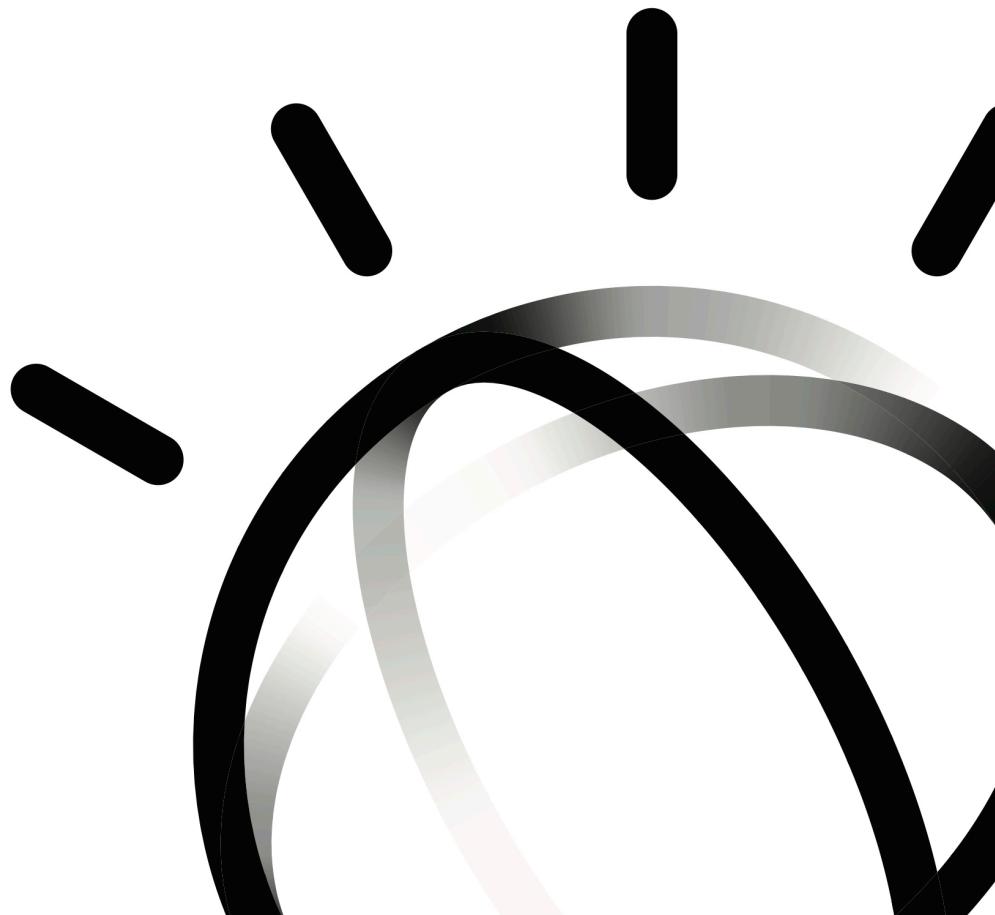


IBM Watson Solutions

Business and Academic Partners



Developing a Chatbot Using the IBM Watson Conversation Service

Prepared by Armen Pischdotchian

Version 8.0 November 2017

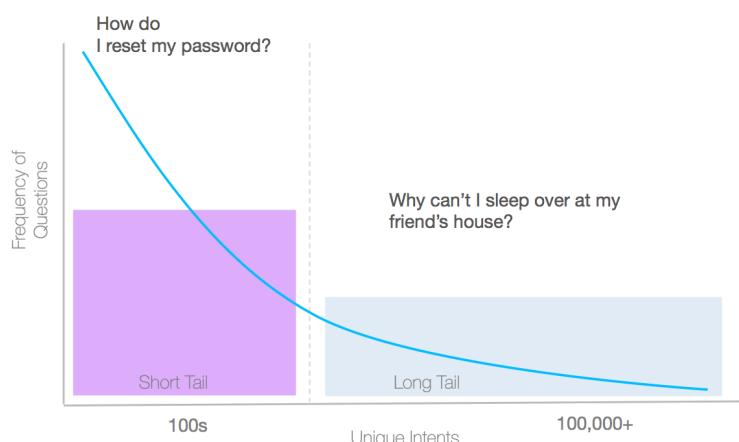
Overview

This guide is 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. The Discovery service, another API on IBM Cloud can address not so commonly asked questions, where the answer may reside in ingested documents that specifically answer esoteric questions that the system gets to learn over time. Combination of both services is a wonderful solution for a robust virtual agent.

Question Distribution



Watson Conversation
Here Watson uses reasoning strategies that focus on the language and context of the question.

Watson Discovery Service
Here Watson uses reasoning strategies that focus on identifying the most appropriate answer.

Consider the following scenario used in this guide: You are driving a cognitive enabled car. You ask in a natural way, the way you speak, for the car to do things for you, such as turn on wipers, play music and so forth. At one point, you ask the about the weather. In this workshop, you will connect the Conversation service with an external service from the Weather Company (<http://api.wunderground.com/?MR=1>) to inform you of real time actual weather in the exact location that you are sitting and performing this lab. The service obtains your location using the browser's geo-location capabilities.

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

This section provides instructions to help you get started quickly with the IBM Watson™ Developer Cloud services using Node.js as your programming runtime environment. To make it easy to get up and running with a functional application that uses the REST Application Programming Interface (API) for any Watson service, IBM provides a Node.js package with wrappers that simplify application development. The package includes simple command-line example applications to let you experiment with any of the available services.

