InterConnect
2017

March 19–23
MGM Grand &
Mandalay Bay
Las Vegas, NV

IBM

Hands-on Lab Session 7204 **Build a Watson Powered Chatbot**

Mitch Mason, IBM

Contents:

1. Creating an instance of Watson Conversation

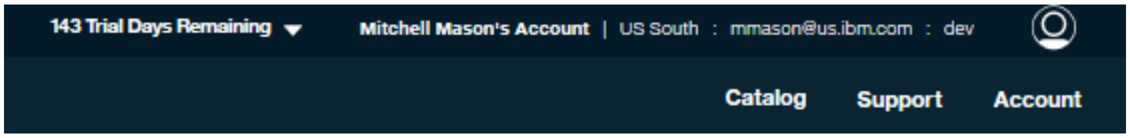
a. Accessing Bluemix

i. Navigate to Bluemix.net

ii. Create an account if you don't already have one

iii. Login using your Bluemix ID

iv. Click **Catalog** in the top navigation bar


143 Trial Days Remaining
Mitchell Mason's Account | US South : mmason@us.ibm.com : dev
Catalog
Support
Account

v. Filter by Watson on the left and click **Conversation**

vi. Enter *Conversation-Demo* as the service name, and use the provided defaults for the other fields and click on the **Create** button.

1. Note that by clicking **View Docs** on the left, you can view the Conversation service documentation, including sample applications and API information.

   https://www.ibm.com/watson/developercloud/doc/conversation/

b. Accessing the Conversation Service tooling

i. Navigate back to your dashboard within Bluemix by clicking **IBM Bluemix** in the top right of the navigation bar

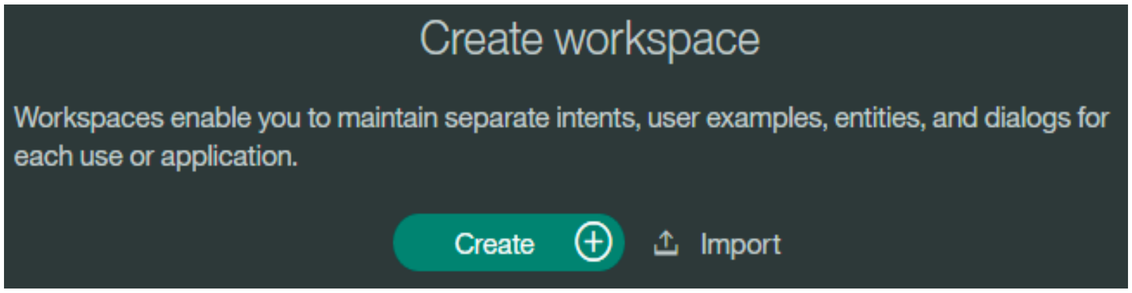
ii. Click the newly created Conversation service instance under **All Services**

iii. Click **Manage** on the left menu bar

iv. Click the 'Launch Tool' button to access the Conversation Service tooling

2. Creating a Workspace

a. Click **Create** near the top

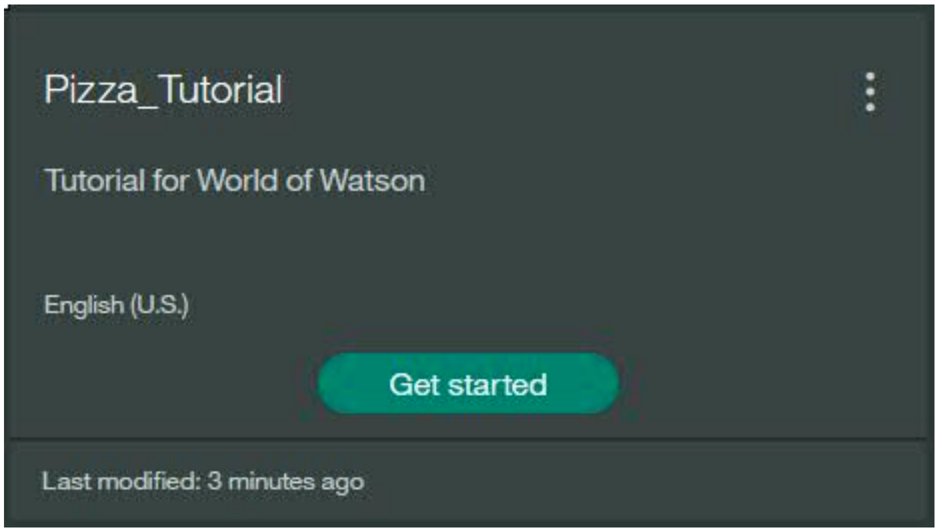


Create workspace

Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.

b. Give the workspace a name (e.g. Pizza_Tutorial) and a description. This manual will focus on creating a pizza ordering application.

c. Click **Create**, and then click **Get Started** on the newly created workspace to enter the tooling.



Pizza_Tutorial

Tutorial for World of Watson

English (U.S.)

Last modified: 3 minutes ago

3. There are 3 pieces to creating a conversational application

a. Intents – The action a user wants to take

b. Entities – The object a user wants to take action on or is referring to

c. Dialog – The conversation flow the end user will experience

   i. Typically, we use Intents & Entities to *understand* what the user has entered, and user Dialog to respond to the user's input.

4. Creating Intents

a. In order for the application to understand what the user is entering we will need to create a few intents. A detailed description and explanation of intents can be found at:

https://www.ibm.com/watson/developercloud/doc/conversation/intent_ovw.shtml
Below are the steps needed in order to create intents for the Pizza app:

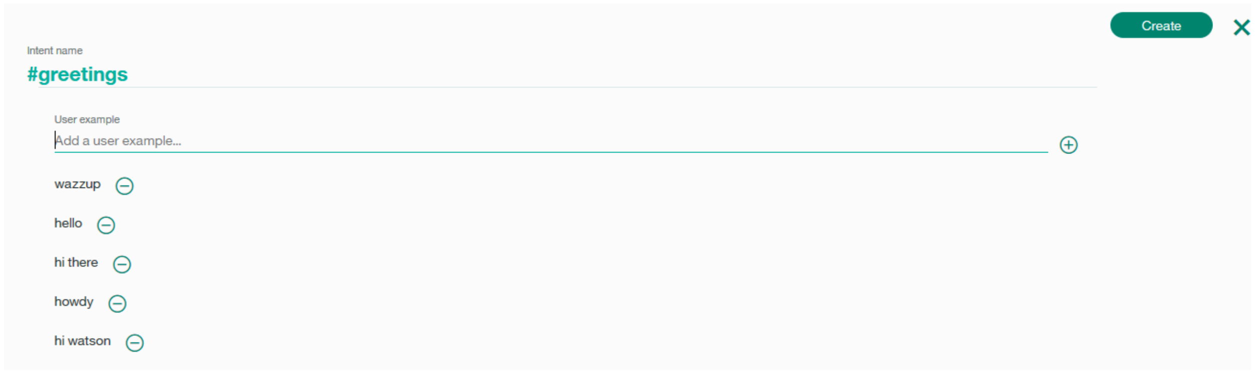
b. Intent recognition within the Conversation service makes use of Machine Learning and Natural Language Processing (NLP). This means you can provide a handful of intent examples, and the service can understand variations of the samples you provided. We recommend providing at least five intent examples, but typically twenty or so examples is considered a 'good' intent definition.

c. The first intent we create will be for greetings (e.g. "hi", "hello" etc...)

d. Click **Create** to begin creating intents.

   i. Next to the hash tag, you will type the intent name. In this case 'greetings'. You will refer to intents in your dialog editor using the hashtag symbol (#).

   ii. Now provide 5 examples of ways your users might greet the service.



#greetings

wazzup
hello
hi there
howdy
hi watson

   i. When you have finished entering the intents, click **Create**.

   ii. We can try out our new intent by clicking on the chat bubble in the upper right corner. Type in a greeting to see how this looks.

Note: The service takes a few minutes to "train" when new intents/entities are added to the workspace. You will not be able to test the changes until training completes.

1. In general, when creating intents it is encouraged to use "real world" examples of how end users would interact with the system. For a Pizza app it is pretty easy to guess how a user would enter various utterances to add toppings/order a pizza, but for most applications real end user data is very beneficial.

2. We have recently added in a built in 'Irrelevant' class to our algorithm. This should help the service understand what is not relevant to your

existing data. It's not perfect, so if you are testing, and see something classified incorrectly, you can change it in the Try It Out panel, or mark as irrelevant.

e. We can create a few more intents that we will need later on:
(For each of these intents provide at least five examples, four examples are given for each to get you going!)

   i. #Order_pizza

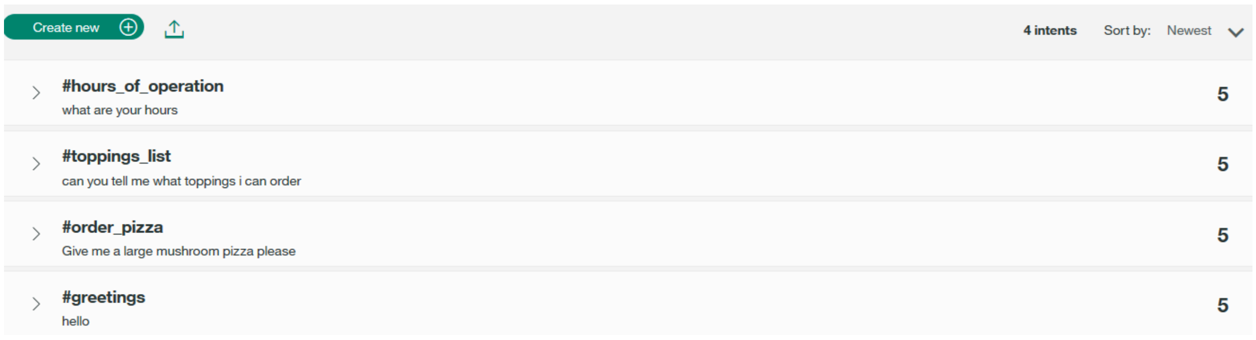
      a. I want to order a pizza
      b. I need a pepperoni pizza
      c. I would like to order a small pizza
      d. Give me a large mushroom pizza please

   ii. #toppings_list

      a. what toppings do you have
      b. can you tell me what toppings I can order
      c. what are my toppings choices
      d. do you have pepperoni

   iii. #hours_of_operation

      a. When do you open
      b. What time can I come in
      c. What are your hours
      d. When do you close



#hours_of_operation what are your hours	5
#toppings_list can you tell me what toppings i can order	5
#order_pizza Give me a large mushroom pizza please	5
#greetings hello	5

5. Creating Entities

a. Next we will create some entities. Entities are generally used to determine *what* the user is talking about. A detailed description of entities, and how to create them can be found at:
https://www.ibm.com/watson/developercloud/doc/conversation/entity_ovw.shtml

b. Currently, entities are extracted using character matching. This way, we can recognize even the most specific jargon, like “McNuggets.”

