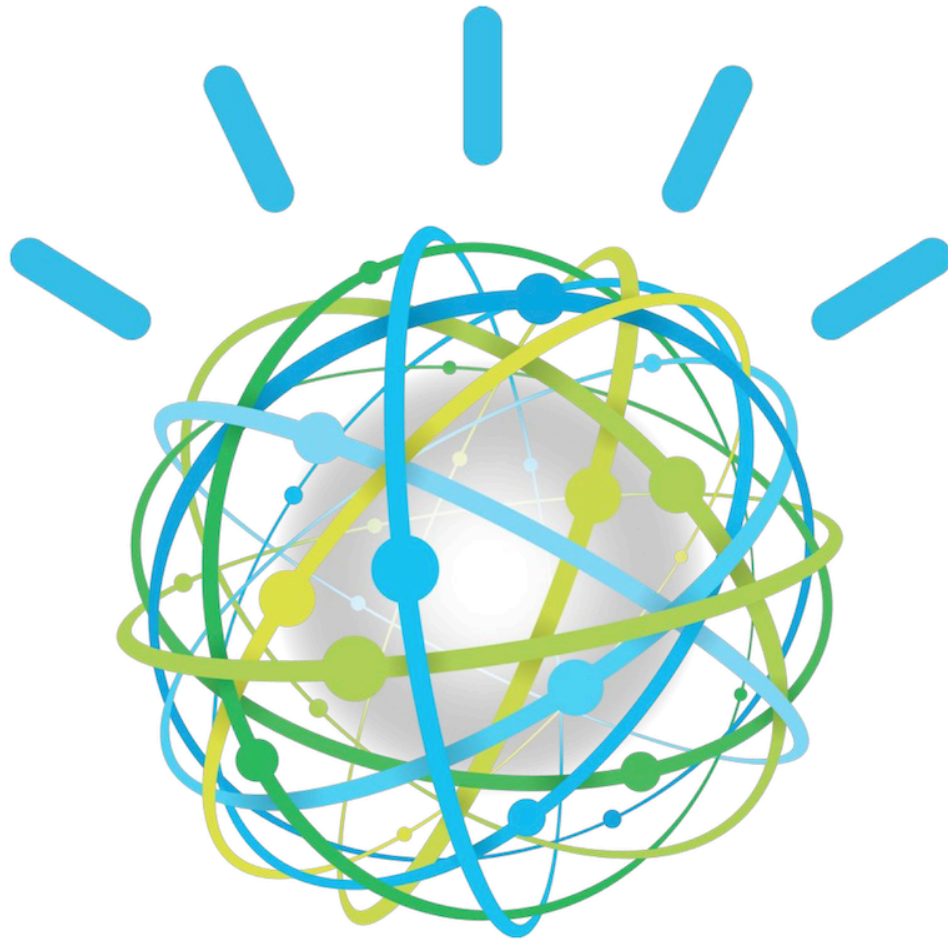


# **IBM Watson Solutions**



## **Building a Helpdesk App Using Dialog and NLC Bluemix Services**

**Prepared by Armen Pischdotchian**

**Version 6.0 May 2016**

## Overview

What is Bluemix you ask? [Bluemix](http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html) 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>

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 applications and services to publish a basic and hypothetical solution using Dialog and Natural Language Classifier (referred to as NLC from here on), where you can then expand on what you have learned and build complex solutions for your specific use cases.

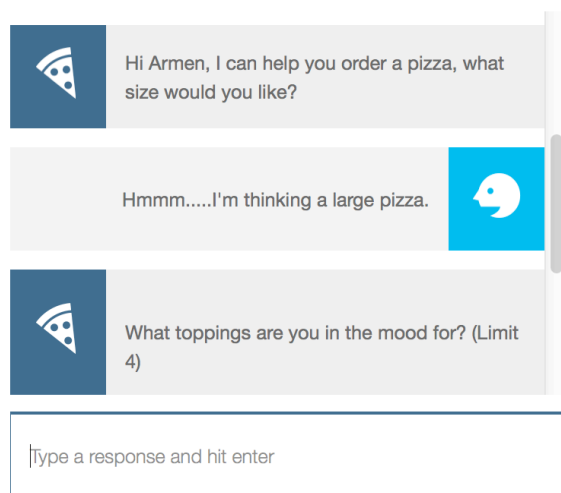
Consider the following scenario used in this guide: You are using a pizza-buying app to order your favorite pizza. At some point Watson asks about the weather: if it is a nice and sunny day, you would opt for pickup, if it is inclement weather, then you will opt for delivery. Instead of exhausting a long traverse through the Dialog tree structure, a certain question by Watson such as “How is the weather?” calls the NLC service, where an intent or class of “bad” weather will prompt Watson to suggest delivery without you asking for it; whereas a “good” weather will prompt Watson to suggest that you may want to opt for pizza pickup (note that “good” and “bad” are intents that you specify in your CSV file – what NLC uses as ground truth -- against certain weather related questions or utterances).

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, 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.

