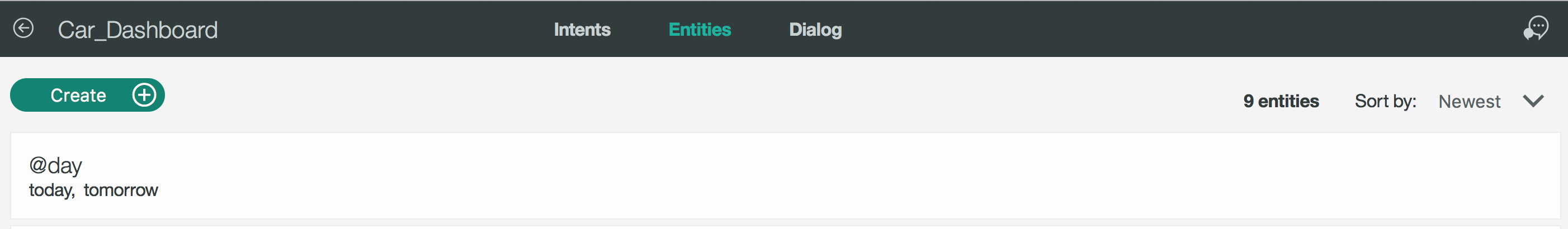
1. Go to the **Car\_Dashboard** conversation and click the **Entity** tab.
2. Click **Create new+** and add an *Entity* of **day** (case matters, use lower case).
3. Add the value of **today** for the Entity of day (case matters, use lower case).
4. Add synonyms such as **this afternoon** and **tonight** (case does not matter here, pick your own synonyms).



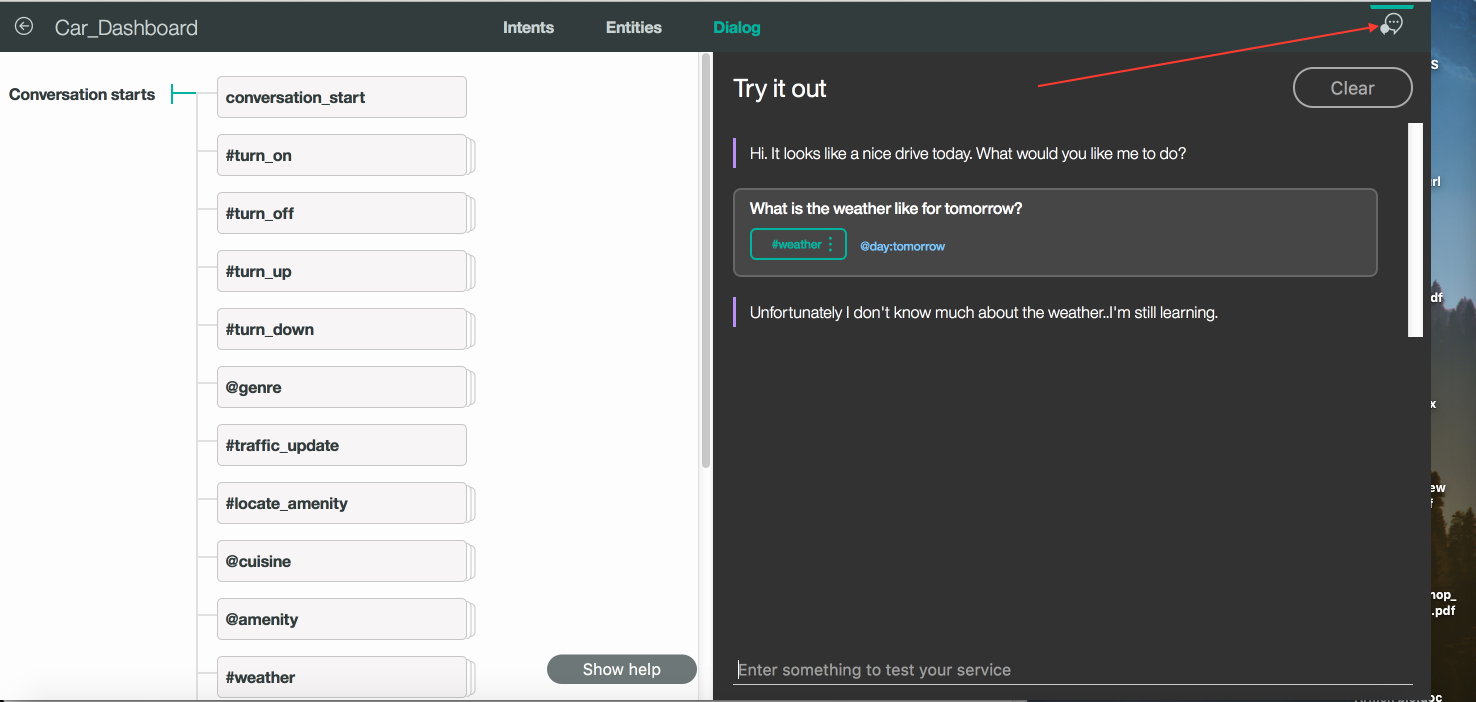
1. Click the plus sign and add another entity *value* of **tomorrow** (case matters, use lower case).
2. Add synonyms such as **tomorrow evening** and **tomorrow night**.