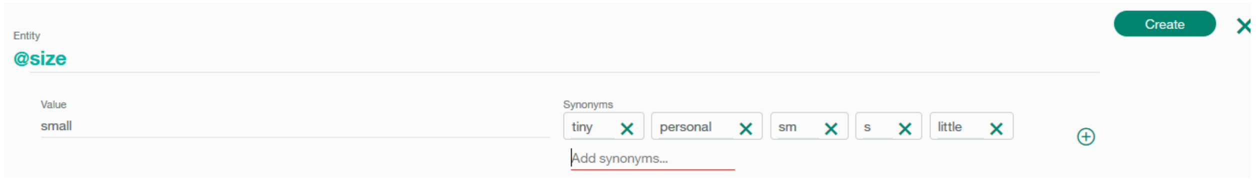
c. We will be creating two entities for our application: *size* and *toppings*

d. Start by navigating to the **Entities** tab within the tooling.

e. Click on the **Create New** button to create your first entity.

f. Enter Size as the entity name, with Small being the first value.

g. In this case, we also need to list all possible entity synonyms our users might enter.



Entity
@size

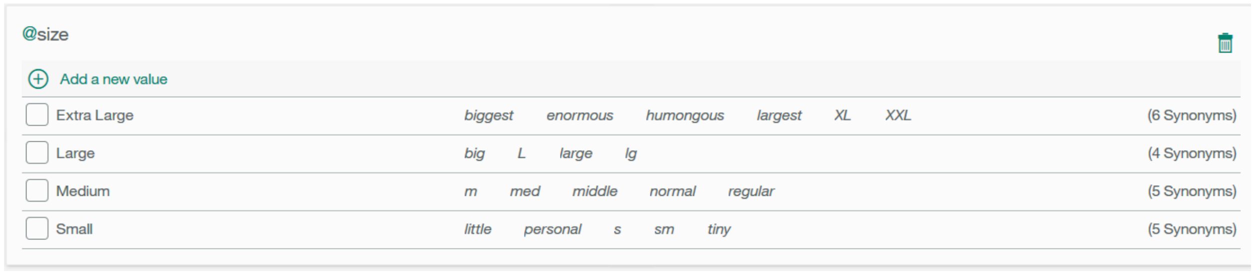
Value	Synonyms
small	tiny, personal, sm, s, little

h. Click the plus on the right to create the next size

i. Name the next entity *Medium*, again adding synonyms for medium.

j. Repeat the previous two steps for *Large* and *Extra Large.*

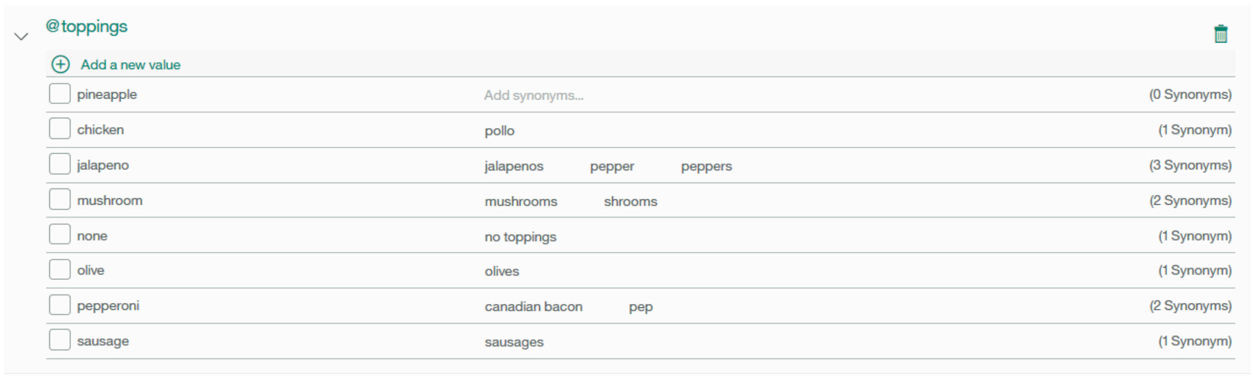
k. Once all entity values (and synonyms) are entered press **Create** to have the tooling save the values to the system.



@size

Extra Large	biggest enormous humongous largest XL XXL	(6 Synonyms)
Large	big L large lg	(4 Synonyms)
Medium	m med middle normal regular	(5 Synonyms)
Small	little personal s sm tiny	(5 Synonyms)

l. Next, create an entity for *Toppings*.



@toppings

Add a new value

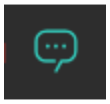
pineapple	Add synonyms...	(0 Synonyms)
chicken	pollo	(1 Synonym)
jalapeno	jalapenos pepper peppers	(3 Synonyms)
mushroom	mushrooms shrooms	(2 Synonyms)
none	no toppings	(1 Synonym)
olive	olives	(1 Synonym)
pepperoni	canadian bacon pep	(2 Synonyms)
sausage	sausages	(1 Synonym)

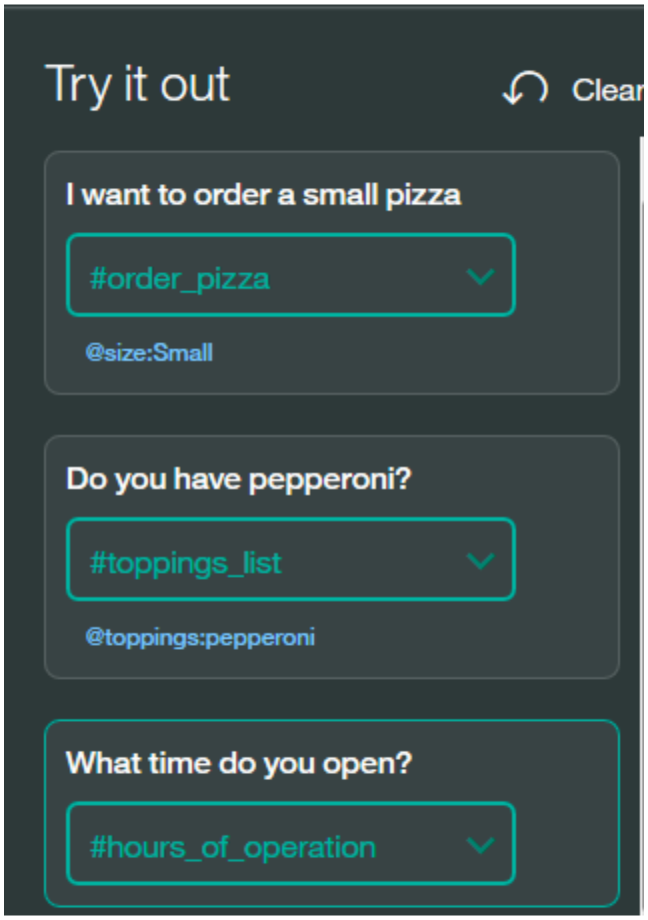
m. Because these are fixed lists, there is no minimum or maximum number of synonyms or values you need to have. You want to be all inclusive for the ways your users will communicate.

6. Testing the Natural Language Understanding (NLU)

a. For the purposes of this tutorial, we will refer to the intents and entities as NLU. This is the information the Machine Learning and Natural Language algorithms extract from a user's utterance (input).

b. Try typing in some things users might actually say, and see how the service understands them in the **Try it out** panel. Hint; the "Try it out" panel can be displayed by clicking on the following icon in the top right:





Try it out

Clear

I want to order a small pizza

#order_pizza

@size:Small

Do you have pepperoni?

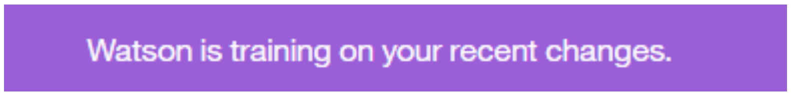
#toppings_list

@toppings:pepperoni

What time do you open?

#hours_of_operation

If you see this message, you must wait until training completes



Watson is training on your recent changes.

c. Notice that anytime toppings were present, they are represented by an @ sign, and intents by a #. This is how we will condition our Dialog in the following section.