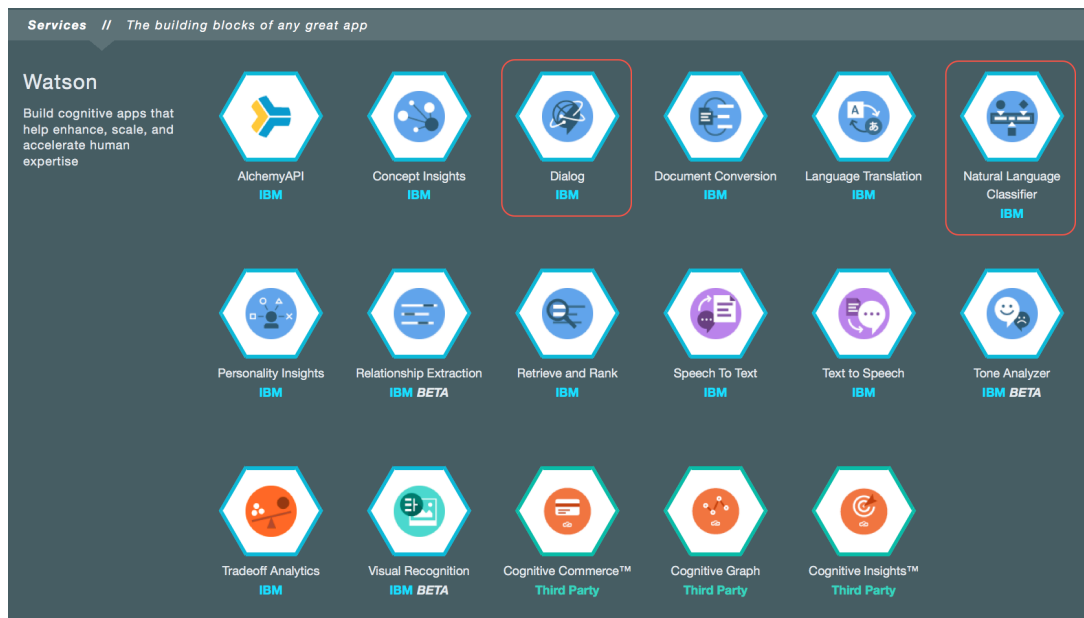
## About your application

The end result of your application will look like the example screen capture below:

### Try the service



In this workshop you will use two services from Bluemix: **Dialog** and **Natural Language Classifier**. At first, let's become familiar with each service separately. After which, you will download the modified code from GitHub to accommodate a new conversation and establish the hooks between the two services.



## 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. Complete the following steps to satisfy the prerequisites

- **Obtain Bluemix credentials:**

As of this writing, if you are a university faculty, you can join the IBM Academic Initiative program. The Academic Initiative for Cloud offer extends Bluemix trial access for faculty for 12 months (renewable) and for students who enroll in approved faculty courses for 6 months (renewable) with no need for credit card.

1. Direct your browser to the Bluemix home page: <https://console.ng.bluemix.net/home/>
2. Click **Sign Up** on the top right (for universities, use an email that ends with edu (your school email)).
3. Enter requested information and click **Create Account**.

- **Install the Node.js runtime:**

The default installation includes both the runtime and package manager. Make sure to include the installed binaries on your PATH environment variable after installation (typically, the default installation locations that the installer selects does the inclusion).

1. Direct your browser to the nodejs.org web site: <https://nodejs.org/download/>
2. Click **Downloads**.
3. Select and install the installer (not the binary) appropriate for your operating system.

- **Install the cf command line**

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**.

## 2. **Download cURL**

You will need to interface with cURL commands to upload documents for NLC for this exercise pending the release of tooling surround NLC and other services where you perform training sets.

1. Before installing it, run the following command: `curl -V` from a terminal or command prompt. If you get the version number then cURL is installed by default, otherwise install it using the following steps (typically, Mac OS X platforms have it installed by default).
2. To install cURL, follow this link: <https://curl.haxx.se/dlwiz/?type=bin>
3. Make sure to select the SSL-enabled version of cURL.

- **Download OS appropriate code-friendly editing tool:**

- If you are using a PC, we recommend using **Notepad ++**
- If you are using Mac, we recommend **Sublime Text**

- **Deploy to the Dialog Tool to Bluemix**

This tool helps you create, manage, and interact with dialogs for the IBM Watson Dialog Service. You merely upload your XML file to the tool to see how the conversation will traverse, you can then make edits in the XML file and upload it again and view your changes to the conversation in the tool. Direct your browser to the following link:

<https://github.com/watson-developer-cloud/dialog-tool.git>

There is no need to download it, merely click the **Deploy to Bluemix** button in the Dialog Tool section.

- **Download the workshop code and documentation from Github**

This package contains custom workshop code that you will merge with the pizza app to include the NLC call from Dialog (you will also see other labs included in the repository, feel free to download all).

1. Direct your browser to Github web site <https://github.com/> (no need to sign up)
2. Search for **bluemix-workshop** in the search box at the top of the web page
3. Select: `apischdo/Bluemix-workshop-assets`
4. Download **BuildingHelpdesk\_Dilaog-NLC.zip** file by clicking the file name and then click View Raw and save the file to your Downloads area (or where ever the default location is set in your browser).
5. Extract the contents of the **BuildingHelpdesk\_Dilaog-NLC.zip** file in the same location.

You will be merging the workshop code with the Dialog demo code.

## Becoming familiar with Dialog and NLC service

Before you delve into the details of each service, let's take a few minutes and run the demos provided for each (and all other) services on Bluemix. Complete the following steps:

1. Sign into **Bluemix** with the same credentials you used to open an account.
2. Click the **CATALOG** link at the top
3. Click **VIEW DOCS** in the left panel and a new tab opens in your browser
4. Scroll down a bit and click **View Demo** in the right panel
5. Have a conversation with the virtual agent; for example, ask for a large pizza, choose a topping such as olives, or ask what toppings they have and select either delivery or pickup.