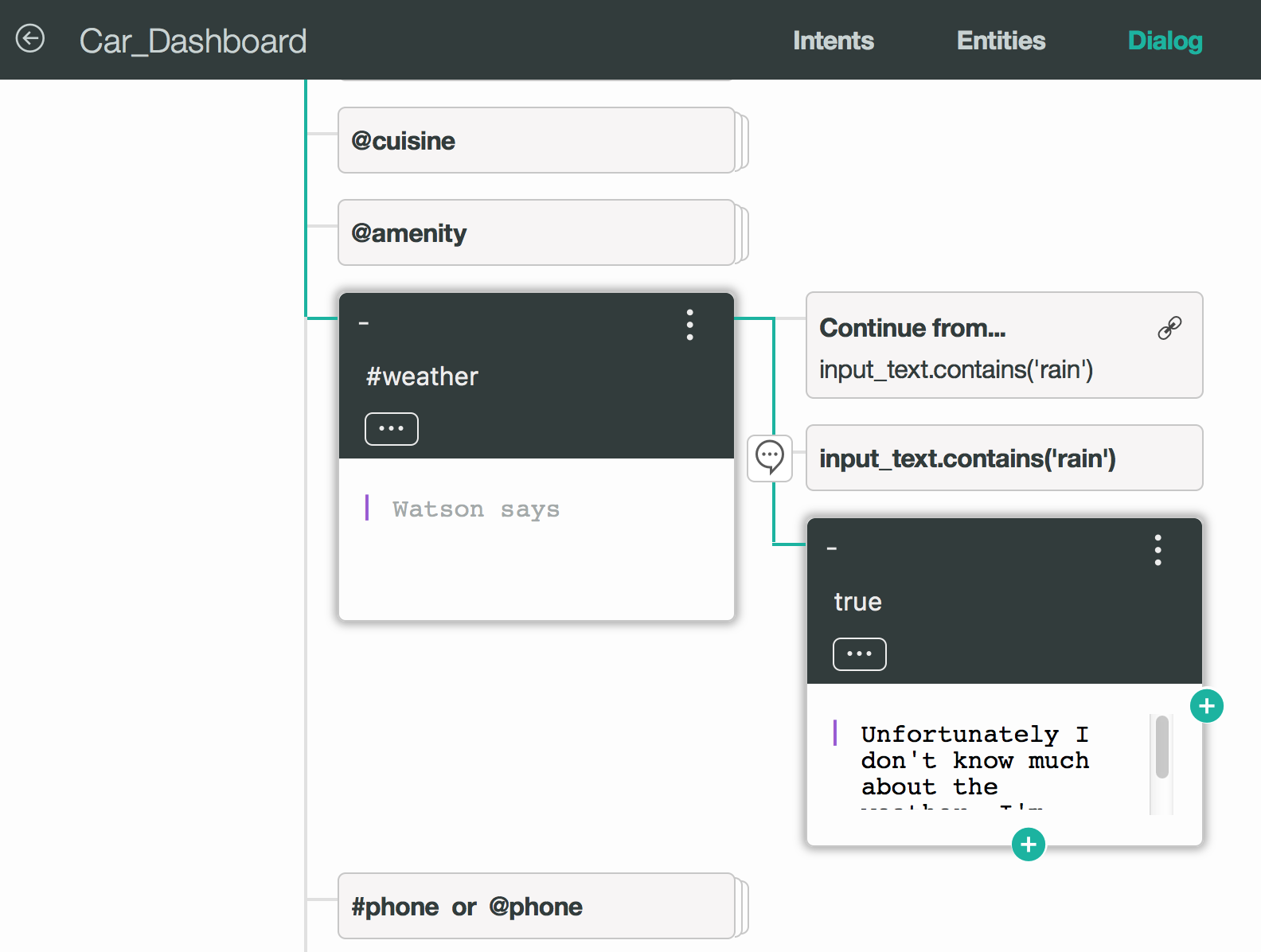
1. Click **Create** (or Done…the UI may have changed).
2. Note, # sign signifies Intent and @ sign signifies Entity. The end result looks like the image below:



