



{ Lab 7 }

Building Advanced Dialog in Watson Assistant Service



Agenda

BEFORE STARTING	3
1. HANDS-ON PRESENTATION	4
SECTION 1. OBJECTIVES	4
SECTION 2. EXPECTED RESULTS	4
2. CREATE AND LAUNCH	5
3. BUILD A CHATBOT	8
SECTION 1. ADD INTENTS AND ENTITIES	8
SECTION 2. CREATE THE DIALOG TREE	11

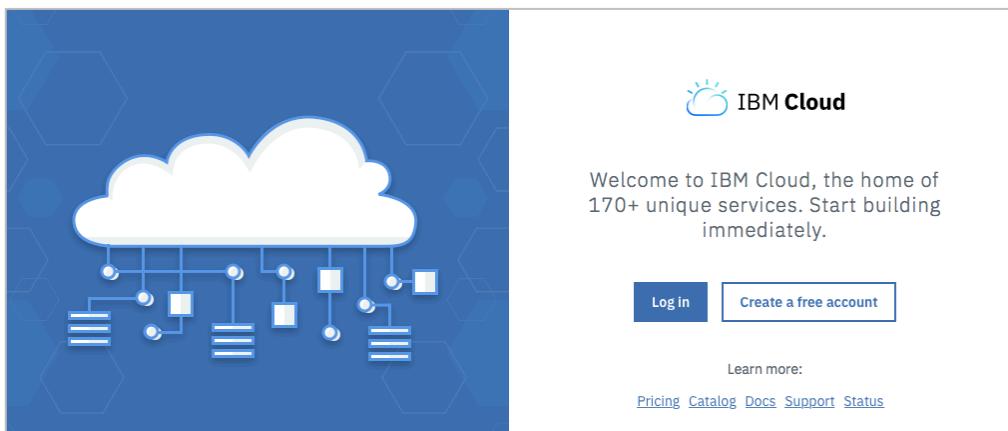
Before Starting



Information

This hands-on required to have an IBM Cloud account. If you don't, you can create one here: <http://bluemix.net/>.

1. Open a browser and access to IBM Cloud: <https://console.ng.bluemix.net>.
Log in.



2. Select organization and space to use during this lab

3. If needed, free resources (GB / #Services) in your IBM Cloud Organization / Spaces to run the lab exercises.
If you encounter a resource contention (Error Message saying you are out of resources), clean up your Spaces by deleting existing Apps or Services.