This is a rules based app, there is no machine learning involved here and you hard code the input and output to your conversations. You can use wild cards for more flexibility in what you type and how the app will respond and at the moment you edit the Dialog app in an XML file as the case is in this workshop. Later releases will include a UI for you to better construct the dialog without having to edit an XML file.

You are now ready to play with the NLC service.

6. Refer back to the **Bluemix** tab in your browser.
7. Click the **CATALOG** link in the top pane
8. Select and click the **Natural Language Classifier** service.

This is a cognitive service and employs machine learning techniques, more specifically, unsupervised hyperbolic tangent gradient descent algorithms for creating biases per each node in it's hidden node layers (deep learning). In this workshop you will be "hooking" this service with the Dialog service to develop a cognitive solution. For more details on artificial neural networks consult the author of this document.

9. Click **VIEW DOCS**.
10. In the ensuing new tab, tryout the demo questions or questions of your own regarding the weather. The classifier is trained on two distinct intents: temperature and condition.

You are now ready to begin the exercises in this workshop.

## Summary of activities in this workshop

The following table summarizes the 9 distinct activities that you will complete in this workshop.

Step	File name	Actions that you perform
1	Download and merge the workshop code with the pizza app code	You begin by downloading two sets of code, one from WDC Github repository and another from the instructor's Github location.
2	Edit the <b>manifest.yml</b> file	The Manifest houses the context of your URL for example: <b>http://mycompany.mybluemix.net</b> and the name of the services that you plan to use.
3	Push app to Bluemix	You perform this task on your local machine
4	Add the Dialog and the NLC service to your app on Bluemix	You perform this task in Bluemix
5	In the Dialog folder edit the sample dialog XML file named <b>pizza_sample2.xml</b> .	This is a rules based (nothing cognitive here) chitchat interaction that you will have with Watson.
6	Edit the ground truth file named <b>weather_training_data.csv</b> file residing in the folder named <b>nlcGT</b>	This folder contains the csv file that is the ground truth (the supervised learning part) of NLC. You train NLC to build patterns based on the utterance and intents expressed here.
7	Edit the interpreter engine: <b>app.js</b>	You edit this file to include basic auth parameters (URL, username and password) from Bluemix, known as VCAP_SERVICE plus the instantiated ID session for both Dialog and NLC. Every time you create a new dialog or develop a new ground truth, you will need a new session ID.
8	Run the app locally	Before you push the app to Bluemix, let's ensure that it runs as intended on your local system
9	Push the app to Bluemix	At this stage, you are ready to deploy your app to Bluemix and access it from any device, mobile or otherwise via a browser.

## Merge the workshop code from WDC with the pizza app code

You begin by merging the *workshop* code from Github repository **/Bluemix-workshop-assets** with the *pizza demo* app that you downloaded from **/dialog-nodejs**.

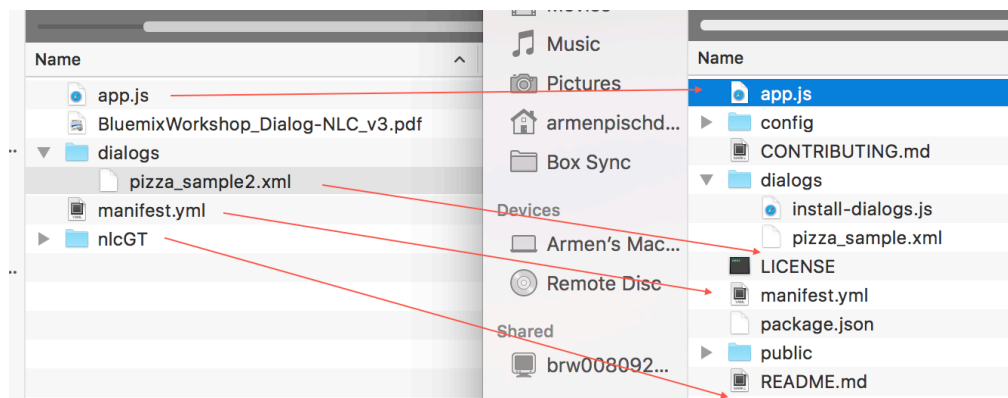
1. Create a new top-level directory on your system. The reason that you want this to be a top level directory is because you will need to access to the directory path frequently in this exercise and it is easier than having to traverse to a deeply imbedded directory.
2. Download the demo pizza app code that uses only the Dialog service from the Watson Developer Cloud (WDC): <https://github.com/watson-developer-cloud/dialog-nodejs.git>

Notice that a new folder named **dialog-nodejs-master.zip** is created for you in the **Downloads** folder.

3. Extract the contents of the **dialog-nodejs-master.zip** file.
4. Copy the contents of the **dialog-nodejs-master** folder to your newly created top-level working directory.