1. Let’s update the Dialog flow. Click the **Dialog** tab.
2. Click the chat icon to the top right corner of the page and do a trial run and ask the system “how is the weather tomorrow?” The result should appear as below:



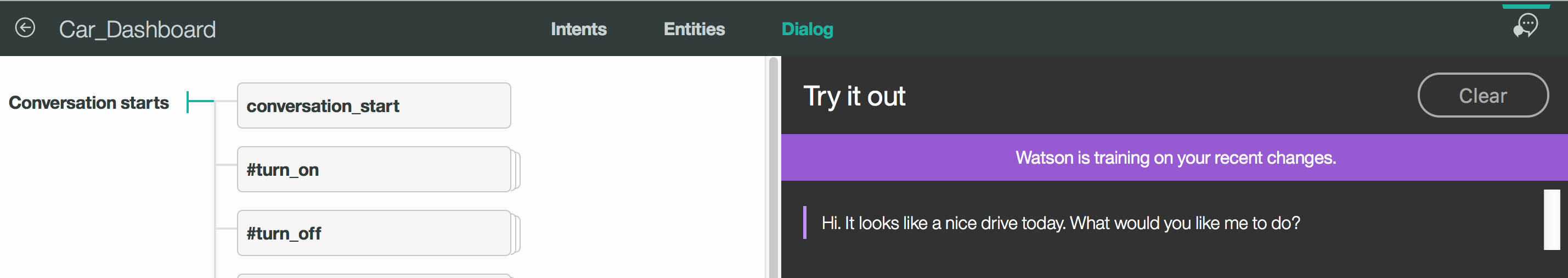
1. Close the chat window and click the #weather Dialog node and expand the tree, notice where the chat got it’s response!



1. Click the three dots in the #weather node and delete the **#weather** node.
2. Click the amenity node (or any node) and then click the plus sign in the bottom of the node, to add sibling Dialog node (if it goes across, it is a child node). Nodes that are siblings (they are stacked on top of each other), are new conversations. Nodes that traverse across across are child nodes (pertain to the same conversation) and are conditioned on the parent node.

|  |  |
| --- | --- |
| 1. Type and then select the **#weather** for Intent. 2. Click the plus sign *inside the green* node and specify the condition of **@day:today** 3. Add the dialog:  The weather for today in {0} is {1}   *(Type the syntax, don’t copy/paste it from here).*  There is the concept of conditions, if the condition is true then the dialog code gets executed. |  |
| 1. Click the plus sign *underneath the dialog* node you just created and add another node: #weather 2. Add the condition of tomorrow: @day:tomorrow 3. Include the same script:  The weather for tomorrow in {0} is {1} |  |

If you have the chat open or if you open it right away, notice the message where it says **Watson is training on your recent changes**. It will become green and state that training has finished and disappears shortly thereafter.



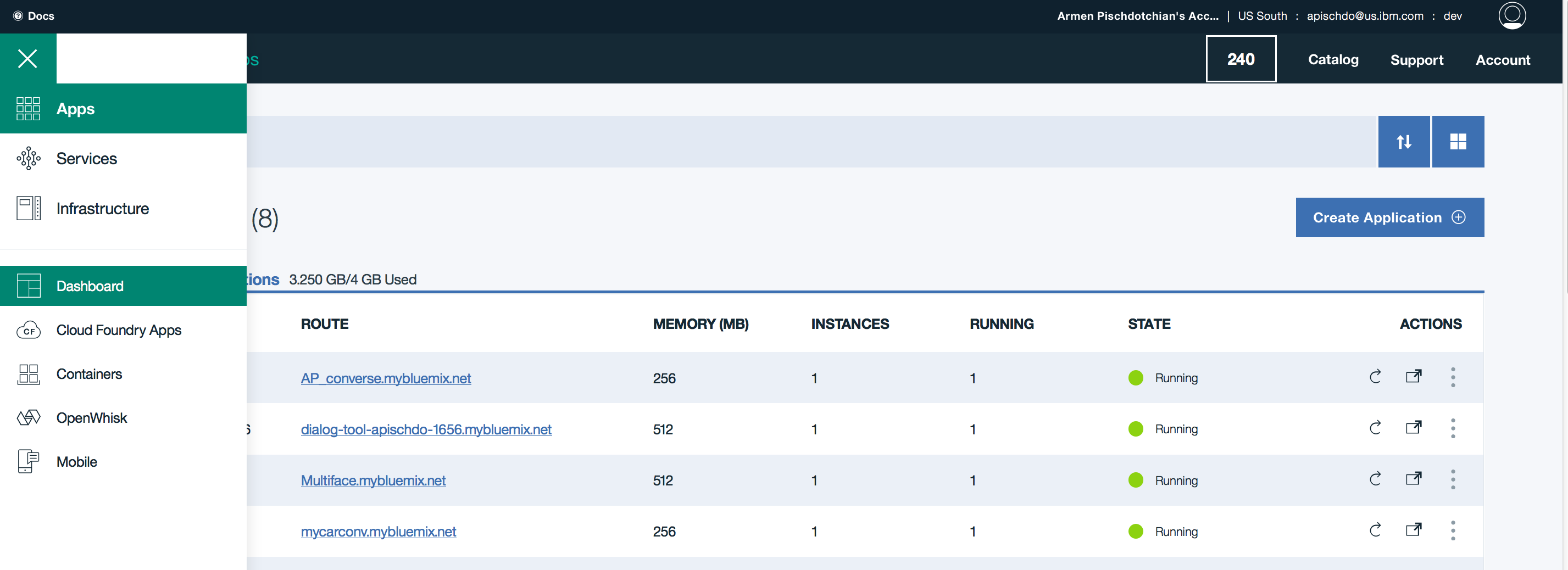
1. At this point, go back to your command console and run the node server.js command again (you may have to Control+C to stop the current process).
2. Ask the question: “How is the weather today?” and notice the correct weather forecast for your location.