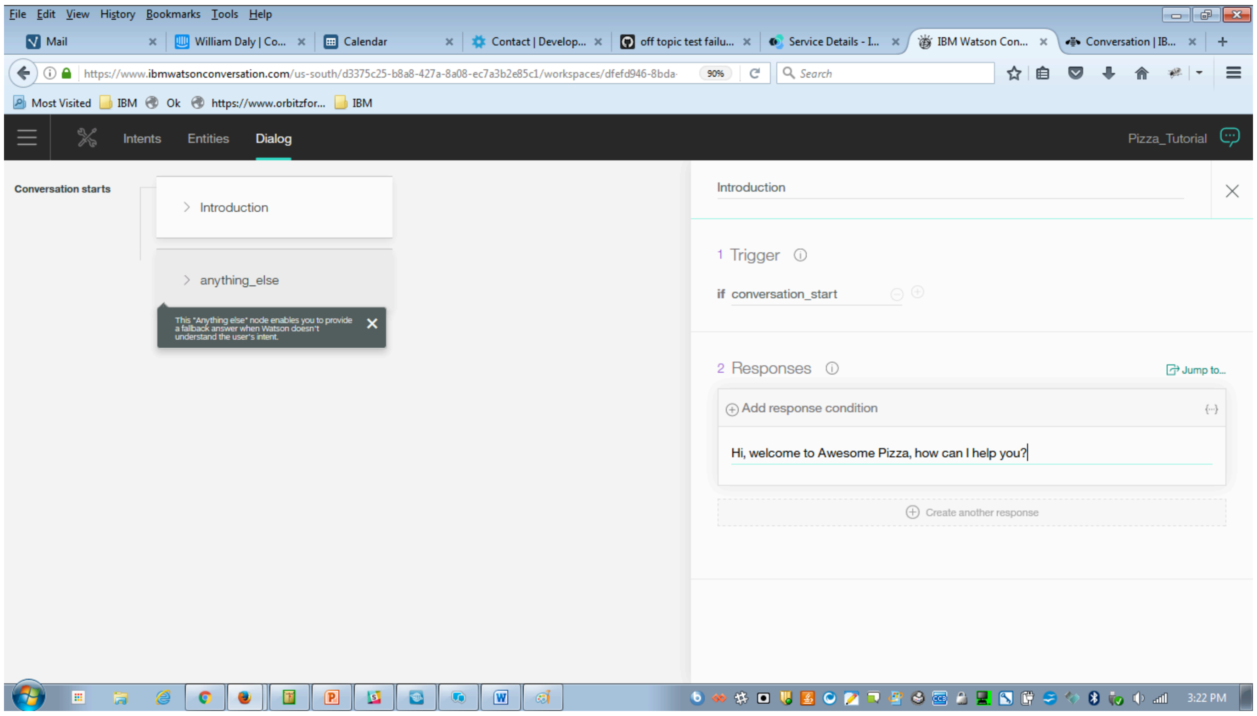
7. Creating Dialog

a. Dialog is the term we use to describe how the system responds to the end user. Typically, we will use the intent of the user's input, and any entities found within the input to determine what the system should return to the end user. A more detailed description of Dialog and its main concepts can be found at: https://www.ibm.com/watson/developercloud/doc/conversation/dialog_ovw.shtml

b. The Dialog editor is where we design the conversation flow your end users will have with your application. Click the **Dialog** tab at the top of the screen.

c. Click **Create**.

d. Typically, you will want to open your conversation with a greeting. We can use a system condition of 'conversation_start' to give an opening message before users say anything. First, we will give our node a name; this is simply a description for your benefit. I have chosen 'Introduction.' Next, in the 'if' field type 'conversation_start' and click **Create new condition. Note:** There are 3 **Create new…** options, you want **condition**. The other two create intents or entities.


File Edit View History Bookmarks Tools Help
https://www.ibmwatsonconversation.com/us-south/d3375c25-b8a8-427a-8a08-ec7a3b2e85c1/workspaces/dfefd946-8bda
Intents
Entities
Dialog
Pizza_Tutorial
Conversation starts
Introduction
anything_else
Trigger
Responses
Jump to...
Add response condition
Create another response

This "Anything else" node enables you to provide a fallback answer when Watson doesn't understand the user's intent.

Introduction

if conversation_start

Hi, welcome to Awesome Pizza, how can I help you?

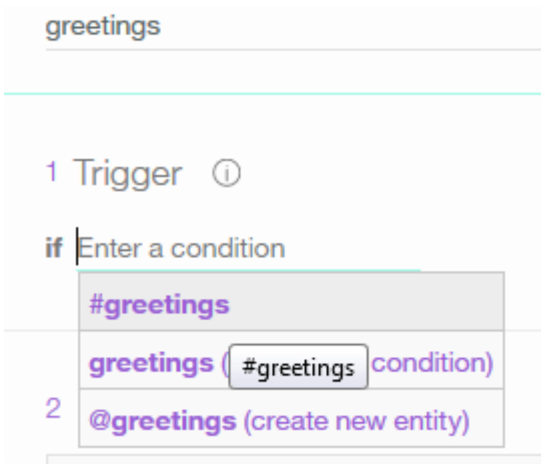
e. You type your conditions in the if section of the box, and the response in the Responses section. Each box within the Dialog editor is referred to as a *Dialog Node*, or sometimes *Node* for short.

i. Notice the system automatically creates a condition for "Anything else", this is used as a fallback when Watson does not have any matching conditions. We will leave this new Dialog node alone for now.

f. The first thing a person might say would be hello, so we should use our previously created *greetings* intent to detect this and respond accordingly.

g. Click the conversation_start node, and then click the plus at the bottom of the node to create a 'sibling' node below. This means they are at the same level of the conversation tree and are evaluated at the same time.

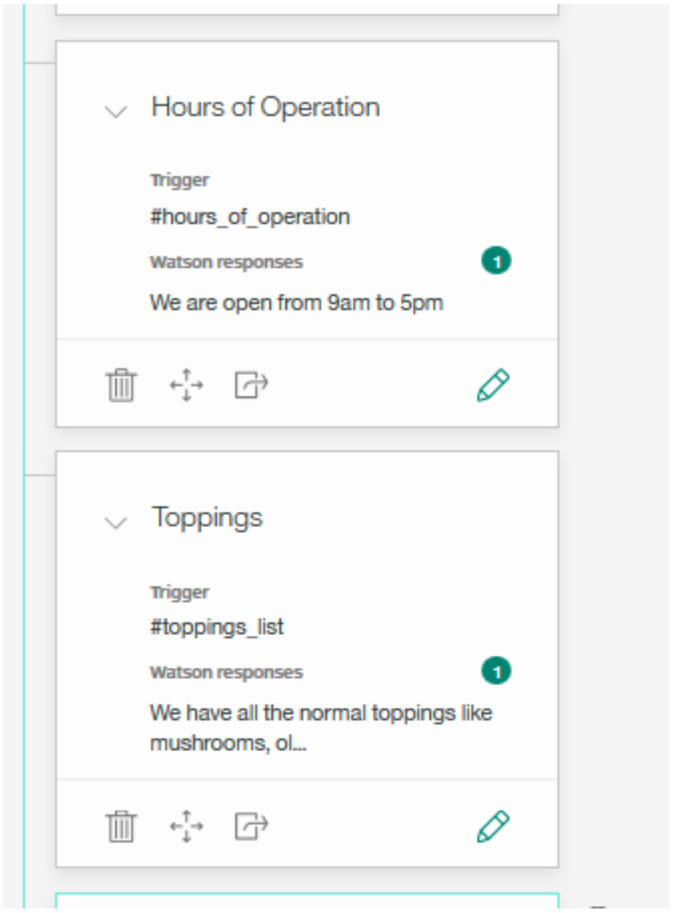
i. Start typing **greetings** in the 'if' section and you will see the system display the closest matching option, in this case, #greetings, which is the intent we've already created.


greetings
1 Trigger
if Enter a condition
#greetings
greetings (#greetings condition)
2
@greetings (create new entity)

h. Give a natural response, like “Hey there! How can I help you with your pizza order?”

i. Always make sure to keep your users on topic and steer them back to the conversation you have designed for.

i. Next, let’s create sibling nodes for our other two basic intents, #hours_of_operation and #toppings_list, we will save ordering a pizza for later as it is more complex and requires a deeper conversation. Be sure to click outside the node to save your changes.



Hours of Operation

Trigger
#hours_of_operation

Watson responses
We are open from 9am to 5pm

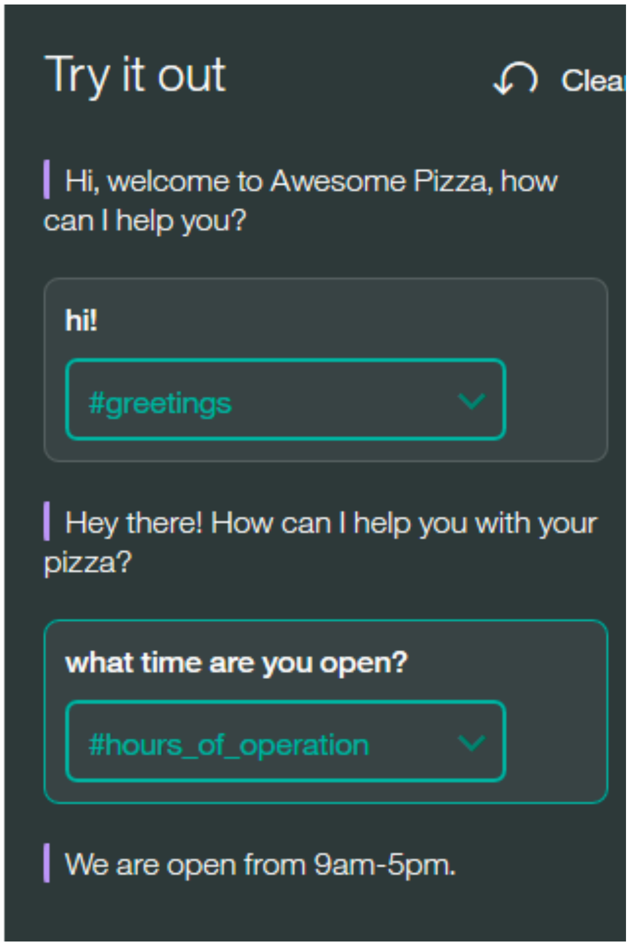
Toppings

Trigger
#toppings_list

Watson responses
We have all the normal toppings like mushrooms, ol...

“We have the normal toppings: pepperoni, mushrooms, olives, jalapenos, sausage & chicken.”

j. We can test the responses we have created to make sure everything is working properly so far.



Hi, welcome to Awesome Pizza, how can I help you?

hi!