### Download the workshop code

5. Access the workshop code from Github repository:  
**<https://github.com/apischdo/Bluemix-workshop-assets>**
6. Click the BuildingHelpdesk\_Dialog-NLC.zip artifact from the list of various workshops from the repository.
7. Click **View Raw** and save the zip file to your Downloads folder. You can also download all that you see from the repository one level up and have all the workshops, but it's the Dialog-NLC that you need.
8. Extract the zip file (don't just click it and view the contents, but an actual extract).
9. Open *two* Windows Explorer or *two* Finder windows side by side. To the left is the workshop code that you extracted in the Downloads folder and to the right is the pizza dialog app that you extracted in the top-level in your working directory.
10. Copy the following artifacts from the *workshop code* from the Downloads location to your *working directory*:
  - a. The **app.js** (replacing the existing app.js)
  - b. The **pizza\_sample2.xml** (goes inside the Dialog folder in your working directory)
  - c. The **manifest.yml** (replacing the existing manifest.yml in your working directory)
  - d. The **nlgGT** folder (from the downloads section to your working directory)



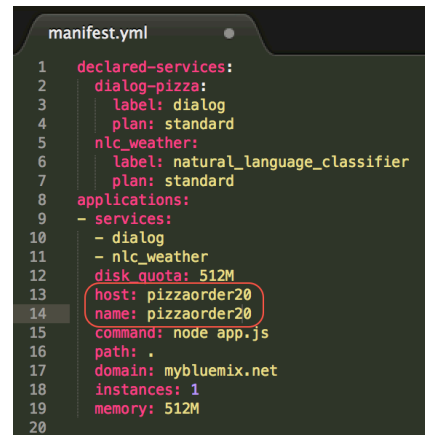
## Edit the manifest.yml file

The application manifests file tells `cf push` what to do with applications. This includes everything from how many instances to create and how much memory to allocate to what services applications should use. A manifest can help you automate deployment, especially of multiple applications at once.

1. Open the **manifest.yml** in the working directory.
2. Change the host name (host:) and app name (name:) of the app from **dialog-nodejs** to something meaningful to you (no spaces). In this example, it is **pizzaorder20** (you pick an app name that is more meaningful to you; this will appear in your app URL after you push the app to Bluemix).

The host name is there so you can run it locally using `http://localhost:3000` before pushing the final product to Bluemix.

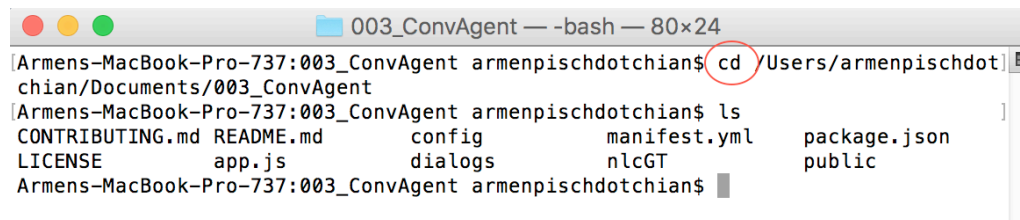
3. Save the file and close it.



```
manifest.yml
1  declared-services:
2    dialog-pizza:
3      label: dialog
4      plan: standard
5    nlc_weather:
6      label: natural_language_classifier
7      plan: standard
8  applications:
9    - services:
10      - dialog
11      - nlc_weather
12      disk_quota: 512M
13      host: pizzaorder20
14      name: pizzaorder20
15      command: node app.js
16      path: .
17      domain: mybluemix.net
18      instances: 1
19      memory: 512M
20
```

## Push your App to Bluemix and add Dialog and NLC services

1. You must now log into Bluemix and you do so by opening a terminal window or a command prompt.
2. Change directory to the folder that houses the app.js file, it's your working top-level directory. Type:  
`cd <location of your top-level-directory>`



```
003_ConvAgent - -bash - 80x24
Armens-MacBook-Pro-737:003_ConvAgent armenpischdotchian$ cd /Users/armenpischdotchian/Documents/003_ConvAgent
Armens-MacBook-Pro-737:003_ConvAgent armenpischdotchian$ ls
CONTRIBUTING.md  README.md      config          manifest.yml    package.json
LICENSE          app.js         dialogs         nlcGT          public
Armens-MacBook-Pro-737:003_ConvAgent armenpischdotchian$
```

3. Login by typing the following command  
`cf login`
4. If it prompts you for an endpoint, type: <https://api.ng.bluemix.net>
5. Type your email and password as prompted. These are the same credentials that you used to create your Bluemix account.
6. Specify your space name, type: `dev` (that is the default space name. It appears in the left panel once you log into Bluemix).



```

Password>
Authenticating...
OK

Targeted org apischdo@us.ibm.com

Select a space (or press enter to skip):
1. dev
2. workshop

Space> dev
Targeted space dev

API endpoint: https://api.ng.bluemix.net (API version: 2.44.0)
User: apischdo@us.ibm.com
Org: apischdo@us.ibm.com
Space: dev
Armens-MacBook-Pro-737:dialog-nodejs armenpischdotchian$

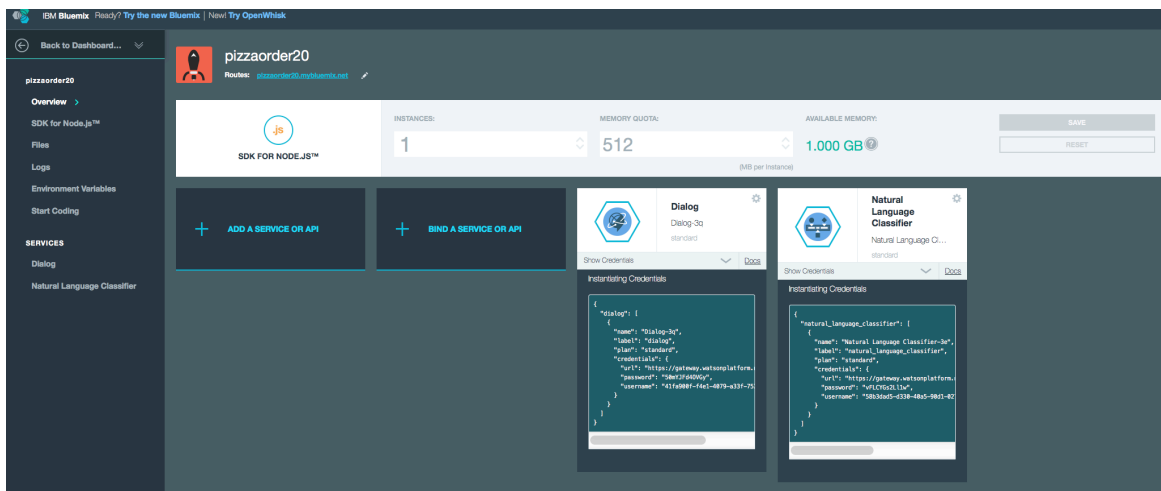
```