Begin with the Pre-requisites document. At a minimum, you will need the following artifacts:

- **Obtain IBM Cloud credentials**
- **Install the Node.js runtime (select the Recommended for Most Users option)**
- **Download OS appropriate code-friendly editing tool (optional, but it is better than just Notepad)**
 - If you are using a PC, we recommend using **Notepad ++**
 - If you are using Mac, we recommend **Sublime Text**
- **Your browser must be geo-location enabled**

You must have geo-location for your browser enabled. If you had turned it off, use the following steps to enable geo-locations services:

For Mac users go to this link: <https://support.apple.com/en-us/HT204690>

For Firefox browser complete these steps:

1. Navigate to the site to that you want to enable the service
2. Go to the *Tools* menu, then select *Page Info*
3. Select the *Permissions* tab
4. Change the setting for *Share Location*

- **You will need the API key from: <http://api.wunderground.com/>**

You can register with the site and obtain your own key, or just use this key: 5aa2ca764b80f41a when the time comes later in this document.

- **Install the cf command line (optional should you decide to deploy to IBM Cloud)**
 1. Direct your browser to a GitHub repository: <https://github.com/cloudfoundry/cli/releases>
 2. Download and install the most recent installer appropriate for your operating system.
 3. You may need to open Preferences → Security and Privacy → General tab (in Mac); unlock and change the Allow applications downloaded from **Anywhere**.

Using the Conversation service

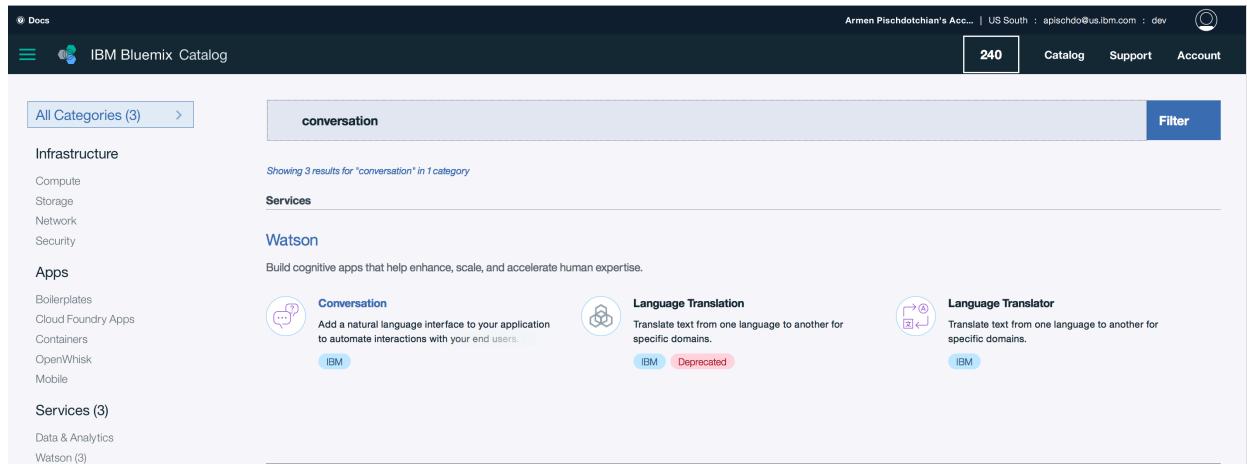
Let's begin our journey with a few good housekeeping practices:

1. Create a top-level directory on your system.
2. Download the code from the Github Repository: <https://github.com/apischdo/Bluemix-workshop-assets/blob/master/Conversation.zip>
3. Extract the contents of the **Conversation.zip**. If it auto extracted in your Downloads folder, then copy the extracted contents to your top-level directory that you created in Step 1.
4. Open a command window or a terminal and change directory (`cd`) to the working directory (in this example, I named my top directory `0001_mychatbot` (you can pick your own directory name).



```
0001_mychatbot — bash — 94x24
Last login: Sun Nov 13 19:42:50 on console
[armens-mbp-772:~ armenpischdotchian$ cd /Users/armenpischdotchian/Documents/0001_mychatbot
[armens-mbp-772:0001_mychatbot armenpischdotchian$ ls
[LICENSE           public
 README.md         readme_images
 app.js            server.js
 conversation_expedited.pdf test
 manifest.yml      training
 package.json
 armens-mbp-772:0001_mychatbot armenpischdotchians$
```

5. Press Enter and issue an `ls` (for Mac) or `dir` (for PC) command to ensure that all the artifacts that you extracted are visible. If you see the `app.js`, the main engine that does the interpreting, then you are working from the correct folder.
 6. From the same terminal, run the node package manager (`npm`) to install dependencies that `node.js` relies on for further processing of the code. You might get some warning and one compile error message, ignore those; it has to do with version incompatibility. Notice that you have a new folder named `node_modules`.
- ```
npm install
```
7. Login into IBM Cloud: <https://console.bluemix.net/catalog/>
  8. Click the **Catalog** tab.
  9. Search for the **Conversation** service and click that tile.



The screenshot shows the IBM Bluemix Catalog interface. At the top, there is a navigation bar with links for 'Docs', 'IBM Bluemix Catalog', '240', 'Catalog', 'Support', and 'Account'. A user profile icon is also present. In the center, there is a search bar with the text 'conversation' and a 'Filter' button. Below the search bar, a message says 'Showing 3 results for "conversation" in 1 category'. There are three service tiles displayed: 'Conversation', 'Language Translation', and 'Language Translator'. The 'Conversation' tile is selected and described as 'Add a natural language interface to your application to automate interactions with your end users.' It is provided by 'IBM'. The 'Language Translation' tile is described as 'Translate text from one language to another for specific domains.' It is provided by 'IBM' and has a 'Deprecated' status. The 'Language Translator' tile is described as 'Translate text from one language to another for specific domains.' It is provided by 'IBM'.