1. Hands-on presentation

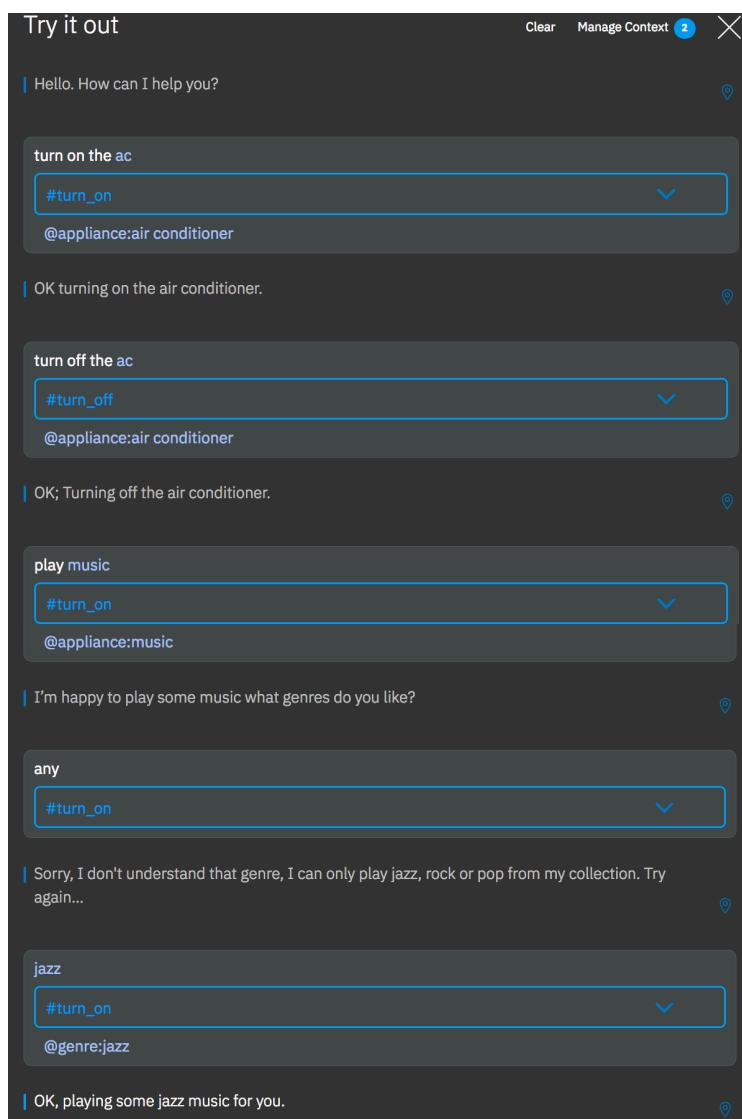
Section 1. Objectives

The purpose of this guide is an instructional approach to working with the IBM Watson Assistant 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 Assistant 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 Assistant service entails the following steps:

- Train Watson to understand your users' input with example utterances: Intents and Examples
- Identify the terms that may vary in your users' input: Entities
- Create the responses to your users' questions: Dialog Builder
- Test and Improve

Section 2. Expected Results

- You created an advanced dialog with IBM Watson Assistant



2. Create and Launch

- 1. In IBM Cloud Catalog, choose “Watson” category

The screenshot shows the IBM Cloud Catalog interface. On the left, there's a sidebar with categories like Storage, Network, Security, Containers, VMware, Platform, and Watson. The Watson category is highlighted with an orange box. The main area displays cognitive services: Watson Assistant (formerly Conversation), Discovery, Knowledge Catalog, Knowledge Studio, Language Translator, and Machine Learning. Each service has a brief description, a small icon, and two plan options: Lite (purple) and IBM (blue). A search bar and a filter button are at the top right.

- 2. Click on **Watson Assistant (formerly Conversation)** & create an instance:
Fill in the App Name & select the *Lite Plan*.

The screenshot shows the creation page for the Watson Assistant service. The "Service name:" field is filled with "MyFirstWatsonAssistant-xu" and is highlighted with an orange box. Below it, there are dropdown menus for "Choose a region/location to deploy in:" (United Kingdom), "Choose an organization:" (Laure.Marchal@ibm.com), and "Choose a space:" (dev). To the left, there's a detailed description of the Watson Assistant service. At the bottom, there are links for "View Docs" and "Terms", a "Create" button, and sections for "Need Help?", "Estimate Monthly Cost", and "Cost Calculator".

- 3. Click **Create**.
Wait for the environment to be created.
- 4. Click the **Launch Tool** button.

Watson / MyFirstWatsonAssistant-XU

Location: United Kingdom Org: Laure.Marchal@ibm.com Space: dev

Get started with the service. Plan: free [Upgrade](#)

Launch tool [Getting started tutorial](#) [API reference](#)

Credentials [Show](#) [Configure credentials](#)

```
{
  "url": "https://gateway.watsonplatform.net/assistant/api",
  "username": "*****",
  "password": "*****"
}
```

- 5. Click the **Workspaces** tab.

IBM Watson Assistant

Home Workspaces

Introducing **IBM Watson Assistant**

Watson Conversation is evolving to simplify how you build and scale virtual assistants. [See what's new](#)

Three easy steps

Follow these steps to create a virtual assistant.

- | | | |
|---|---|---|
| <p>1 Create intents and entities
Determine what your virtual</p> | <p>2 Build your dialog
Utilize the intents and entities you created, plus context from the</p> | <p>3 Test your dialog
Try out the virtual assistant in the tool to see how it recognizes the</p> |
|---|---|---|

6. Click **Create** a new workspace, name it: "MyCarDemo" and click **Create**.

The screenshot shows the 'Workspaces' section of the IBM Watson Assistant interface. A 'Create' button with a '+' icon is highlighted with an orange box. A tooltip box is overlaid on the screen, containing the text: 'Create a new workspace', 'Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application.', and 'You are using 0 of 5 available workspaces in this instance.' Below the tooltip is another 'Create' button with a '+' icon, also highlighted with an orange box.