7. Push app to Bluemix. You don't need to specify app name because it is picked up from the manifest.yml file:  
`cf push`
8. Wait until it returns a prompt.

## Add Dialog and NLC services

You perform the following steps in Bluemix.

1. Sign into Bluemix and click the Dashboard link.
2. Open your application on the Dashboard. Allow enough time for the app to complete staging and its health denotes "Running." If the app had stopped at it's last stage, click **START**. This may take a few minutes.
3. Click **ADD A SERVICE OR API** twice: once for the **Dialog** app and again for the **NLC** app. Allow each service to bind and then click **CREATE** for each service one at a time, also click **RESTAGE** for each. And if you want to get back to the app to add the second service, click the **Overview** to the left.
4. Click and open the **Show Credentials** link in the bottom of each service, you can alternatively click **Environment Variables** in the left panel.



5. Save the credentials on a text editor for later use in the app.js file.
6. Keep this text editor handy, it is your staging area, you will be copy/pasting lots of code snippets in there.

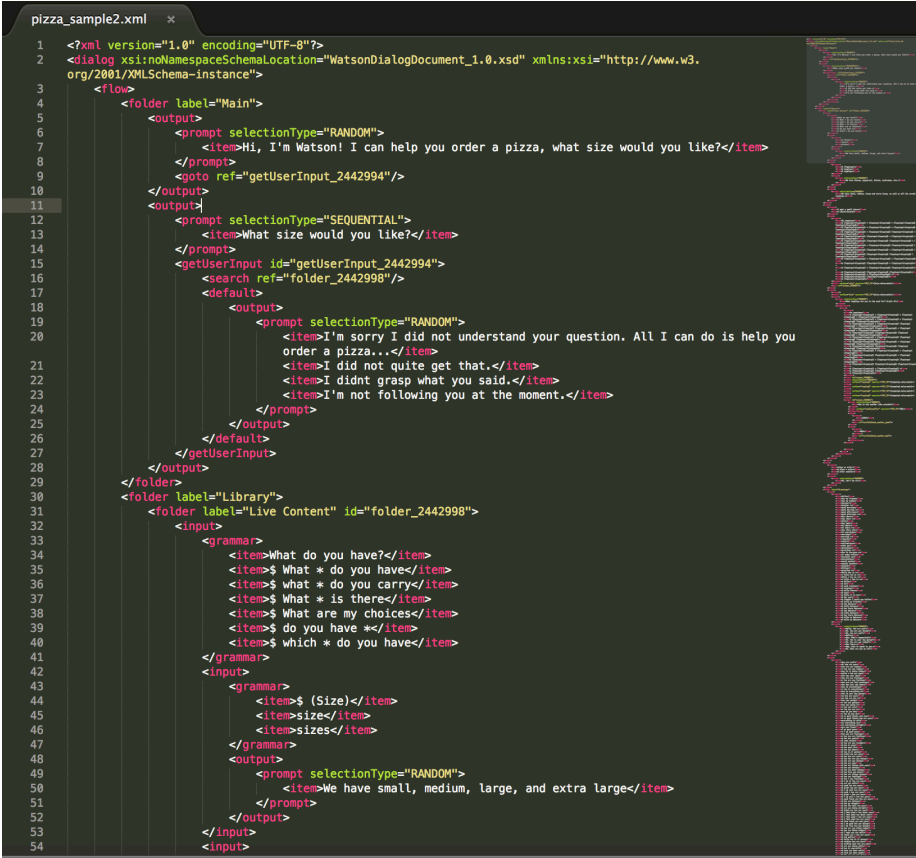
## Customizing the Dialog XML file

When you design your dialog such that it will call NLC based on your response to Watson's output of "What is the weather like outside?" [Or you can use your own sentence, but ensure the word *weather* is included]. If your response reveals inclement weather, then the class or intent of "BAD" will garner a higher confidence value; and if BAD, then Watson will suggest delivery.

If you reply back that the weather is nice, sunny, pleasant etc. then it will call the "GOOD" intent, which now has a higher confidence, hence prompting Watson to suggest that you may want to pick it up.

Take a few minutes and change the conversation format in the XML file. For example you may want to start with a different output message from the help desk.