But if you ask the same question from the Conversation service, it will give you the syntax answer: The weather for tomorrow in {0} is {1} and that is because the service on Bluemix does not know about your local app; not yet anyway. Your next step is to deploy your app to Bluemix.

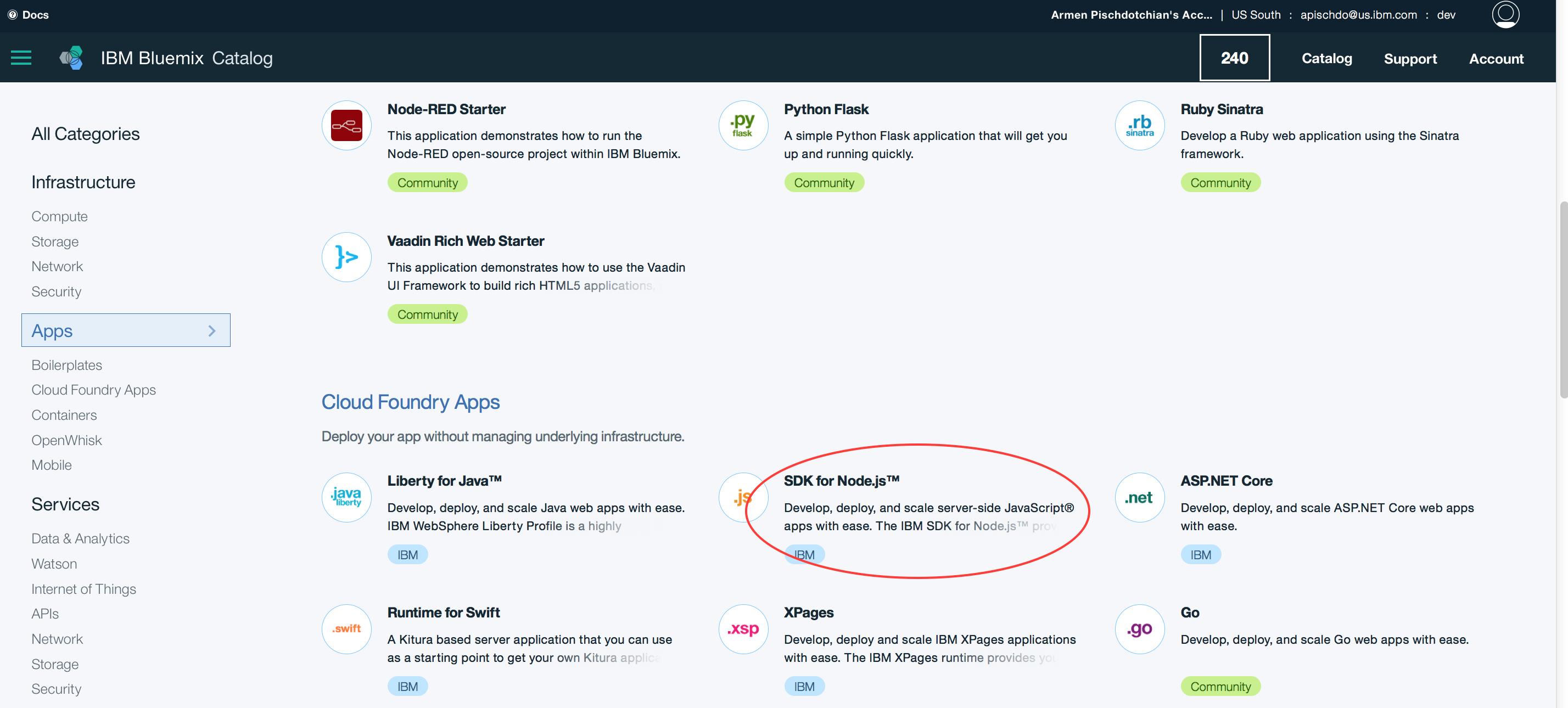
# Deploying your app to Bluemix

An easy way to deploy your app is by including an app name in the manifest.yml file and then using the cf push command to deploy the app onto Bluemix. However, this guide will make greater use of Bluemix capabilities in ensuring that you app is deployed properly.