10. 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 IBM Cloud once again, then you can see the Create button in the bottom right corner).

Watson

Conversation-mychatbot

Manage    Service Credentials    Connections

## Conversation

Add a natural language interface to your application to automate interactions with your end users. Common applications include virtual agents and chat bots that can integrate and communicate on any channel or device.

Conversation tooling

Launch tool

Developer resources:

- Documentation
- Demo

11. Click the **Launch Tool** button. It may appear as an arrow in your interface
12. Login once again using the same credentials that you use to login IBM Cloud.
13. Click **Import**.
14. Navigate to the **training/car\_workspace.json** file and click **Open** to import the file.

Watson Conversation

Instances: Conversation-mychatbot

Import a workspace

Select a JSON file then choose which elements from the workspace to import.

Choose a file

Import

Everything (Intents, Entities, Entities, Entities)

Intents and Entities

Favorites

Dropbox

iCloud Drive

Applications

Desktop

Documents

Downloads

Library

Movies

Music

Pictures

armenpischdotchian

| Name                       | Date M |
|----------------------------|--------|
| 0001_mychatbot             | 3:18 P |
| node_modules               | 3:14 P |
| public                     | 2:41 P |
| readme_images              | 2:41 P |
| training                   | 2:41 P |
| sample_manifest.yml        | 10/6/1 |
| weather.js                 | 7/22/1 |
| car_workspace.json         | 7/19/1 |
| conversation_expedited.pdf | 10/6/1 |
| app.js                     | 10/6/1 |
| manifest.yml               | 7/19/1 |
| LICENSE                    | 7/19/1 |
| package.json               | 7/19/1 |
| README.md                  | 7/19/1 |

Format: \*.json

Options

Cancel    Open

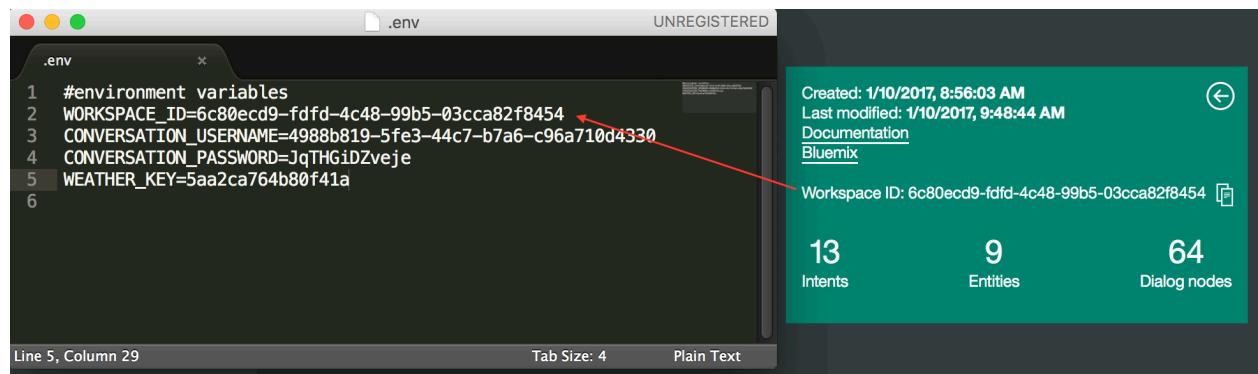
You must now provide four parameters to the code: conversation ID and the service credentials (username and password). Bear in mind that files that start with a dot “.” are hidden system files, but with Sublime or Notepad++, by opening the folder, instead of just a file, you can see those hidden files. And if you just can't see internal files, then no worries just save the file and trust it is there.

15. In this example, select the top-level directory (0001\_mychatbot) and open it with Sublime.
16. Open a new file with sublime and copy paste the below in that file; you may have to enter the line breaks after each equal sign so it appears as below in four separate lines:

```
#environment variables
WORKSPACE_ID=
CONVERSATION_USERNAME=
CONVERSATION_PASSWORD=
WEATHER_KEY=5aa2ca764b80f41a
```

You are now ready to populate the environment variables:

17. Obtain the workspace ID by clicking the three dots in Workspace box and then click **View Details**.



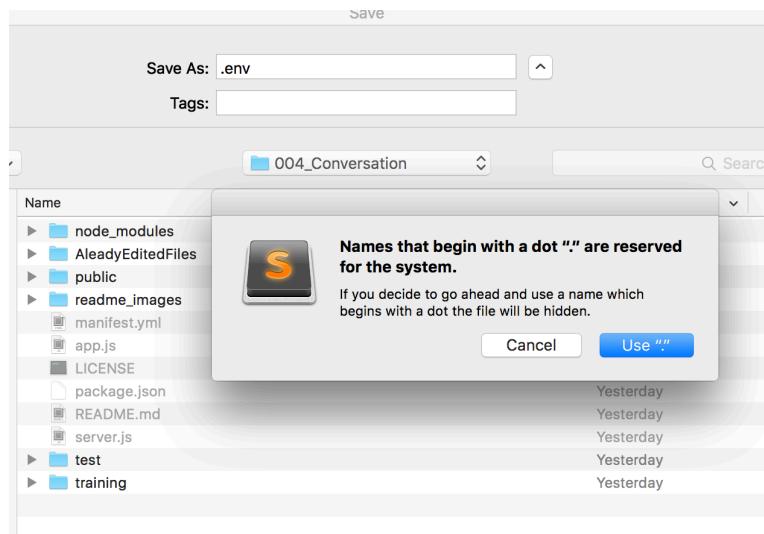
18. Go to the Conversation service on IBM Cloud, and obtain the username and password of the Conversation service from the **Service Credential** link and then click **View Credentials**.