1. Open the **pizza\_sample2.xml** file and edit **line 7** to a greeting output to your liking. But make sure it ends with asking the user what size they would like.
2. Press Command+F (Mac) on your keyboard and search for **useClassifier**. Perhaps you would like to change line 134 for variation or paraphrase of the same question. It must have something to do with the weather; after all, your ground truth in NLC uses intents for weather related events.
3. Search for **useClassifier** again and it brings you to line 1434. Consider changing the wording in line 1437 and/or 1445. Important to note, this is called **profile variable** in Dialog, and this is how it jumps to NLC. It is hardcoded here by using the exact intent name used in NLC (GOOD, BAD). You can also use relative links. Future releases of Dialog will include a UI, where you can drag and drop various nodes and profile variables, in this lab you get to see the code behind the UI.



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <dialog xsi:noNamespaceSchemaLocation="WatsonDialogDocument_1.0.xsd" xmlns:xsi="http://www.w3.
3   org/2001/XMLSchema-instance">
4   <flow>
5     <folder label="Main">
6       <output>
7         <prompt selectionType="RANDOM">
8           <item>Hi, I'm Watson! I can help you order a pizza, what size would you like?</item>
9         </prompt>
10        <goto ref="getUserInput_2442994"/>
11      </output>
12      <output>
13        <prompt selectionType="SEQUENTIAL">
14          <item>What size would you like?</item>
15        </prompt>
16        <getUserInput id="getUserInput_2442994">
17          <search ref="folder_2442998"/>
18          <default>
19            <output>
20              <prompt selectionType="RANDOM">
21                <item>I'm sorry I did not understand your question. All I can do is help you
22                  order a pizza...</item>
23                <item>I did not quite get that.</item>
24                <item>I didnt grasp what you said.</item>
25                <item>I'm not following you at the moment.</item>
26              </prompt>
27            </output>
28          </default>
29        </getUserInput>
30      </output>
31    </folder>
32    <folder label="Library">
33      <folder label="Live Content" id="folder_2442998">
34        <input>
35          <grammar>
36            <item>What do you have?</item>
37            <item>$ What * do you have</item>
38            <item>$ what * do you carry</item>
39            <item>$ What * is there</item>
40            <item>$ What are my choices</item>
41            <item>$ do you have *</item>
42            <item>$ which * do you have</item>
43          </grammar>
44          <input>
45            <grammar>
46              <item>$ (Size)</item>
47              <item>size</item>
48              <item>sizes</item>
49            </grammar>
50            <output>
51              <prompt selectionType="RANDOM">
52                <item>We have small, medium, large, and extra large</item>
53              </prompt>
54            </output>
55          </input>
56        </input>
57      </folder>
58    </folder>
59  </flow>
60 </dialog>
```

Basically, have fun here and remember indentation is crucial.

4. Save the file as `pizza_sample3.xml` (or some other name of your choosing). Be sure to save it in the **Dialog** folder.

At this point you are about to run two cURL commands, each executing a POST method: the *first* one creates a dialog session between the service and the dialog XML file, the result is a `dialog_id` that the service returns. You use this `dialog_id` in a *second* POST cURL command to set the environment variables in your code. Basically, you are telling the rest of your code what to do with a newly established dialog.

1. Before you run the command, you need to change directory to the dialogs location. From the same terminal window `cd` to **dialogs**.
2. Run the following command (copy/paste it in your terminal window) using the variables from the generated service specific to your instance:

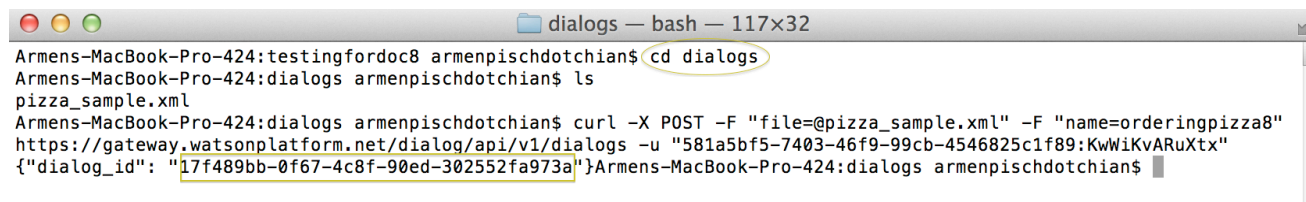
```
curl -X POST -F "file=@pizza_sample3.xml" -F "name=pizzaorder20"
https://gateway.watsonplatform.net/dialog/api/v1/dialogs -u "581a5bf5-
7403-46f9-99cb-4546825c1f89:KwWiKvARuXtx"
```

Bear in mind that the app name, username and password are specific to your settings. The XML file is named **pizza\_sample3.xml** and resides in the **dialogs** folder. Everything is on one line.

Pay close attention to the quotation marks. They need to be straight, courier font, not curved as the case may be if you copy/pasted from a rich text format.

The output looks like the example below except depicting *your* `dialog_id`.

In this example, it is: `17f489bb-0f67-4c8f-90ed-302552fa973a`

A terminal window titled 'dialogs — bash — 117x32' on a Mac. The prompt is 'Armens-MacBook-Pro-424:testingfordoc8 armenpischdotchian\$'. The user enters 'cd dialogs' (highlighted with a yellow oval). The prompt changes to 'Armens-MacBook-Pro-424:dialogs armenpischdotchian\$'. The user enters 'ls', showing 'pizza\_sample.xml'. The user then enters a long cURL command: 'curl -X POST -F "file=@pizza\_sample.xml" -F "name=orderingpizza8" https://gateway.watsonplatform.net/dialog/api/v1/dialogs -u "581a5bf5-7403-46f9-99cb-4546825c1f89:KwWiKvARuXtx"'. The output is '{"dialog\_id": "17f489bb-0f67-4c8f-90ed-302552fa973a"}' (the ID is highlighted with a yellow box). The prompt returns to 'Armens-MacBook-Pro-424:dialogs armenpischdotchian\$'.

3. Now that you have a dialog created, the code needs to know about it. To do so, set an environment variable to point to the `dialog_id`. But first, change directory back up one level (`cd ..`)
4. Run the following command substituting your app name and `dialog_id`. Allow enough time for the modules to load and that you see the message: App started.