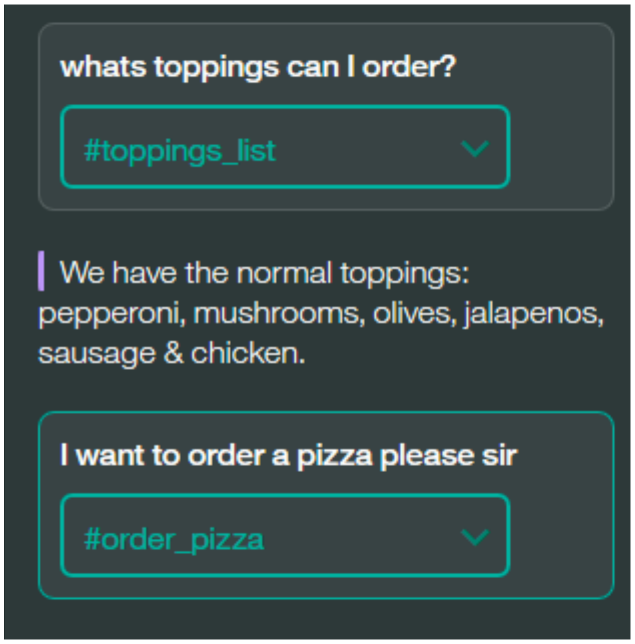
#greetings

Hey there! How can I help you with your pizza?

what time are you open?

#hours_of_operation

We are open from 9am-5pm.



whats toppings can I order?

#toppings_list

We have the normal toppings: pepperoni, mushrooms, olives, jalapenos, sausage & chicken.

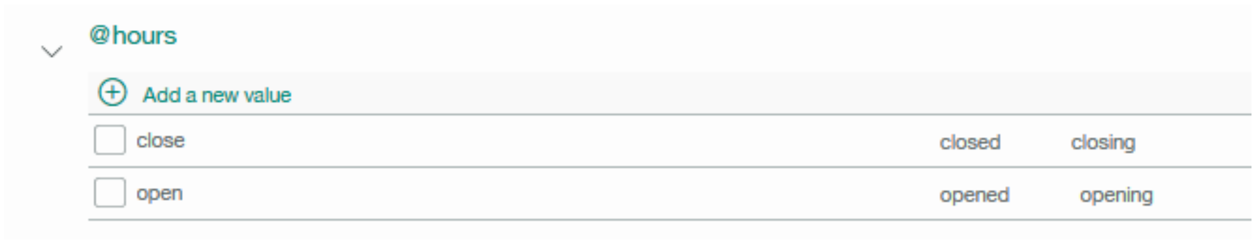
I want to order a pizza please sir

#order_pizza

k. Make sure you get all expected responses as shown above before moving on to creating a more complex Dialog for building a pizza.

8. Using Multiple Conditional Responses (MCR) to answer more accurately

a. One of our newer features is called MCR. It allows you to easily respond to multiple conditions in a single dialog box. A simple example would be responding to whether someone asked about opening or closing times. First, we will create entities for open and for closed.



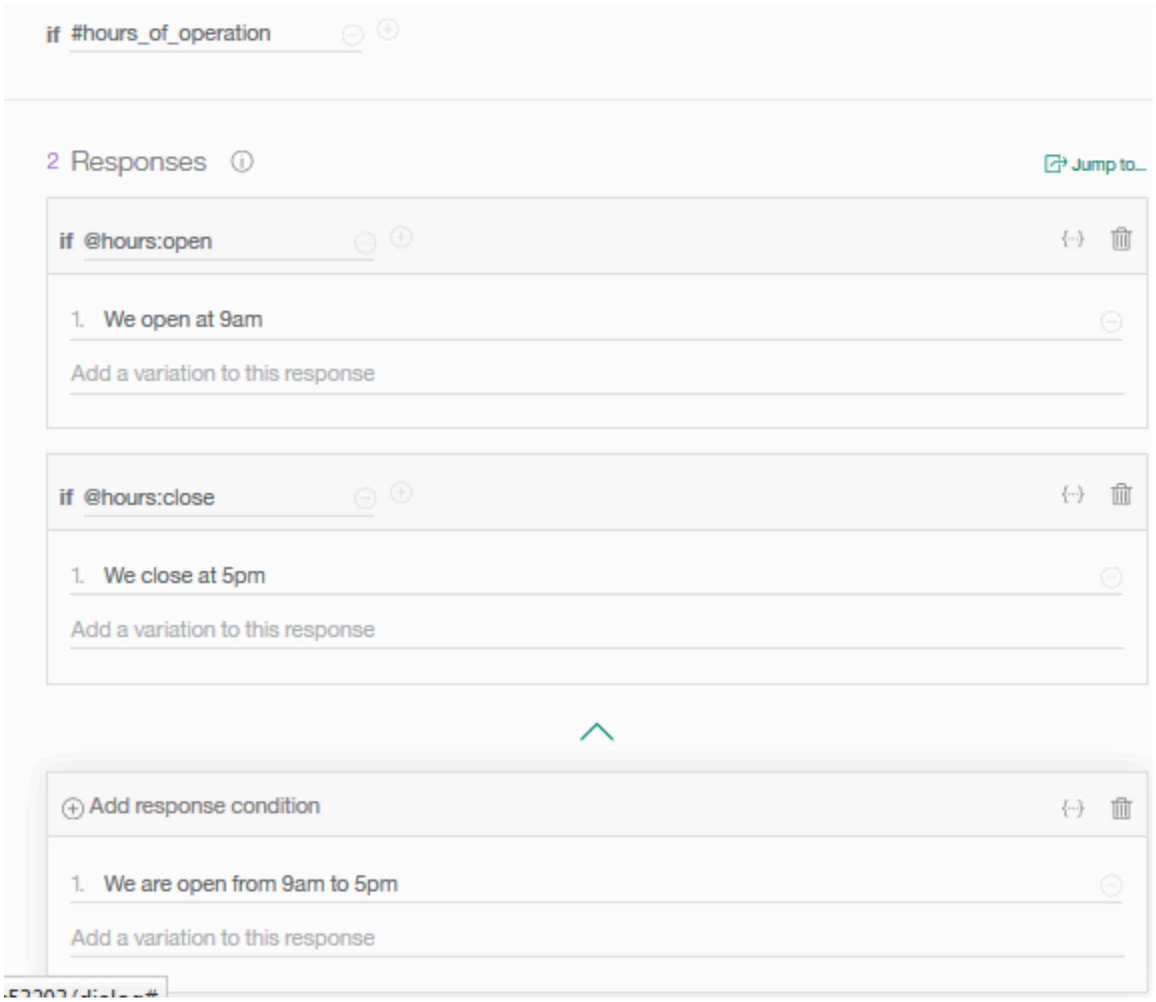
close	closed	closing
open	opened	opening

b. Next, we will create additional response conditions in our Dialog node for #hours_of_operation

   i. First we click the plus for **Add response condition** and add a condition for @hours:open. This means that if the user asked something about hours of operation, matched by our intent, and used one of our entities for open, we will give a specific message. "We open at 9am."

   ii. Do the same for the close entity.

   iii. Finally, create a third response for when no entity is given, that gives all of the information "We open at 9am and close at 5pm." While this is a very basic example, the MCR feature allows us to do a lot more within a single Dialog node, keeping our overall design much cleaner and free of extra boxes.



if #hours_of_operation

2 Responses

Jump to...

if @hours:open

1. We open at 9am

Add a variation to this response

if @hours:close

1. We close at 5pm

Add a variation to this response

Add response condition

1. We are open from 9am to 5pm

Add a variation to this response

9. Running a Sample Application Locally

Before we create a more in depth dialog, we're going to deploy a sample UI so that it is easier to view the context that is being set. This helps us remind the stateless Conversation service what has already been covered, like which size or toppings a user has chosen. In this chapter, we will configure your development environment and show how to run a local application which connects to the instance of the Conversation service you've configured.

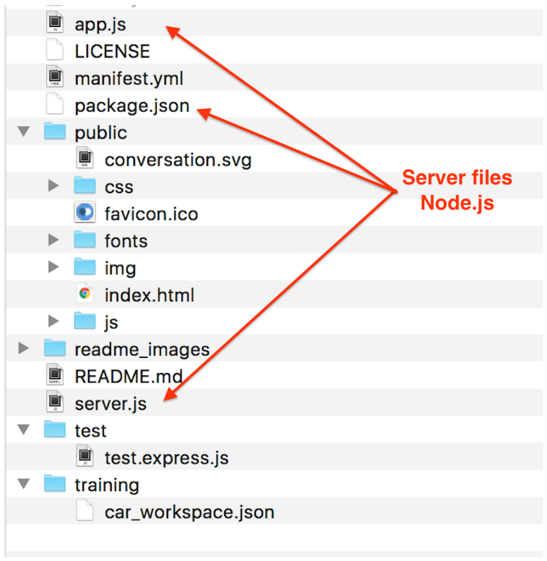
a. Cloning the application

Navigate to
https://github.com/watson-developer-cloud/conversation-simple

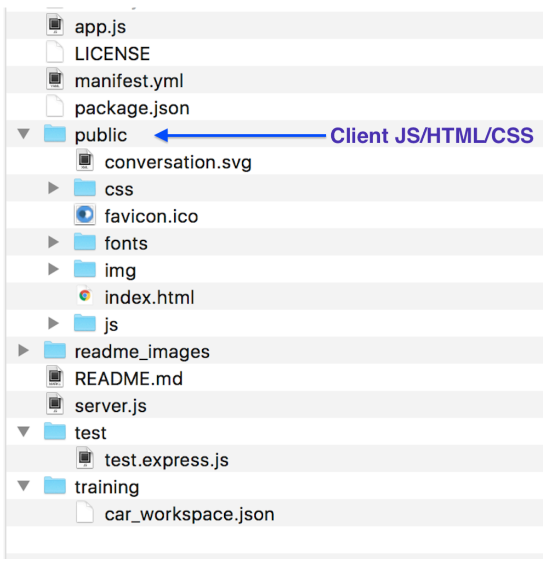
And download this repo to your local machine to get started. You can use the command "git clone https://github.com/watson-developer-cloud/conversation-simple" in the Terminal to create a folder and download the lab files.