The screenshot shows the Watson Conversation service in the IBM Cloud dashboard. It displays a service credential with the following JSON content:

```
{
 "url": "https://gateway.watsonplatform.net/conversation/api",
 "password": "UL7DCJUQ2Eh2",
 "username": "a502832b-09ed-40e7-9352-9ddb5f098d88"
}
```

19. The Weather API key is: 5aa2ca764b80f41a
20. Enter all four parameters in a new file in Sublime or Notepad ++.

21. Save the file as **.env** (notice, there is a dot in front of the file name, this is a system internal file, typically hidden) and click **Use “.”** Ensure that you save this file in the top-working directory where all other files, same level as app.js reside.



If you are using a PC, for file type ensure to select **All types**. You don't want to save it as a txt file.

22. Go back to the terminal window and run the following command:

```
Node server.js
```

## Binding the Conversation service with an external API

For this next section you are going to use the geo-location capabilities of your browser to check the actual weather forecast and for that you will need to include the API key from the Weather Company web site that you obtained from the Pre-requisites section.

1. From your top-level directory navigate to the **training** folder and open the **weather.js** file and position it to the left of your screen.
2. Navigate to the **public/js/conversation.js** folder and open the **conversation.js** file and position it to the right of your screen; just like the image below, you want them side-by-side.

```
weather.js
```

```
1
2
3 //Client JS Code.. Add this to conversation.js, overwriting the init function
4 //new init function
5 // Initialize the module
6 function init() {
7 chatUpdateSetup();
8 if(navigator.geolocation){
9 navigator.geolocation.getCurrentPosition(geoSuccess, geoError);
10 }
11 else{
12 console.log("Browser geolocation isn't supported.");
13 geoSuccess(position)
14 }
15 setupInputBox();
16}
17
18//private functions
19function geoSuccess(position){
20 var context = null;
21 if(position && position.coords){
22 context = {};
23 context.long = position.coords.longitude;
24 context.lat = position.coords.latitude;
25 }
26 // The client displays the initial message to the end user
27 Api.sendRequest("", context);
28};
29
30//Sends in null to ask for zip code
31function geoError(){
32 geoSuccess(null);
33};
```

```
conversation.js
```

```
21
22
23 return {
24 init: init,
25 inputKeyDown: inputKeyDown
26 };
27
28 // Initialize the module
29 function init() {
30 chatUpdateSetup();
31 Api.sendRequest("", null);
32 setupInputBox();
33
34 // Set up callbacks on payload setters in Api module
35 // In this case the displayMessage Function to be called when messages are sent
36 function chatUpdateSetup() {
37 var currentRequestPayloadSetter = Api.setRequestPayload;
38 Api.setRequestPayload = function(newPayloadStr) {
39 currentRequestPayloadSetter.call(Api, newPayloadStr);
40 displayMessage(JSON.parse(newPayloadStr), settings.authorTypes.user);
41 };
42
43 var currentResponsePayloadSetter = Api.setResponsePayload;
44 Api.setResponsePayload = function(newPayloadStr) {
45 currentResponsePayloadSetter.call(Api, newPayloadStr);
46 displayMessage(JSON.parse(newPayloadStr), settings.authorTypes.watson);
47 };
48
49 function setupInputBox() {
50 var input = document.getElementById('textInput');
51 var dummy = document.getElementById('textInputDummy');
52 var padding = 3;
53
54 if (dummy == null) {
55 var dummyJson = {
```

3. Copy the `init()` function, `geoSuccess()` and `getError()` functions (lines 6 through 33) from the `weather.js` file *replacing* the `init()` function in the `conversation.js` file, *over writing* lines 27 through 31.
  4. Save the `conversation.js` file and close it.
  5. Run the `node server.js` command again and if need be, (Ctrl+C) to end the current terminal session.
  6. Back to your localhost:3000 tab and refresh the browser.
  7. Ask the following questions: **Turn on my wipers** and **What is the weather like today?**

Hi. It looks like a nice drive today. What would you like me to do?

turn on the wipers

Ok. Turning on the wipers.

what is the weather like today

Unfortunately I don't know much about the weather..I'm still learning.

User input

```
1 {
2 "input": {
3 "text": "what is the weather like today"
4 },
5 "context": {
6 "long": -71.4708425,
7 "lat": 42.549521999999996,
8 "conversation_id": "d56af184-882f-4b18-967a-069bc116084e",
9 "system": {
10 "dialog_stack": [
11 "root"
12],
13 "dialog_turn_counter": 2,
14 "dialog_request_counter": 2
15 },
16 "defaultCounter": 0,
17 "reprompt": true
18 }
19 }
```

Watson understands

```
1 {
2 "input": {
3 "text": "what is the weather like today"
4 },
5 "context": {
6 "long": -71.4708425,
7 "lat": 42.549521999999996,
8 "conversation_id": "d56af184-882f-4b18-967a-069bc116084e"
```

Type something

Notice that it understands the concept of weather, but the app cannot fetch that data at this point.