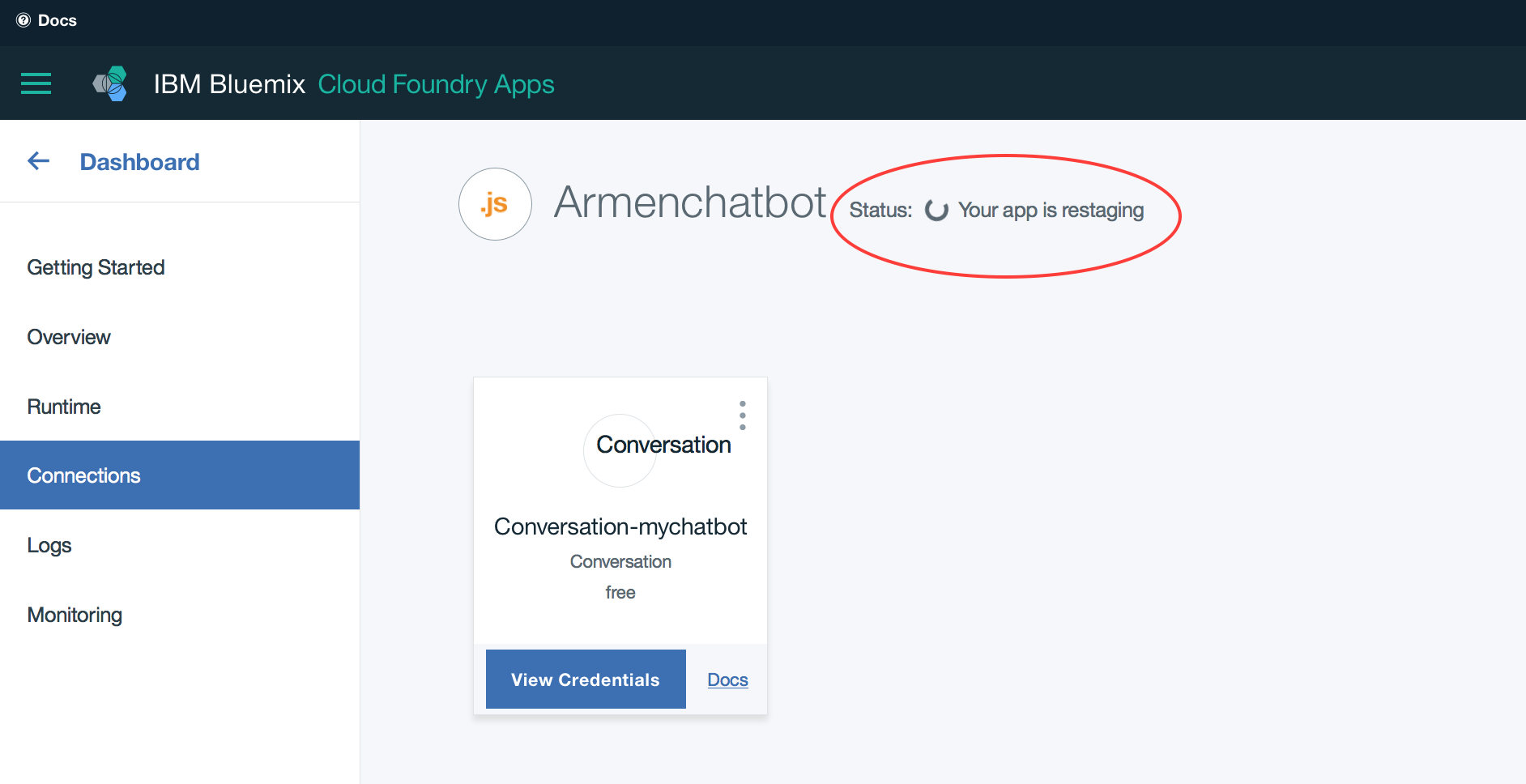
1. Back to Bluemix, and click the Dashboard view from the left contents drop down link.