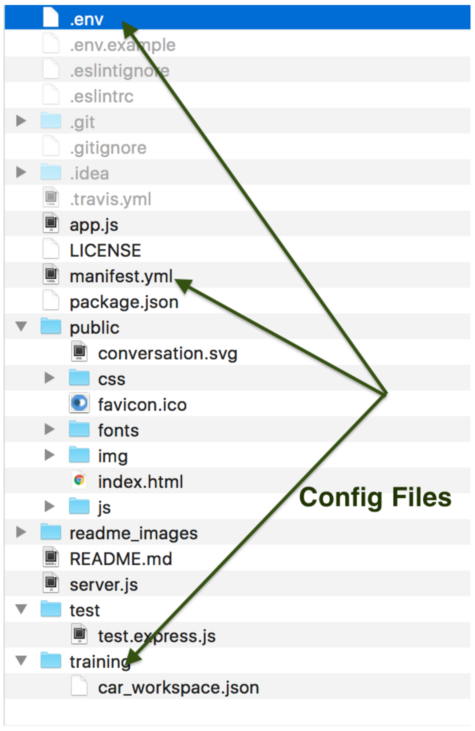
b. Application code overview

The application can essentially be broken down into three sections; server code (node.js code), client code (js/html/css) and configuration files:



```
app.js
LICENSE
manifest.yml
package.json
public
    conversation.svg
    css
    favicon.ico
    fonts
    img
    index.html
    js
readme_images
README.md
server.js
test
    test.express.js
training
    car_workspace.json
```





c. Configuring the Application to use Watson Conversation Service Instance

i. The application will be run on your local machine.

ii. Rename the file *.env.example* to *.env*

iii. Open the file named *.env*, located at /conversation-simple-master/.env

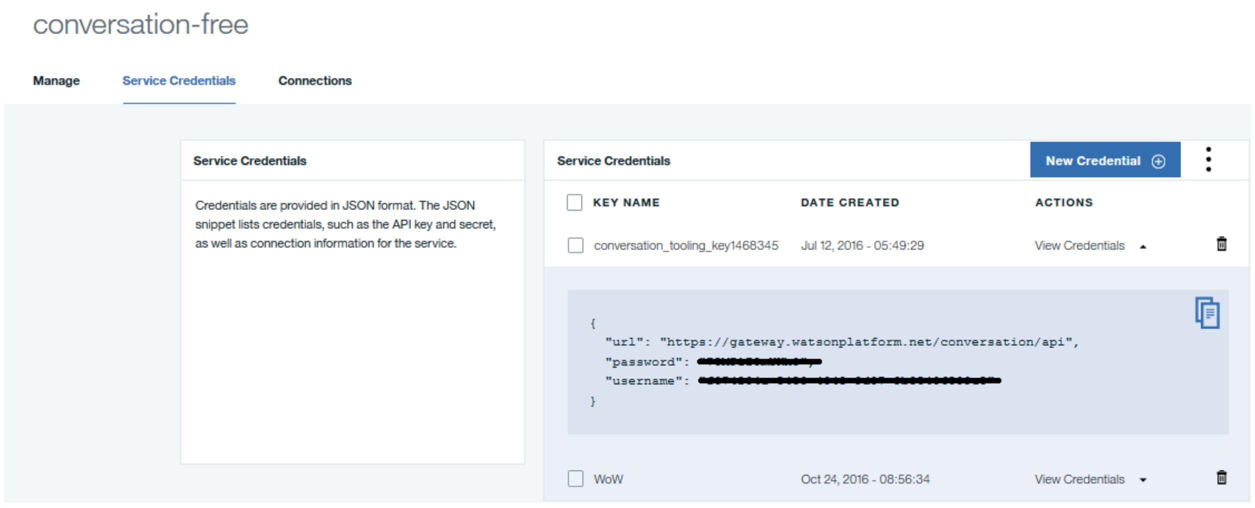
iv. There are three variables which need to be populated:

WORKSPACE_ID

CONVERSATION_USERNAME

CONVERSATION_PASSWORD

v. To get this information return to Bluemix (in your web browser), navigate to your workspace, and click on the **Service Credentials** tab:



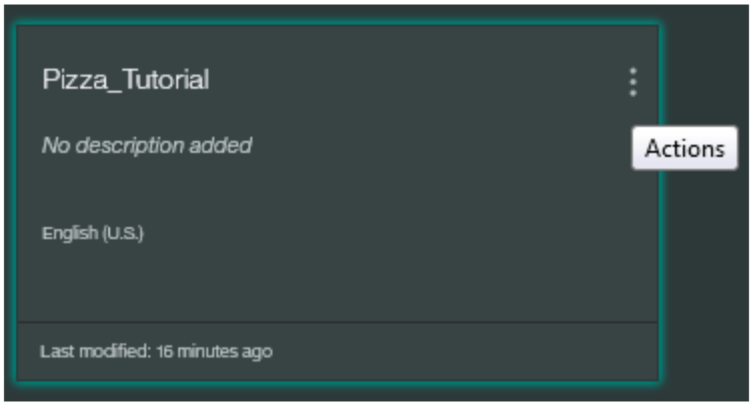
Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

KEY NAME	DATE CREATED	ACTIONS
conversation_tooling_key1468345	Jul 12, 2016 - 05:49:29	View Credentials
WoW	Oct 24, 2016 - 08:56:34	View Credentials

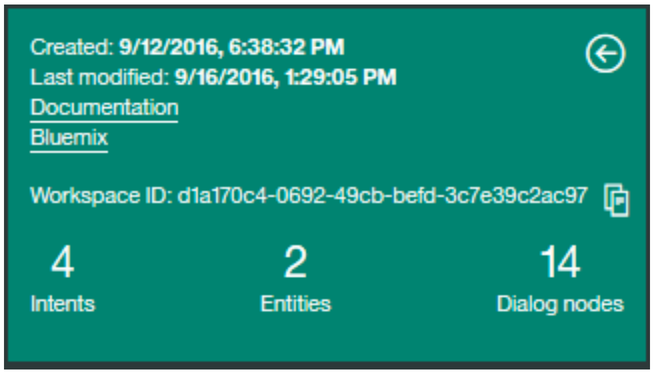
```
{
  "url": "https://gateway.watsonplatform.net/conversation/api",
  "password": [redacted]
  "username": [redacted]
}
```

vi. Click on the **View Credentials** link to expand the credentials section. This will display the username, password, and url for the service instance. Copy these and set them as the values of CONVERSATION_USERNAME and CONVERSATION_PASSWORD within the *.env* file.

vii. Return to the Tooling for the *Order_pizza* workspace.



viii. Click on the workspace menu, and select the **View details** menu item. Once the workspace details are displayed copy the workspace id:



Created: 9/12/2016, 6:38:32 PM
Last modified: 9/16/2016, 1:29:05 PM
Documentation
Bluemix

Workspace ID: d1a170c4-0692-49cb-befd-3c7e39c2ac97

Paste the workspace id as the WORKSPACE_ID environment variable in the .env file.

Save the .env file.

d. Running the Application locally

i. Open a command prompt and type

ii. cd conversation-simple/

iii. Run the following command from within the conversation-simple directory:

npm install

iv. This command will take a few moments to complete (up to a minute is normal). This command installs all of the modules defined in *package.json* (highlighted in config files figure above).

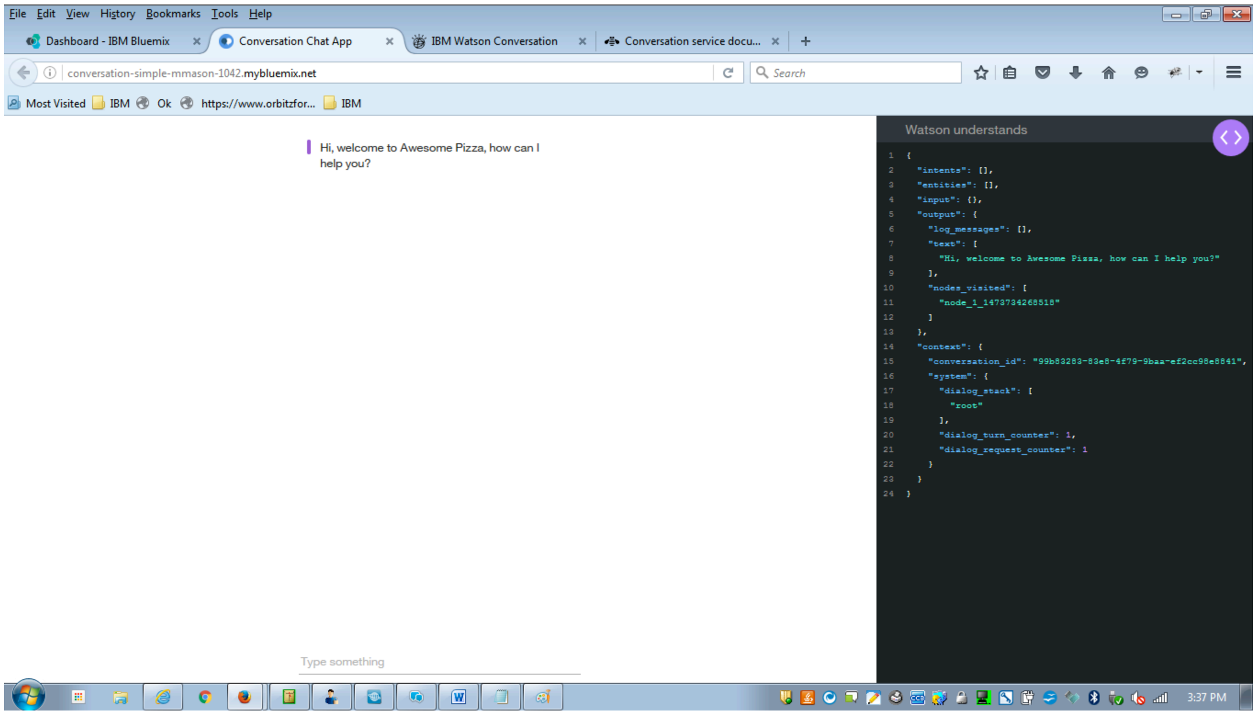
v. Once the operation has completed run the following command:

➢ *node server.js*

vi. This command launches the local application, and if successful you will see the following message in the console

➢ Server running on port: 3000

vii. Navigate to localhost:3000 in your web browser. The application will launch:



Hi, welcome to Awesome Pizza, how can I help you?