You are now ready to incorporate these changes in the dialog structure.

# 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. Turn on the **@sys-date** system entity.

The screenshot shows the Entity tab in the Watson Assistant workspace. The tab bar includes 'Intents', 'Entities' (selected), and 'Dialog'. Below the tabs, there are two sections: 'My entities' and 'System entities'. Under 'System entities', two entities are listed: '@sys-time' (disabled) and '@sys-date' (enabled). A note at the top states: 'These are common entities created by IBM that could be used across any use case. They are ready to use as soon as you add them to your workspace. \*System entities cannot be edited. Learn more'.

3. Click the **Dialog** tab.
  - a. Select and click the **Weather** Dialog to open it for editing.
  - b. Delete all three child nodes to the right of it.
  - c. Make sure the condition is “#weather” and the response field is blank.
  - d. Add two child nodes of **#weather**, named “Weather date specified” and “Weather date not specified”.
  - e. For “Weather date specified” include a condition of “@sys-date” and a response of

`The weather for <?@sys-date.reformatDateTime('EEEEEEEEE')?> in {0} is {1}`

- i. Ensure that the single quotes around EEEEEEEE are straight, san serif, when you enter it in the node, not curved as it appears in this rich text format below. Type over it in the code.

- f. For “Weather date not specified” include a condition of “!@sys-date” and a response of

`For which day would you like to know the weather?`

- g. Return to the **#weather** node and change the *And finally* to be a Jump to “Weather date specified”.

- h. In the Triggered by section, add an or function (change the and to or) and specify it as: **@sys-date**

- i. Add two new responses: the true statement: **@sys-date**, which contains the following date format

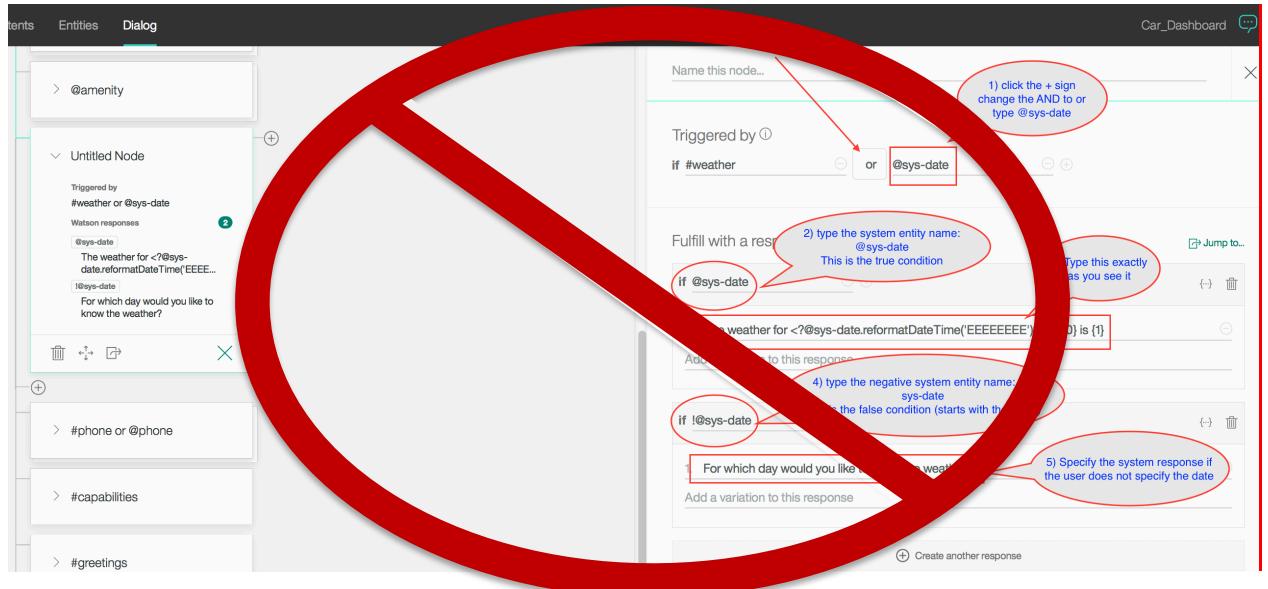
- j. Ensure that the single quotes around EEEEEEEE are straight, san serif, when you enter it in the node, not curved as it appears in this rich text format below. Type over it in the code.

`The weather for <?@sys-date.reformatDateTime('EEEEEEEEE')?> in {0} is {1}`

- k. Add the false statement: **!sys-date**...notice the ! sign up front...that means NOT.

1. Include a system response for users who do not indicate for when they want to know the weather:

~~For which day would you like to know the weather?~~



Now let's test the conversation

4. Click **Try it out** in the upper right corner.
5. Ask: **How is the weather?**

6. The system asks you to specify a date. Type a day, for example **Friday**.

Notice it understood the intent and returned: The weather for Friday in {0} is {1}