1. From the Dashboard view and click **Create Application** (you may have to scroll down a bit).
2. Scroll down and from the Cloud Foundry Apps, select the **SDK for Node.js**

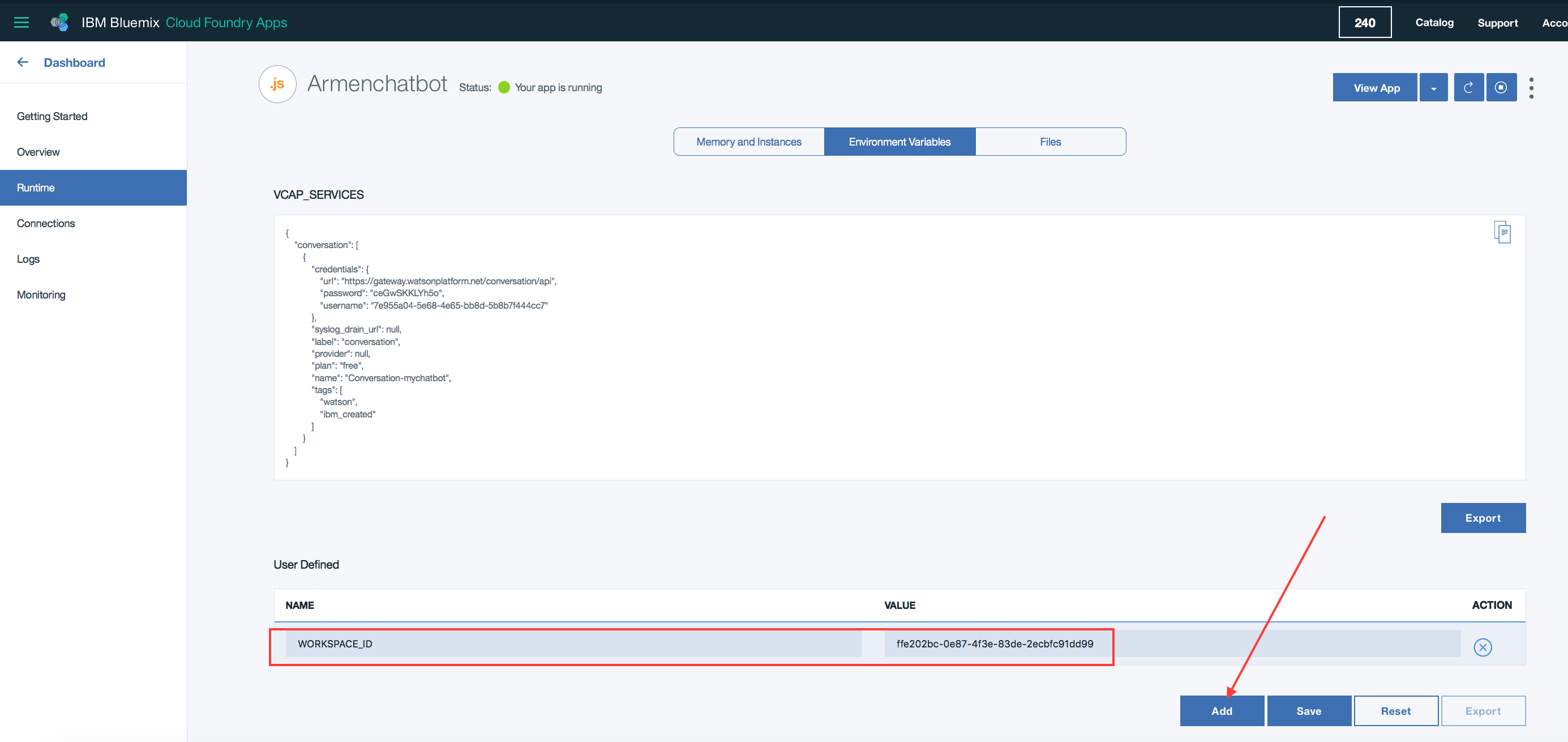


1. Specify a unique name, you may have to include your name, for example, I used Armenchatbot.
2. Click **Connections** link from the left panel.
3. Click **Existing** and find the very Conversation service that you created earlier in this lab (for example: mychatbot)
4. Click **Connect** and then click **Restage**. Allow enough time for the restaging to complete.



1. Click the **Runtime** link in the left panel.
2. Click **Environment Variables** (it takes a minute or two for the VCAP service to appear; no worries if you don’t see them right away; you are ready to include the user-defined variables, which is your WORKSPACE\_ID.