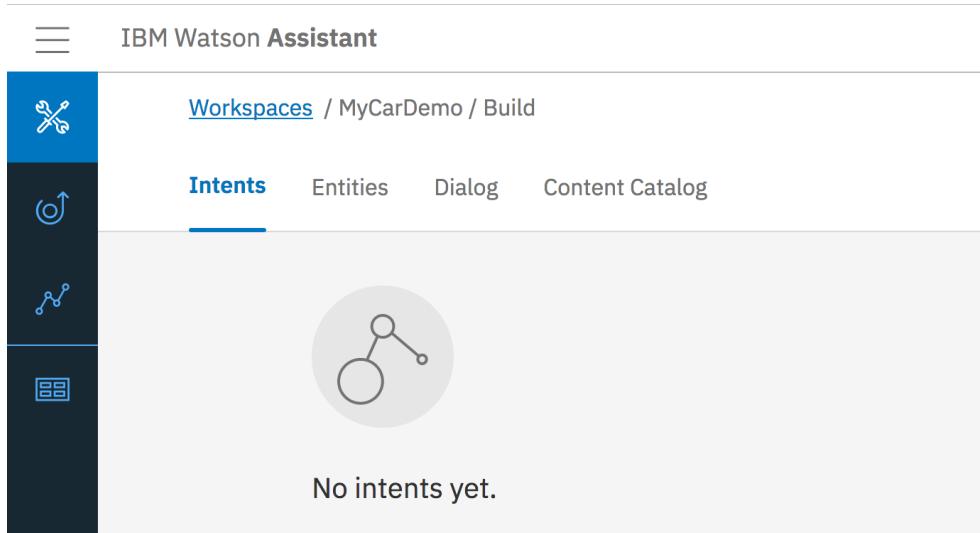
Below this, a modal window titled 'Create a workspace' is displayed. It contains the following fields:

- Name:** MyCarDemo (highlighted with an orange box)
- Description:** (empty field)
- Language:** English (U.S.) (highlighted with an orange box)
- Create** button (highlighted with an orange box)

3. Build a chatbot

Section 1. Add Intents and Entities

- 1. Wait for the workspace to be created, then you start building your chatbot and click **Add Intent**.



- 2. Add a new **Intent**, name it **capabilities** and click **Create Intent**.
- 3. Add a few examples of how you might ask your car what are its capabilities; for example, *help*, *help me*, *how can you help me?*, *what can you do?* *What are your capabilities?* And so forth. Minimum of 5. The more the better. After all, machine learning is about building patterns.
- 4. Click the **blue arrow** to go back to the workspace.

A screenshot of the 'capabilities' intent creation page. The intent name '#capabilities' is at the top, with a blue arrow pointing to the left. Below it is another '#capabilities' entry. A section for 'Description' is present with a placeholder 'Add a description to this intent'. A 'Add user examples' section has a placeholder 'Add user examples to this intent'. An 'Add example' button is highlighted with an orange box. Below this, a list of user examples is shown with checkboxes:

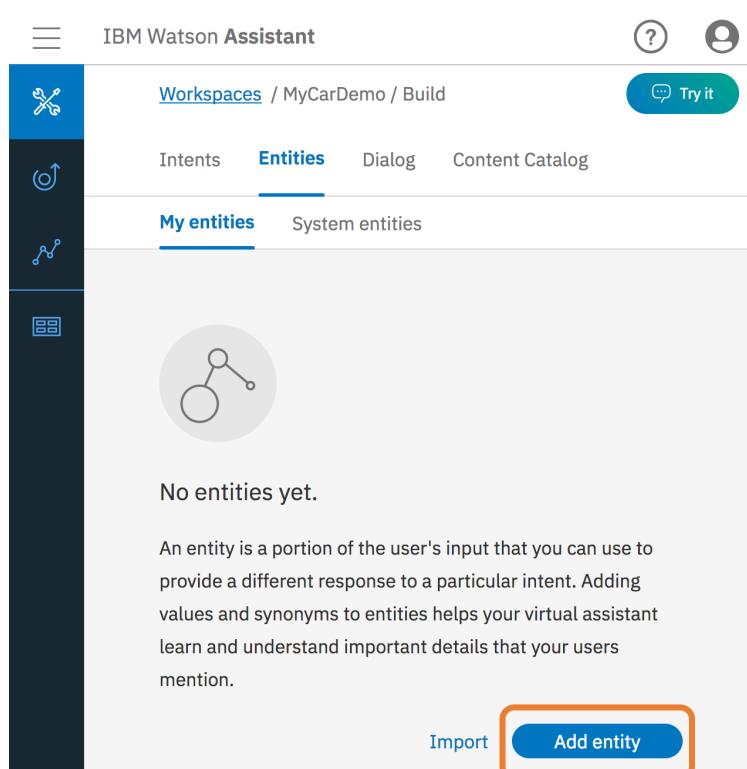
- help
- help me
- how can you help me?