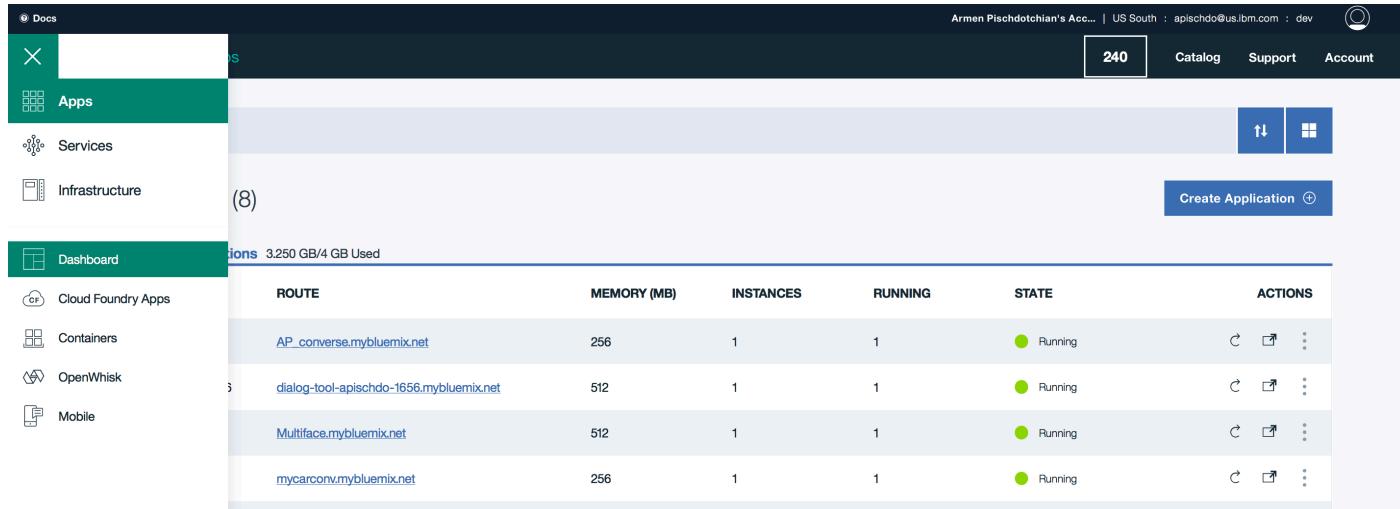
7. Try it in **<http://localhost:3000>**
8. Ask the question: "How is the weather today?" or just "Friday". Notice the accurate weather forecast at your location.

If time permits, the following exercises are optional.

# Deploying your app to IBM Cloud

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 IBM Cloud. However, this guide will make greater use of IBM Cloud capabilities in ensuring that your app is deployed properly.

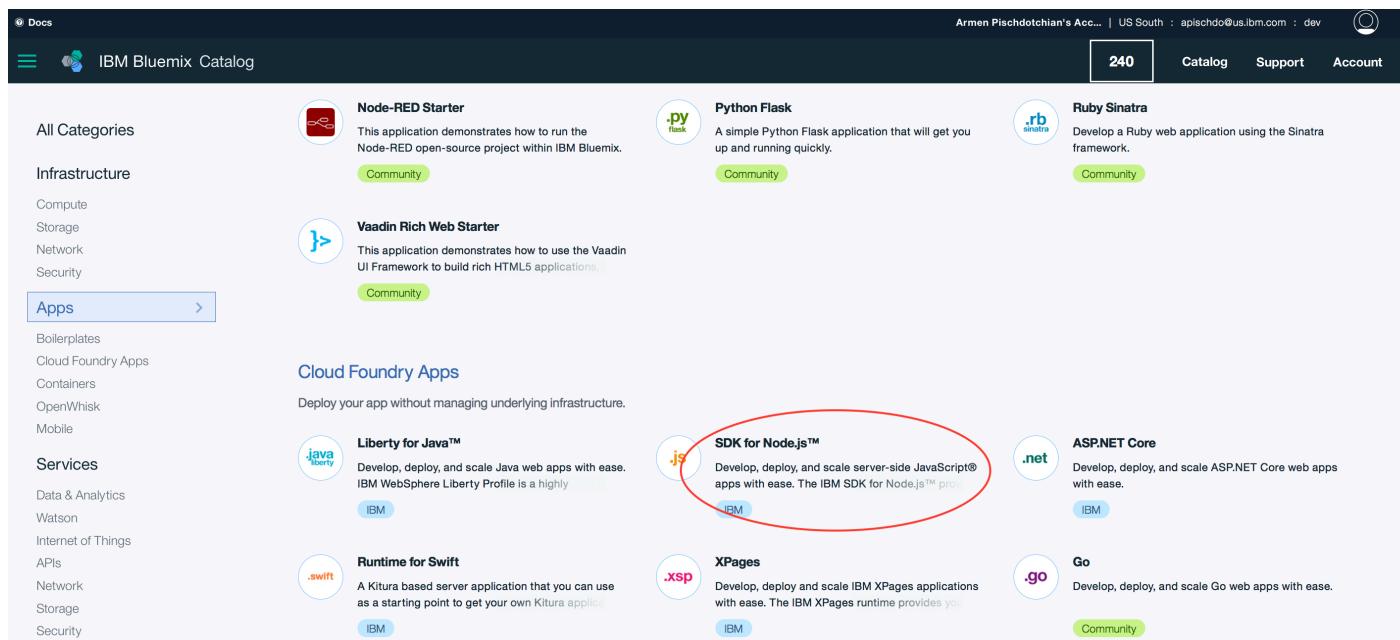
1. You will need to install the CLI command line from here: <https://github.com/cloudfoundry/cli/releases>
2. Back to IBM Cloud, and click the **Dashboard** view from the left contents drop down link.



The screenshot shows the IBM Cloud dashboard with the following details:

- Cloud Foundry Apps:** 3.250 GB / 4 GB Used
- ROUTE:** AP\_converse.mybluemix.net, dialog-tool-apischdo-1656.mybluemix.net, Multiface.mybluemix.net, mycarconv.mybluemix.net
- MEMORY (MB):** 256, 512, 512, 256
- INSTANCES:** 1, 1, 1, 1
- RUNNING:** All four applications are listed as "Running" with green status dots.
- STATE:** All four applications are listed as "Running".
- ACTIONS:** Each application has a "Logs" button, a "View" button, and a "More" button.