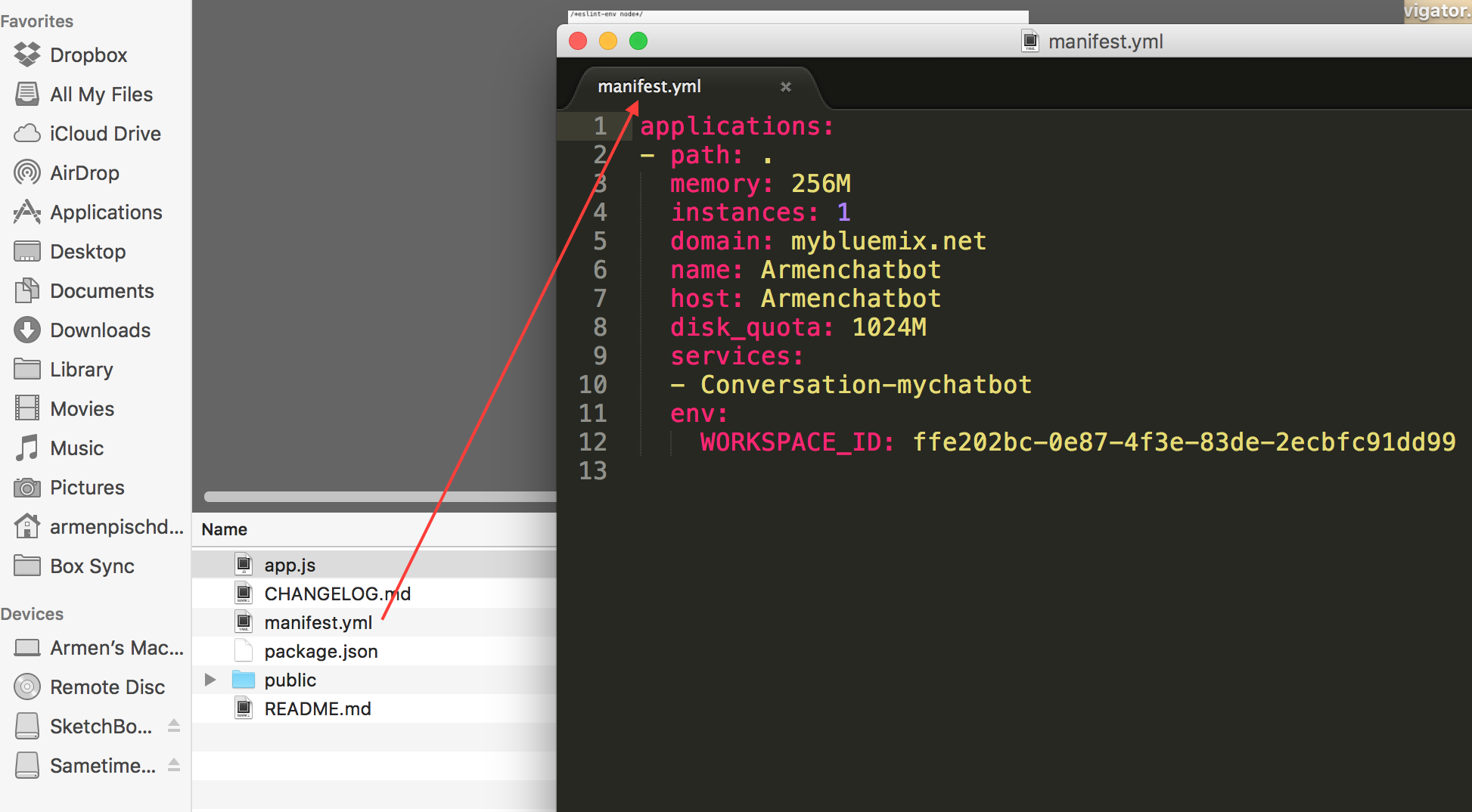
WORKSPACE\_ID 01f8ecc8-1701-4ed1-a1f9-0e76bd6b0d69  
(use your ID number from the Car\_Dashboard workspace……the details view).



1. Click **Save**.
2. Click the **Getting Started** link from the left panel.

The reason you created a dummy file is two fold: The Starter Code, which you will download in the next step contains the manifest file that you will replace with the manifest.yml file in your working directory. It has the proper URL and all the credentials that you need; the second reason, is so you can see the commands needed to upload your app (with the new manifest.yml file) to Bluemix. You can copy/paste the commands that appear in the Getting Started page, except replace bluemix in the command line, with cf. We did not use the bluemix plugin, we used the cf (Cloud Foundry) plugin.

1. Click **DOWNLOAD STARTER CODE**.
2. Extract the contents of the downloaded code in a temporary location.
3. Copy the **manifest.yml** file that has an app name and the user defined variable value, replacing the **manifest.yml** file in your working directory. Alternatively, you could have just changed the Manifest file with app name and the custom env variable, but it’s good to see where and how it’s generated.