```
cf se pizzaorder8 DIALOG_ID 17f489bb-0f67-4c8f-90ed-302552fa973a
```

5. You may be prompted to restage your app. If so, run the following command:

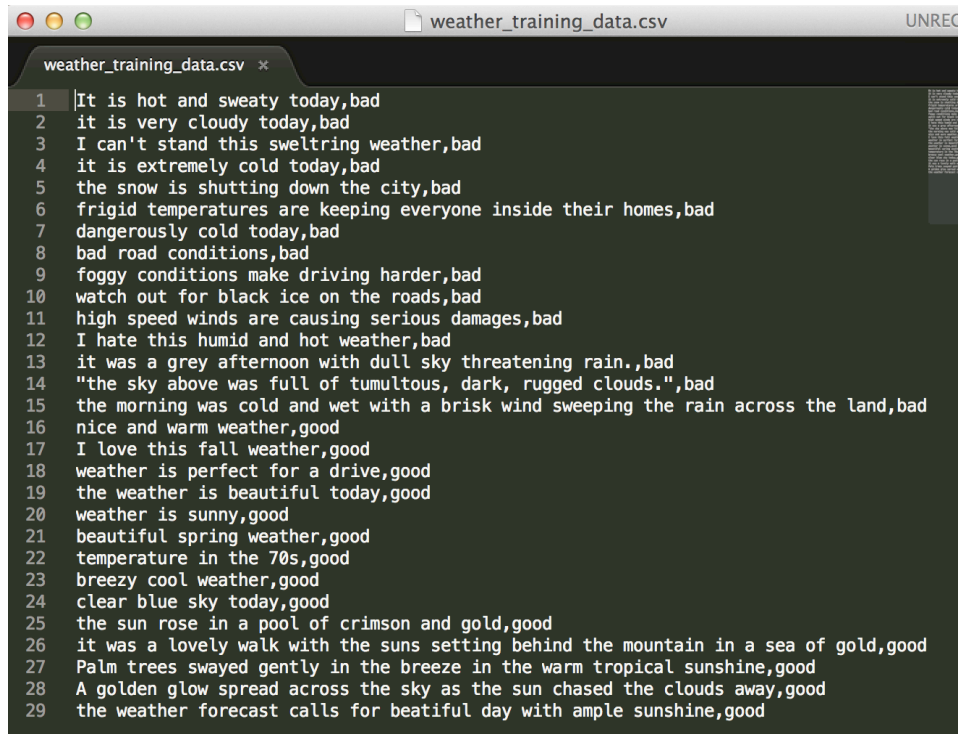
```
cf restage <your_app_name>
```

Lots of activities are taking place now. This will take easily a minute or two. Wait until the prompt is returned with state as *running* along with time stamp, memory, cpu and all that good stuff.

## Editing the NLC Ground Truth

The exercise you are about to perform with NLC entails that you train (machine learning using convolution neural network) the **weather\_training\_data.csv** file. A sample CSV file is provided for you where add a few more weather related questions and classify them as “good” or “bad.”

Notice the format: question, comma and intent or class [no spaces].



```
weather_training_data.csv
1 It is hot and sweaty today,bad
2 it is very cloudy today,bad
3 I can't stand this sweltering weather,bad
4 it is extremely cold today,bad
5 the snow is shutting down the city,bad
6 frigid temperatures are keeping everyone inside their homes,bad
7 dangerously cold today,bad
8 bad road conditions,bad
9 foggy conditions make driving harder,bad
10 watch out for black ice on the roads,bad
11 high speed winds are causing serious damages,bad
12 I hate this humid and hot weather,bad
13 it was a grey afternoon with dull sky threatening rain.,bad
14 "the sky above was full of tumultuous, dark, rugged clouds.",bad
15 the morning was cold and wet with a brisk wind sweeping the rain across the land,bad
16 nice and warm weather,good
17 I love this fall weather,good
18 weather is perfect for a drive,good
19 the weather is beautiful today,good
20 weather is sunny,good
21 beautiful spring weather,good
22 temperature in the 70s,good
23 breezy cool weather,good
24 clear blue sky today,good
25 the sun rose in a pool of crimson and gold,good
26 it was a lovely walk with the suns setting behind the mountain in a sea of gold,good
27 Palm trees swayed gently in the breeze in the warm tropical sunshine,good
28 A golden glow spread across the sky as the sun chased the clouds away,good
29 the weather forecast calls for beautiful day with ample sunshine,good
```

Keep in mind that the NLC service is already training on Wikipedia, so entity relations, sentiment and an entire NLP stack are already in place. The CSV file that you are using as ground truth narrows it down to a domain specific to your interests. Let's begin.

1. In the Finder or Windows Explorer, navigate to the **nlcGT** folder and open the **weather\_training\_data.csv** file.
2. Add a few questions of yours, followed by a comma and the intent (no space after the comma).
3. Save the file (no need to change it's name).
4. From a command window or terminal, change directory to the **nlcGT** folder that you copied earlier to your working directory.
5. Run the following command (of course, using your credentials instead of the example below and your path to the csv file will be different than the example below, change that also first in your sandbox text editor).