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



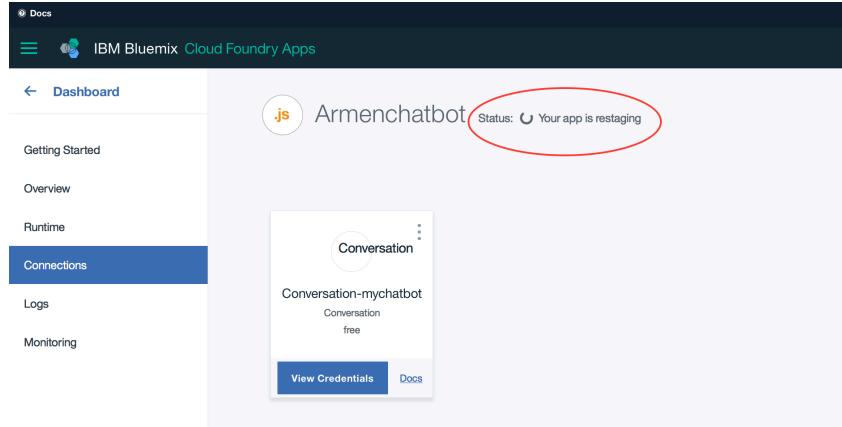
The screenshot shows the IBM Bluemix Catalog with the following details:

- Cloud Foundry Apps:** Deploy your app without managing underlying infrastructure.
- SDK for Node.js™:** Develop, deploy, and scale server-side JavaScript® apps with ease. The IBM SDK for Node.js™ provides a simple interface for building web applications using Node.js.
- Other Applications:** Node-RED Starter, Python Flask, Ruby Sinatra, Vaadin Rich Web Starter, ASP.NET Core, Go, Runtime for Swift, XPages.

5. Specify a unique name, you may have to include your name, for example, I used Armenchatbot.
6. Click **Connections** link from the left panel.

7. Click **Existing** and find the very Conversation service that you created earlier in this lab (for example: mychatbot)

8. Click **Connect** and then click **Restage**. Allow enough time for the restaging to complete.

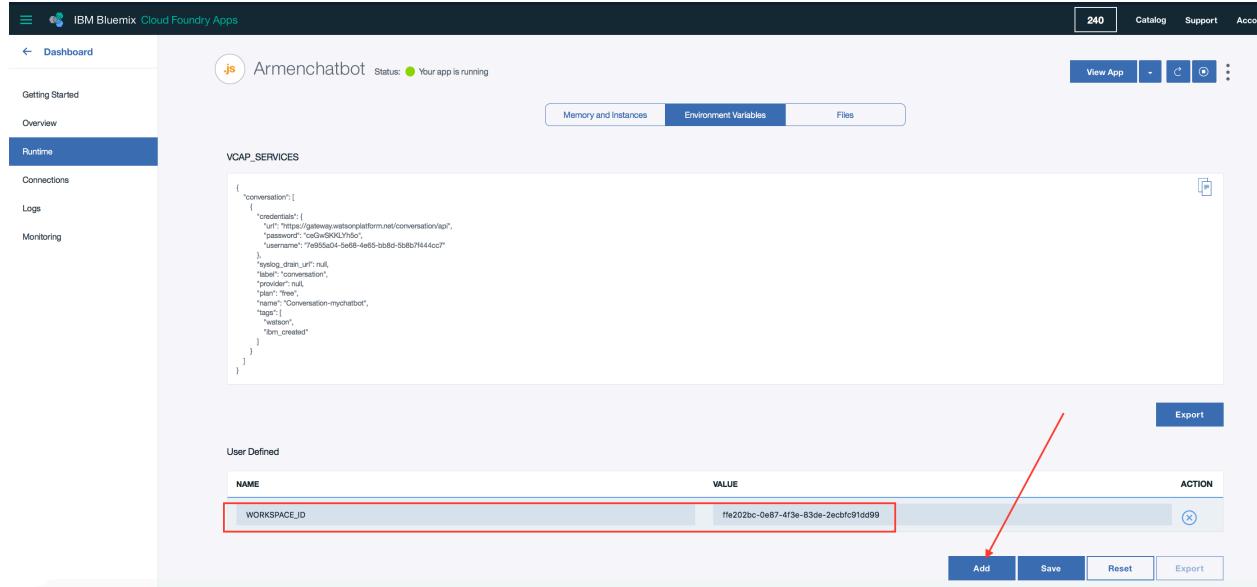


The screenshot shows the IBM Bluemix Cloud Foundry Apps interface. On the left, there's a sidebar with links: 'Docs', 'Dashboard', 'Getting Started', 'Overview', 'Runtime' (which is highlighted in blue), 'Connections', 'Logs', and 'Monitoring'. The main area displays an application named 'Armenchatbot'. A status message 'Status: Your app is restaging' is shown above the application card, which includes a 'Conversation' icon and the text 'Conversation-mychatbot', 'Conversation', and 'free'. Below the card are buttons for 'View Credentials' and 'Docs'.

9. Click the **Runtime** link in the left panel.

10. 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).