Watson understands

```
{
  "intents": [],
  "entities": [],
  "input": {},
  "output": {
    "log_messages": [],
    "text": [
      "Hi, welcome to Awesome Pizza, how can I help you?"
    ],
    "nodes_visited": [
      "node_1_1473734268518"
    ]
  },
  "context": {
    "conversation_id": "99b83283-83e8-4f79-9baa-ef2cc98e8841",
    "system": {
      "dialog_stack": [
        "root"
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1
    }
  }
}
```

e. Asking questions of this application will yield the same results as asking questions via the **Try it out** panel within the tooling.

f. To close out the loop, and verify that the app is working as expected, ask a question in the application (such as "what toppings do you have?" etc), to verify that the system responds as expected.

10. Creating a *multi-turn Dialog*

a. The real power of the Dialog editor comes to bear when creating a more complicated conversation flow that requires multiple "turns" of conversation in order to walk a user through the necessary steps (a *turn* is used to describe a single user input and system response).

   i. In this case, the user wants to order a pizza which requires the user to give a size, and a topping before completing their order.

   ii. As stated previously, a smart dialog requires understanding of context. While the Conversation service does not consider every single thing that has been said, you can teach it what is important to remember for future reference. We use the context object for this. In our pizza use case, we will be storing size and toppings as context.

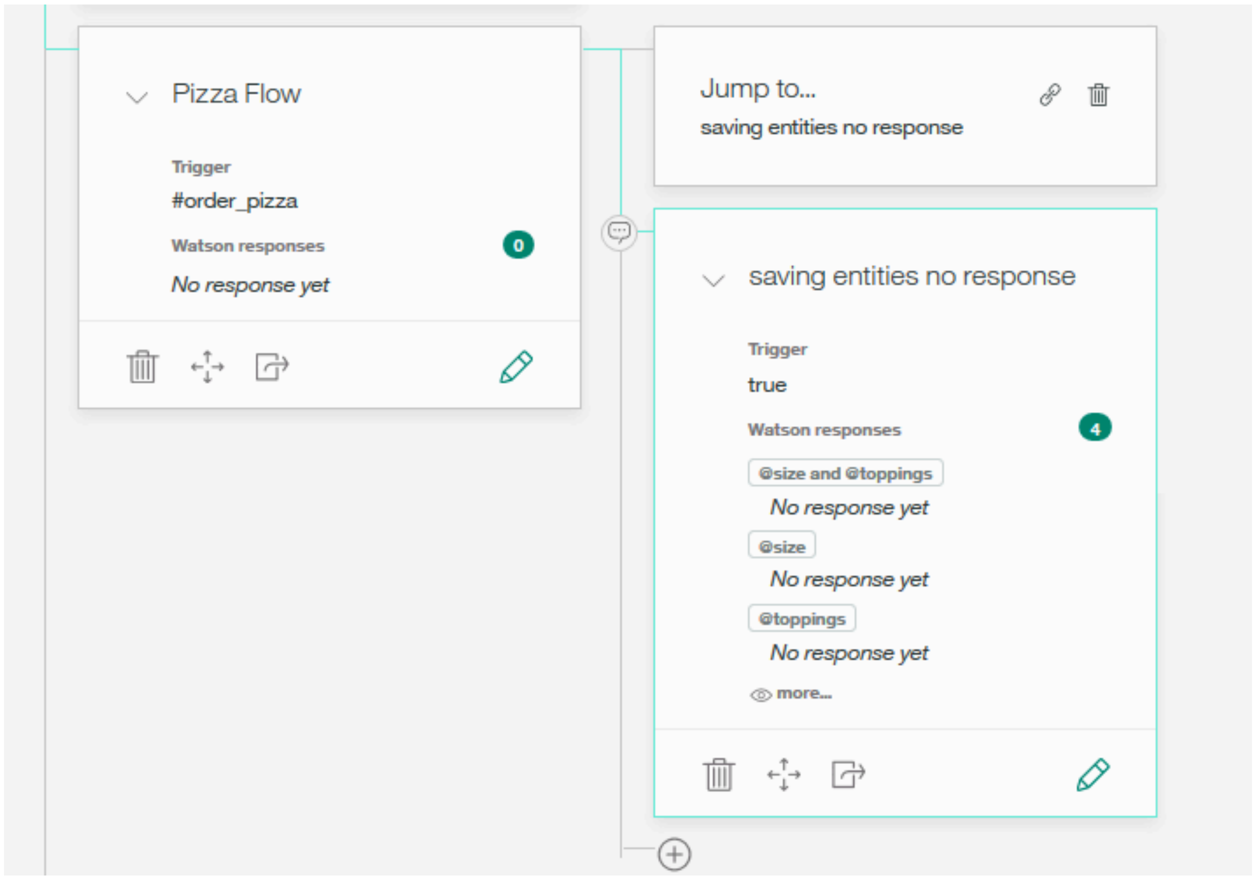
   iii. It could also be used to reference user information from a back-end database by your application injecting data into the context element.

iv. We have also used it to query a database like weather information to display relevant results to users

b. First, we need to create the initial *#order_pizza* node, at 'Root' level (the main section we've been using) for the #order_pizza intent. To do this, click on the plus sign at the bottom of the *#toppings_list* node, and enter *#order_pizza* as the condition. For now, leave the response blank, because we are going to add additional checks

c. Click the plus to the right of the *#order_pizza* node, to create what is called a "child node". Child nodes are only evaluated AFTER the parent node's condition evaluates to true.

i. This first child node will simply be used to extract the entities identified. We will loop back here potentially multiple times until all information is collected and we can move forward. There are likely multiple ways to build this type of interaction; this is one of the simpler ones we have found.



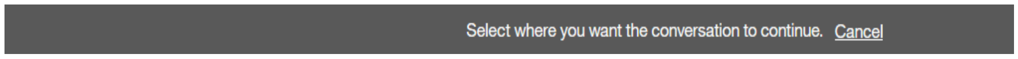
ii. The child node should have a Trigger of 'true' and then the conditional responses will be for the various entities someone might say. : @toppings &&

@size, @size, @toppings, and finally a fourth one left blank in case the user does not say either size or toppings.

d. Next, we need to add a 'Continue from…'

  i. Click the three dots on the #order_pizza node, and click Continue from…

  ii. You will see a message at the top:



Select where you want the conversation to continue. Cancel

  iii. Click the child node, and on the left choose **Go-to Condition.**

    1. This feature directs the system to jump directly to evaluating the conditions instead of waiting for the user to type something else in.

    2. You could also have it go straight to the response, and not even evaluate the condition.

    3. Lastly, you could point it to the speech bubble in between nodes to wait for the User to type something (this is called a 'User Input Node') before evaluating the next conditions, essentially repositioning where the user is in the tree.

    4. By default, the service waits for user input after each dialog node, regardless of whether or not the dialog node outputs text. However, in our case, we don't want to wait for user input, we just want to go straight to the next layer in the tree so we can determine if the user has provided enough information to proceed with the order, or if we will need to prompt for more info (i.e. size and/or toppings) Note that the service evaluates conditions from top to bottom. So we put the node with the most selective conditions at the top, and work our way down to 'true' at the bottom, which will always evaluate to true.

  iv. Next, we need to go into the 'Advanced response' view to save things as context. Click the **{…}** symbol to open the advanced view. This is where we can edit the response in JSON format to do nearly anything we can think of. In this case, we will just be editing context variables.

    1. When editing in the 'Advanced response' view keep your eye out at the bottom of the box for a red line indicating that you have invalid JSON format. Typically this means you need a few brackets or commas somewhere. For now, you can type (or copy/paste if possible) from here.

a. For the response for @toppings && @size you should have

```
{
  "context": {
    "size": "@size",
    "toppings": "@toppings"
  },
  "output": {}
}
```