1. Start from Step 4 onward from the **Getting Started** page on Bluemix and you perform these in the same terminal or Command prompt where the app.js and the server.js files reside (start with cf not bluemix)

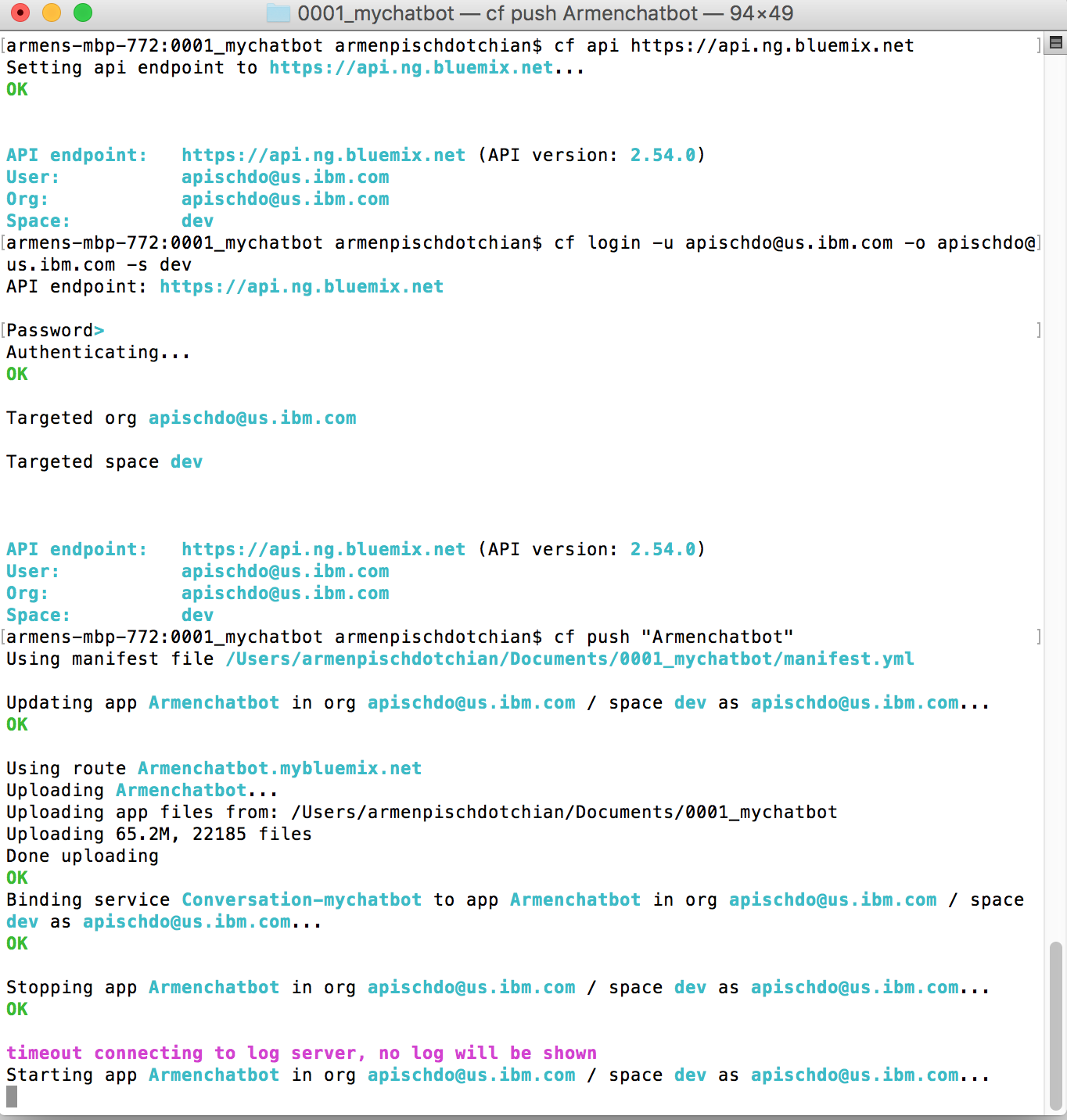
% cf api https://api.ng.bluemix.net

% cf login -u apischdo@us.ibm.com -o apischdo@us.ibm.com -s dev

% cf push "Armenchatbot"

The above is just an example of my credentials and settings. Yours would have your email address, org name and space name. When logging in, use the same Bluemix password when you created the account.

1. Allow enough time for the app to upload and restage. This takes a few minutes. Watch the console.



1. From within your application click **View App** (you may need to Restart the app again from Bluemix).
2. Accept geo-location sharing request.
3. Ask away some typical car questions and then hit it with the weather questions.

Congratulations, you just deployed a chatbot onto Bluemix.