The screenshot shows the 'Runtime' environment variables page. The 'VCAP\_SERVICES' tab is selected, displaying JSON data for the Conversation service. The data shows a single entry for 'conversation' with various fields like 'url', 'password', and 'name'. Below this, the 'User Defined' section contains a table with one row for 'WORKSPACE\_ID'. The 'NAME' column has 'WORKSPACE\_ID' and the 'VALUE' column has 'ffe202bc-0e87-4f3e-83de-2ecbfc91de99'. At the bottom of the table, there are buttons for 'Add', 'Save', 'Reset', and 'Export'. A red arrow points to the 'Save' button.

11. Repeat the previous step and add the weather variables:

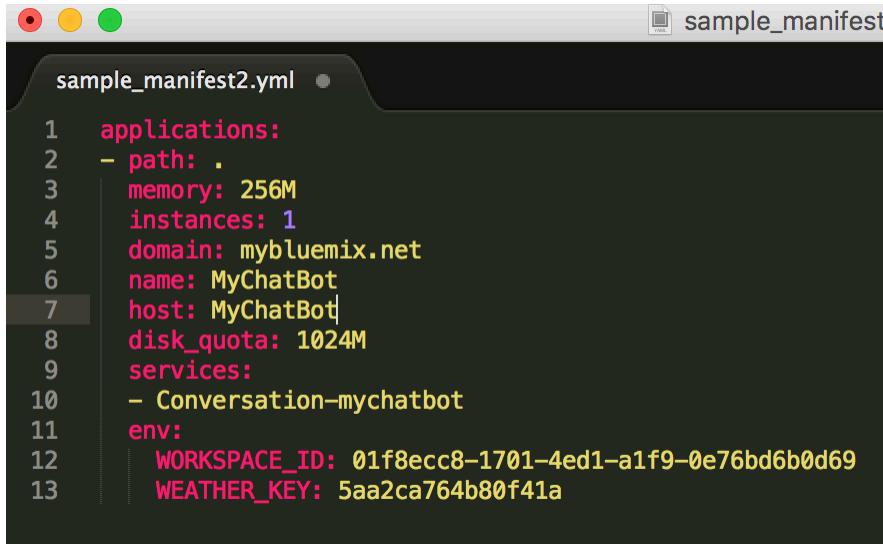
WEATHER\_KEY                    5aa2ca764b80f41a

12. Click **Save**.

13. 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 IBM Cloud. 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.

14. Click **DOWNLOAD STARTER CODE**.
15. Extract the contents of the downloaded code in a temporary location.
16. 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.



```
sample_manifest2.yml
1 applications:
2 - path: .
3 memory: 256M
4 instances: 1
5 domain: mybluemix.net
6 name: MyChatBot
7 host: MyChatBot
8 disk_quota: 1024M
9 services:
10 - Conversation-mychatbot
11 env:
12 WORKSPACE_ID: 01f8ecc8-1701-4ed1-a1f9-0e76bd6b0d69
13 WEATHER_KEY: 5aa2ca764b80f41a
```

17. Start from Step 4 onward from the **Getting Started** page on IBM Cloud 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).  
The command below is one line. You org and space name may differ.

```
% cf login -u apischdo@us.ibm.com -o apischdo@us.ibm.com -s dev -sso API
endpoint: https://api.ng.bluemix.net
```

18. Copy and paste the resulting URL in your browser to obtain the One Time Code  
[<https://login.ng.bluemix.net/UAALoginServerWAR/passcode>](https://login.ng.bluemix.net/UAALoginServerWAR/passcode)
19. Enter the code from the URL back into the command prompt or terminal as the password.
20. Push the app to IBM Cloud:

```
%cf push <your_app_name_as_it_appears_in_the_manifest_file>
```

The above is just an example of my credentials and settings. Yours would have your email address, org name and space name.

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

```
[armens-mbp-772:0001_mychatbot armenpischdotchian$ cf api https://api.ng.bluemix.net
Setting api endpoint to https://api.ng.bluemix.net...
OK

API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: apischdo@us.ibm.com
Org: apischdo@us.ibm.com
Space: dev
[armens-mbp-772:0001_mychatbot armenpischdotchian$ cf login -u apischdo@us.ibm.com -o apischdo@us.ibm.com -s dev
API endpoint: https://api.ng.bluemix.net

>Password>
Authenticating...
OK

Targeted org apischdo@us.ibm.com
Targeted space dev

API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: apischdo@us.ibm.com
Org: apischdo@us.ibm.com
Space: dev
[armens-mbp-772:0001_mychatbot armenpischdotchian$ cf push "Armenchatbot"
Using manifest file /Users/armenpischdotchian/Documents/0001_mychatbot/manifest.yml

Updating app Armenchatbot in org apischdo@us.ibm.com / space dev as apischdo@us.ibm.com...
OK

Using route Armenchatbot.mybluemix.net
Uploading Armenchatbot...
Uploading app files from: /Users/armenpischdotchian/Documents/0001_mychatbot
Uploading 65.2M, 22185 files
Done uploading
OK
Binding service Conversation-mychatbot to app Armenchatbot in org apischdo@us.ibm.com / space dev as apischdo@us.ibm.com...
OK

Stopping app Armenchatbot in org apischdo@us.ibm.com / space dev as apischdo@us.ibm.com...
OK

timeout connecting to log server, no log will be shown
Starting app Armenchatbot in org apischdo@us.ibm.com / space dev as apischdo@us.ibm.com...]
```

22. From within your application click **View App** (you may need to Restart the app again from IBM Cloud).

23. Accept geo-location sharing request.

24. Ask away some typical car questions and then hit it with the weather questions.

Congratulations, you just deployed a chatbot onto IBM Cloud.