b. For the response for @size you should have

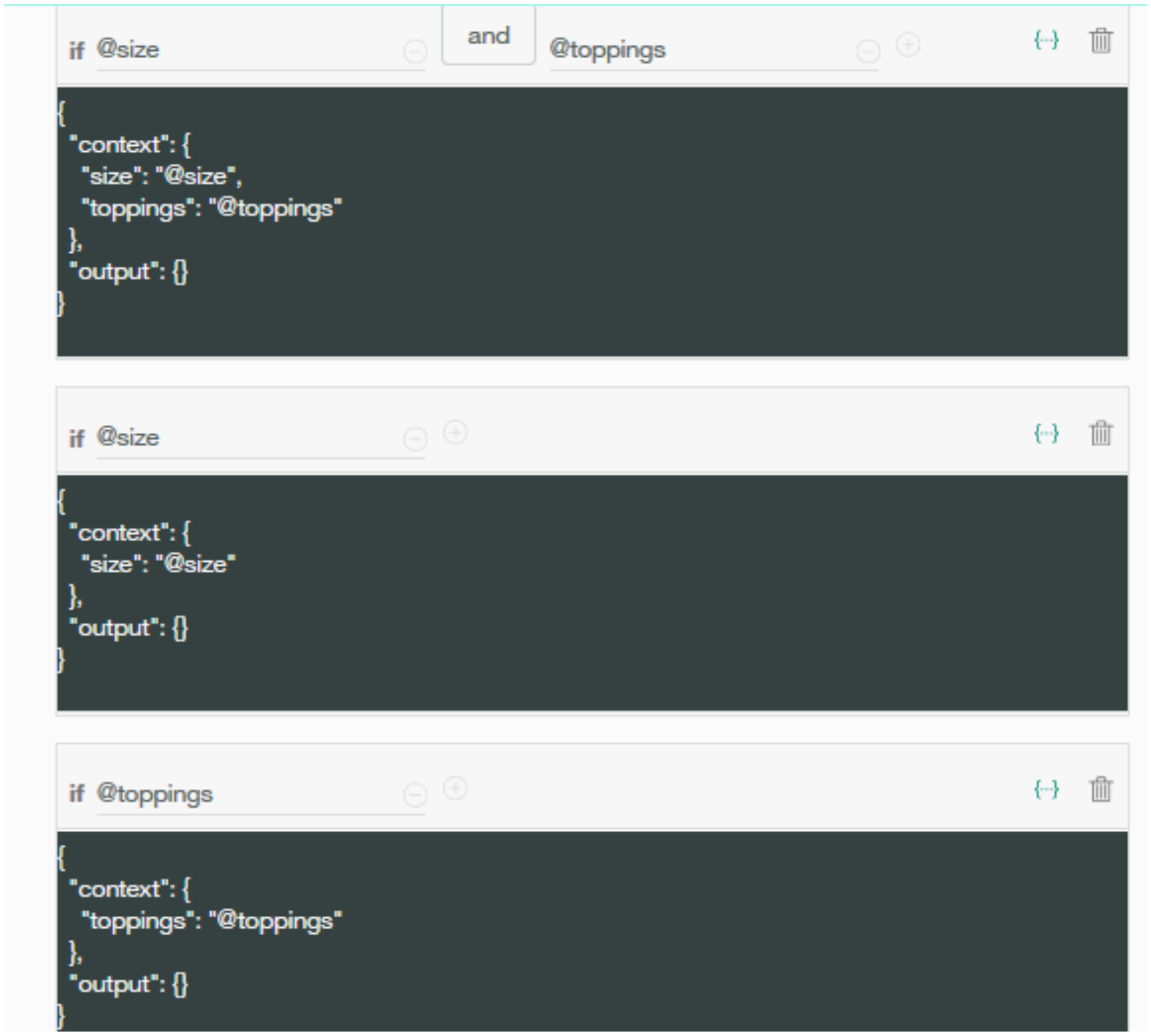
```
{
  "context": {
    "size": "@size"
  },
  "output": {}
}
```

c. For the response for @toppings:

```
{
  "context": {
    "toppings": "@toppings"
  },
  "output": {}
}
```

d. And the final response for no entities needs no response and no context.

v. Again, at this point all we are doing is using this as a return point for extracting any entities.



```
{
 "context": {
  "size": "@size",
  "toppings": "@toppings"
 },
 "output": {}
}
```

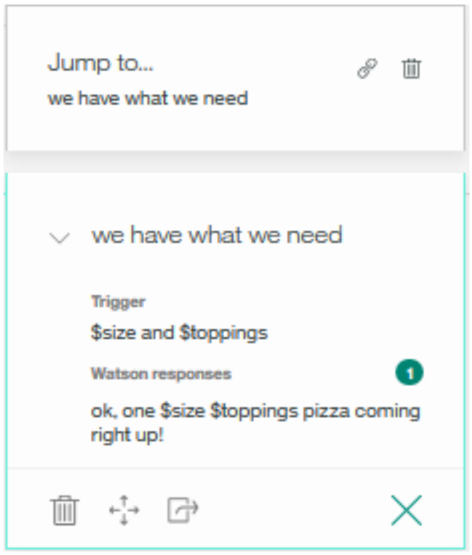
```
{
 "context": {
  "size": "@size"
 },
 "output": {}
}
```

```
{
 "context": {
  "toppings": "@toppings"
 },
 "output": {}
}
```

vi. Next, we will create two child nodes. One for when all information is complete and we want to move on, and a second for when we need to give the correct prompt and loop back.

1. Create a child node with a condition of **$size && $toppings** and give a response indicating we are all finished taking their order. “OK, one $size $toppings pizza coming right up!” This will dynamically place your context variables into your output to assure the user we understood correctly.

2. Since no response was given previously, the user would not know to type anything new still, so we will want to once again **Jump To…** the **Condition** for this newly created child node.


Jump to...
we have what we need

we have what we need

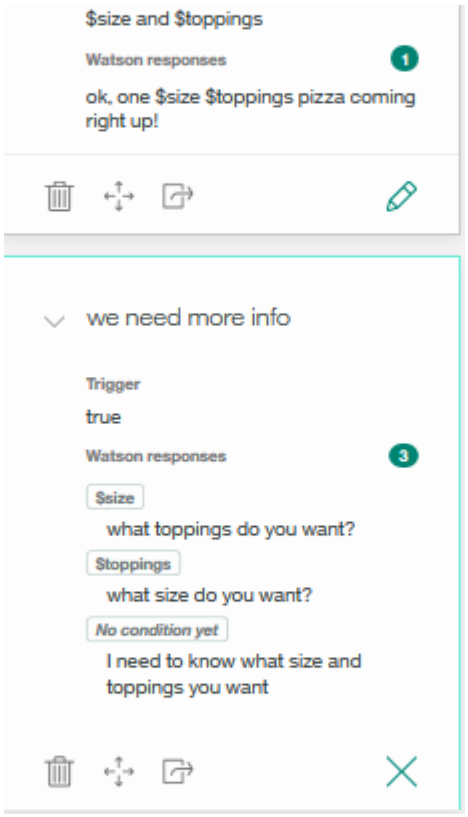
Trigger
$size and $toppings

Watson responses
ok, one $size $toppings pizza coming right up!

3. Next, we will create a node with a few prompts to request only the specific information we have left to gather.

    a. We will again use a main Trigger of ‘true’ and use the Conditional Responses to respond appropriately.

    b. There are three scenarios where we may need to loop back. We still need @size, we still need @toppings, we still need both. Let’s create that node now as a sibling (same column) as the node for all done.



$size and $toppings

Watson responses

ok, one $size $toppings pizza coming right up!

we need more info

Trigger

true

Watson responses

$size

what toppings do you want?

$toppings

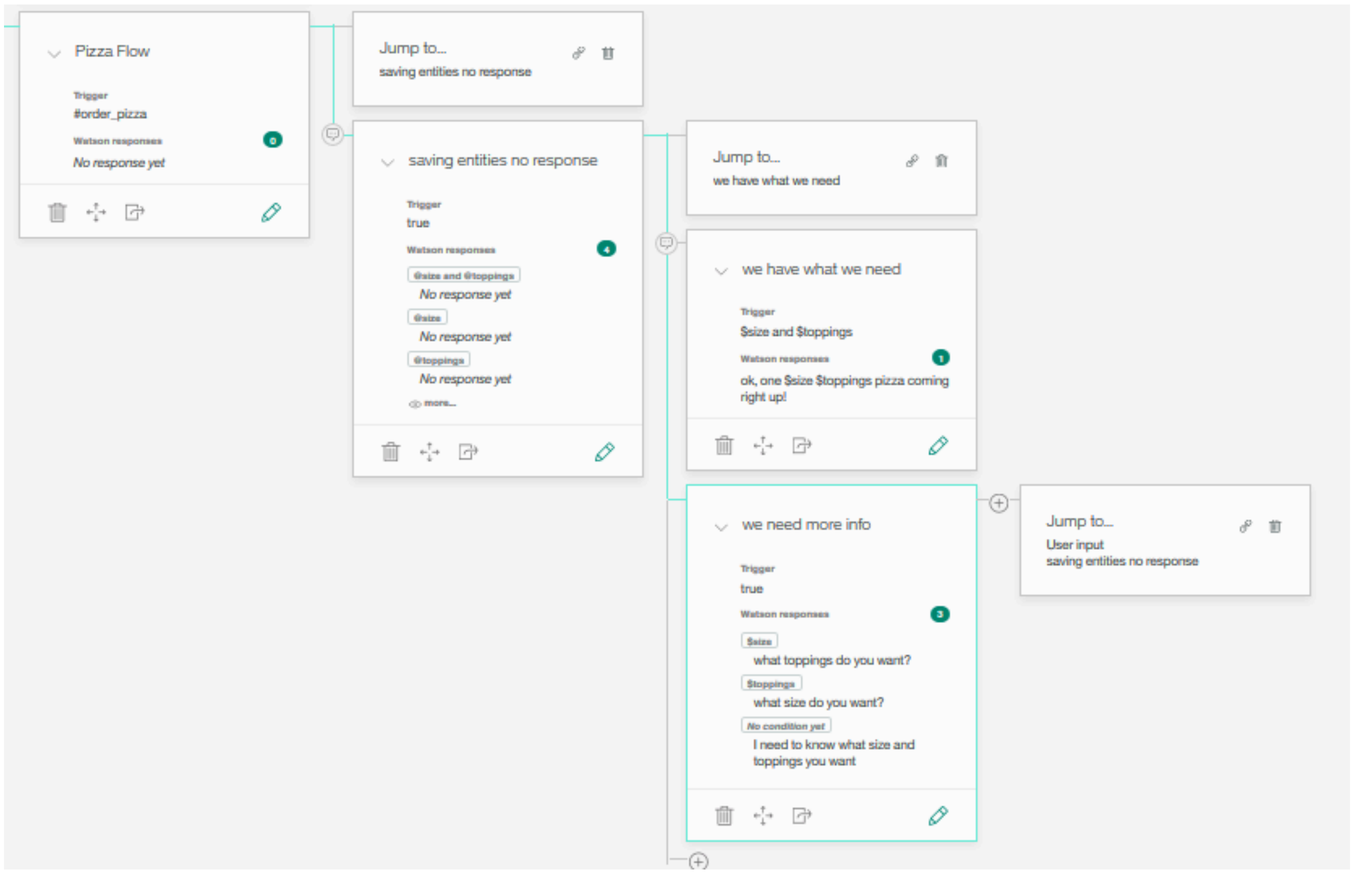
what size do you want?