```
curl -i -u "7acal872-4bd1-4d78-8014-e71b9a76beef":"tRTgpdscx6n8" -F training_data=@
/Users/armenpischdotchian/Documents/02_Dialog-
NLC/testingForDoc8/nlcGT/weather_training_data.csv -F
training_metadata="{\"language\":\"en\", \"name\":\"TutorialClassifier\"}"
https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers -v -k
```

Few things to watch out for: there is no space after @ and the directory path; and double check your double quotes, they have to be san serif, straight up and down, not rich text curled. This entire command is in one line.

It takes upwards of 10 minutes or so for the training to complete. Larger CSV files with thousands of questions will take longer, but it is not a linear time frame. Just because you have twice as much data, does not mean it will take twice as long. Typical client ground training may take closer to one hour.

6. To check the status of your training, run the following command:

```
curl -u "922b8968-2135-40c0-9eb7-55f8b9500ab4":"ZbHZUgfYCOT6"  
"https://gateway.watsonplatform.net/natural-language-  
classifier/api/v1/classifiers/<classifier_id>"
```

Notice that you now have a new classifier\_id

```
{  
  "classifier_id" : "A3FCCBx16-nlc-1677",  
  "name" : "TutorialClassifier",  
  "language" : "en",  
  "created" : "2015-12-28T00:57:23.413Z",  
  "url" : "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/  
/classifiers/A3FCCBx16-nlc-1677",  
  "status" : "Training",  
  "status_description" : "The classifier instance is in its training phase, not  
yet ready to accept classify requests"  
* Connection #0 to host gateway.watsonplatform.net left intact
```

Notice that the status is **Training**, run the following command periodically, until you see the status **Available**. For this exercise, it may be upwards of ten minutes.

```
curl -u "userid":"password" "https://gateway.watsonplatform.net/natural-language-  
classifier/api/v1/classifiers/<classifier_id>"
```

## Edit the app.js file

You are now ready to populate the app.js file with a total of six entries, three for dialog and three for NLC.

1. Open the **app.js** file.
2. Edit lines 32, 33 and 37 with relevant information regarding the Dialog service.
3. Edit lines 45, 46 and 51 with relevant information from the NLC service.

```
28  
29 // if bluemix credentials exists, then override local  
30 var credentials = extend({  
31   url: 'https://gateway.watsonplatform.net/dialog/api',  
32   username: '581a5bf5-7403-46f9-99cb-4546825c1f89',  
33   password: 'KwWiKvARuXtx',  
34   version: 'v1'  
35 }, bluemix.getServiceCreds('dialog')); // VCAP_SERVICES  
36  
37 var dialog_id = process.env.DIALOG_ID || '17f489bb-0f67-4c8f-90ed-302552fa973a';  
38  
39 // Create the service wrapper  
40 var dialog = watson.dialog(credentials);  
41  
42 // add NLC call  
43 var nlc_credentials = extend({  
44   url: 'https://gateway.watsonplatform.net/natural-language-classifier/api',  
45   username: '7aca1872-4bd1-4d78-8014-e71b9a76beef',  
46   password: 'tRTgpdscx6n8',  
47   version: 'v1'  
48 }, bluemix.getServiceCreds('natural_language_classifier')); // VCAP_SERVICES  
49 var nlc = watson.natural_language_classifier(nlc_credentials);  
50  
51 var nlc_id = process.env.NLC_ID || 'A3FCCBx16-nlc-1677';  
52  
53 app.post('/conversation', function(req, res, next) {  
54   var params = extend({ dialog_id: dialog_id }, req.body);  
55   var dialogresp = JSON.stringify(params.response);  
56   // check if the statement from Dialog references the weather  
57   // If yes, call NLC
```

4. Save the file.

## Run the app locally

1. Run the following command from the same terminal or window (from prior step) to install the npm modules. But first, ensure you are in the same directory that app.js lives. Last you were in the nlcGT directory, so you will likely have to do a `cd ..` (cd space followed by two dots).

```
npm install
```

2. Run the app locally by issuing the following command:

```
node app.js
```

3. Copy the resulting localhost URL (`http://localhost:3000`) in a new browser tab.
4. Test the Dialog app from your local system.

## Deploy your app to Bluemix

You are already logged into Bluemix, so you are one command away from stardom!

1. Run the following command to push the app to Bluemix

```
cf push
```

2. Allow for the staging to complete until you have a live URL evident in the details of your application.
3. Test app in Bluemix.

Congratulations, you have created an application live on Bluemix that is using cognitive services. We have left lots of room for you to improve on the scenario, better yet, use other ground truths or dialogs of your choosing and various services to enrich your application.

## Try the service

The image shows a chat application interface on the left and its Bluemix console details on the right.

**Chat Interface:**

- Message 1: User asks "How is the weather where you are?". Assistant responds "very cold and raining hard".
- Message 2: User asks "Ok, let us deliver it for you just to make life a little easier!".
- Input field: "Type a response and hit enter"

**Bluemix Console Details:**

- Tabs: Data, JSON
- Section: Information
  - Dialog ID: 2ef18f39-931f-4132-8039-6cfc67f962c0
  - Conversation ID: 2325642
  - Client ID: 2336408
- Section: Profile
  - size: Large
  - topping1: Olives
  - useClassifier: YES