No condition yet

I need to know what size and toppings you want

c. For the 'all information collected' node we did not need any further children because we were finished. The service will automatically return to root position and be able to start on any top level topic.

d. However, for this 'we need more info' node we will want to return to the point where we collect information again. So we will use a **Jump To…** to the **Condition** of our entity extraction node. Your overall **#order_pizza** flow should look like this:



Pizza Flow
Trigger
#order_pizza
Watson responses 0
No response yet

Jump to...
saving entities no response

saving entities no response
Trigger
true
Watson responses 4
@size and @toppings
No response yet
@size
No response yet
@toppings
No response yet
more...

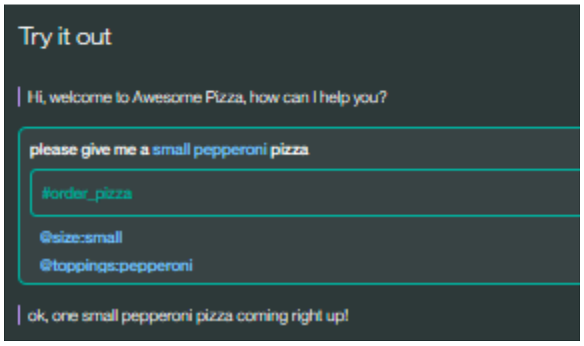
Jump to...
we have what we need

we have what we need
Trigger
$size and $toppings
Watson responses 1
ok, one $size $toppings pizza coming right up!

we need more info
Trigger
true
Watson responses 3
$size
what toppings do you want?
$toppings
what size do you want?
No condition yet
I need to know what size and toppings you want

Jump to...
User input
saving entities no response

And let's test.



Try it out

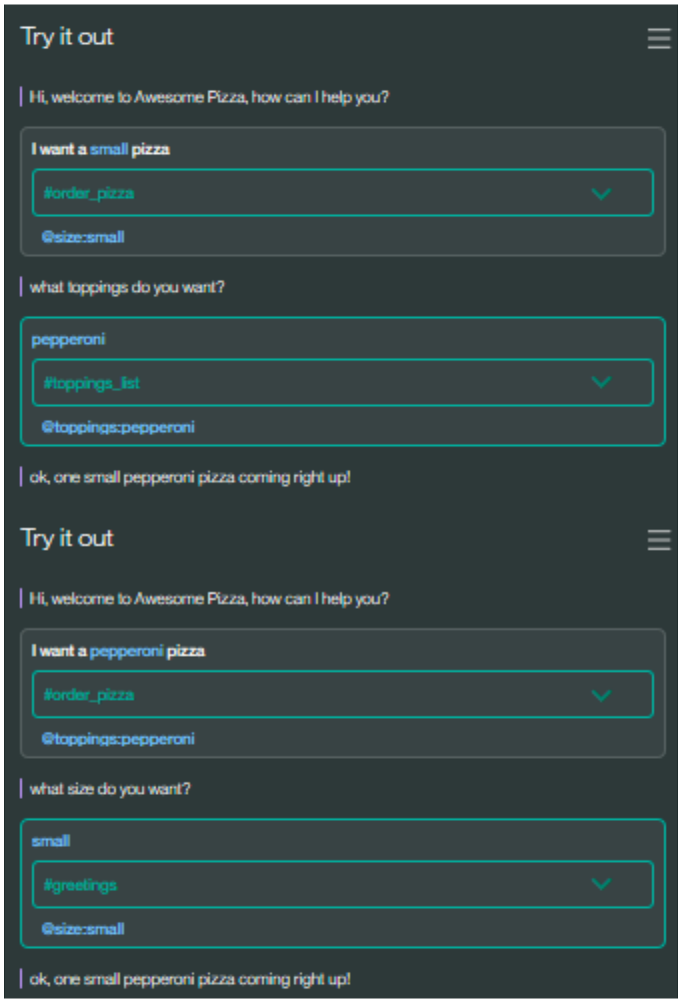
Hi, welcome to Awesome Pizza, how can I help you?

please give me a small pepperoni pizza

#order_pizza

@size:small
@toppings:pepperoni

ok, one small pepperoni pizza coming right up!



Try it out

Hi, welcome to Awesome Pizza, how can I help you?

I want a small pizza

#order_pizza

@size:small

what toppings do you want?

pepperoni

#toppings_list

@toppings:pepperoni

ok, one small pepperoni pizza coming right up!

Try it out

Hi, welcome to Awesome Pizza, how can I help you?

I want a pepperoni pizza

#order_pizza

@toppings:pepperoni

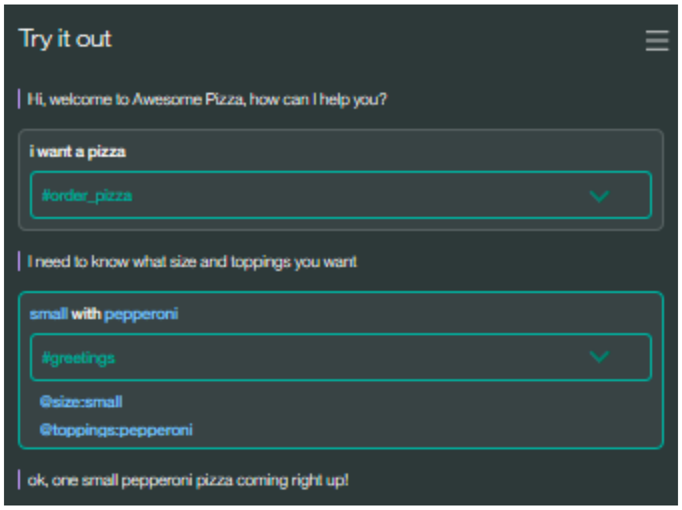
what size do you want?

small

#greetings

@size:small

ok, one small pepperoni pizza coming right up!


Try it out

Hi, welcome to Awesome Pizza, how can I help you?

i want a pizza

#order_pizza

I need to know what size and toppings you want

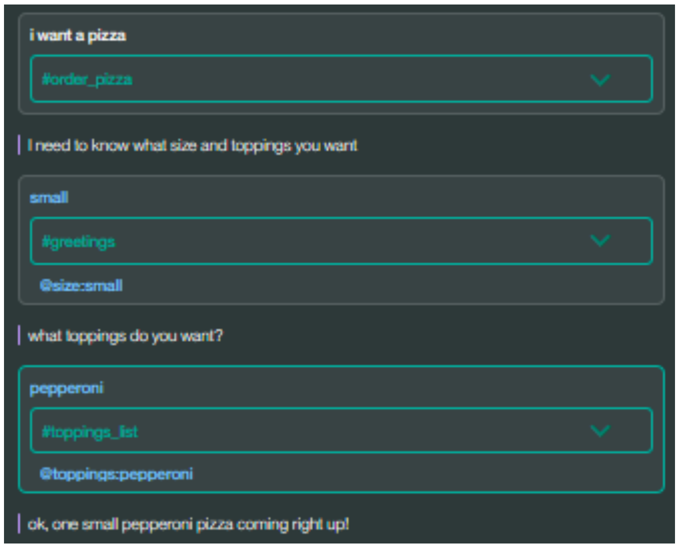
small with pepperoni

#greetings

@size:small

@toppings:pepperoni

ok, one small pepperoni pizza coming right up!


i want a pizza

#order_pizza

I need to know what size and toppings you want

small

#greetings

@size:small

what toppings do you want?

pepperoni

#toppings_list

@toppings:pepperoni

ok, one small pepperoni pizza coming right up!

11. Using Spring Expression Language for multiple entities

a. By now you've noticed we have only been capturing a single topping even when the user says multiple. Entities are captured in an array by the NLU engine, so we actually want to display the entire array in this case.

b. Instead of saving our *toppings* context as @toppings, we will use the Spring Expression of <? entities['toppings'] ?>

   i. The Dialog engine relies on spring expressions for much of its navigation through the tree. We have simplified the common paths using @, #, and $, but sometimes this is not enough and we require a specific Spring Expression.

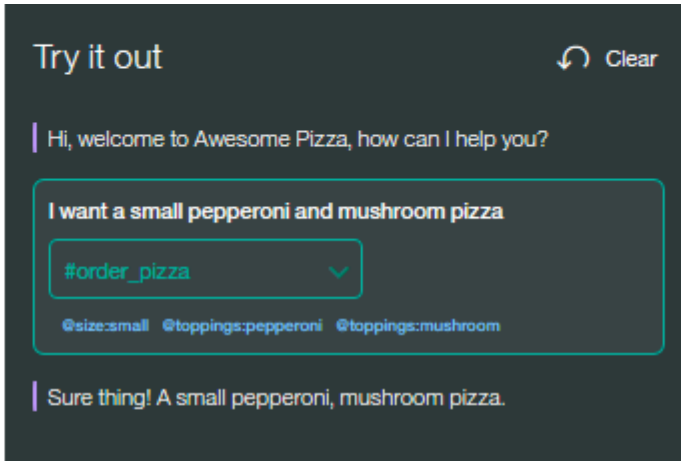
c. In order to display the entire array, we will simply save our $toppings context as

"toppings": "<? entities['toppings'] .toString() ?>"

This saves the toppings context variable as the entire array of entities that were captured, and then formats them as a string, separated by commas.

d. Go through each node, and anywhere we had in our context “toppings”: “@toppings” and switch the @toppings with the expression above. There should be 2 places in the entity extraction node

Now we go back and test.



Hi, welcome to Awesome Pizza, how can I help you?

I want a small pepperoni and mushroom pizza

#order_pizza

@size:small @toppings:pepperoni @toppings:mushroom

Sure thing! A small pepperoni, mushroom pizza.