Next you will create two new intents of just what it is that the car can do; for example, turning things on and off is a good place to start.

- ___ 5. Add a **new Intent**, name it **turn_off** and click **create intent**.
- ___ 6. Add a few examples of how you might ask your car to turn off certain things; for example: *off, power off, stop, turn off, quit* and so forth. Minimum of 5. The more the better.
- ___ 7. Click the **blue arrow** to go back to the workspace.
- ___ 8. Add another **Intent**, name it **turn_on** and click **create intent**.
- ___ 9. Add a few examples of how you might ask your car to turn on certain things; for example: *begin, on, play, power on, turn on* and so forth. Minimum of 5. The more the better.
- ___ 10. Click the **blue arrow** to go back to the workspace.

You are now ready to create Entities. In this example, Intents are about “what can the system do”, how can you classify and group together in one bucket similar intended things that the car can do. Entities are about what “specific things” can be acted upon now that the system knows what to do. If the system, knows that your intention is to turn on something, then the entity specifies what are some of the things that you can turn on in a car: heater, engine, air conditioner, music—all sorts of things that you can turn on or off in a car.

- ___ 11. Click the **Entities** tab and click **Add entity**.

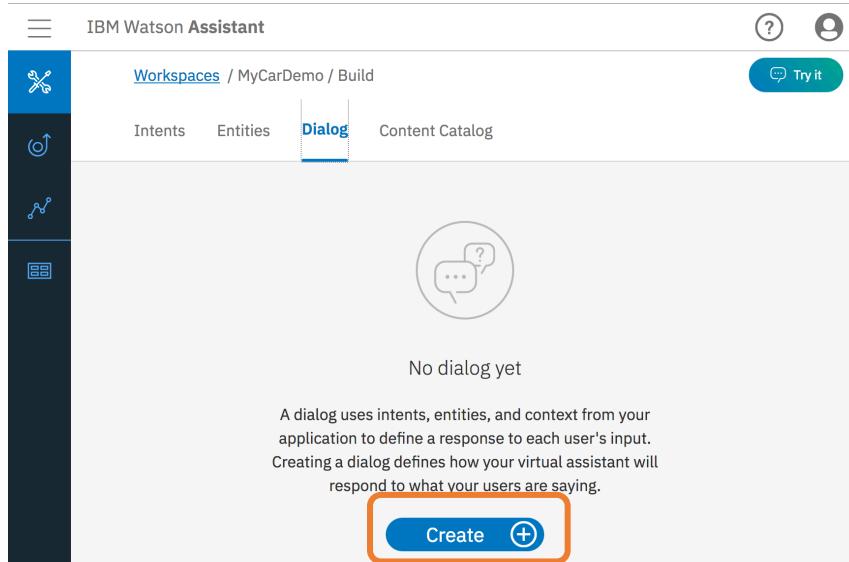


- ___ 12. Type **appliance** and click **Create entity**.
- ___ 13. Add new value of **air conditioner**, plus synonyms such as *ac, cooler, fan* and click **Add value**.
- ___ 14. Add another value of **heater** and add synonyms such as *heat, hot air* and so forth. Click **Add value**.
- ___ 15. Repeat the above steps and add another value of **music** with synonyms such as *radio, song, songs, track* and so forth. Click **Add value**.
- ___ 16. Click the **blue arrow** to go back to the workspace.

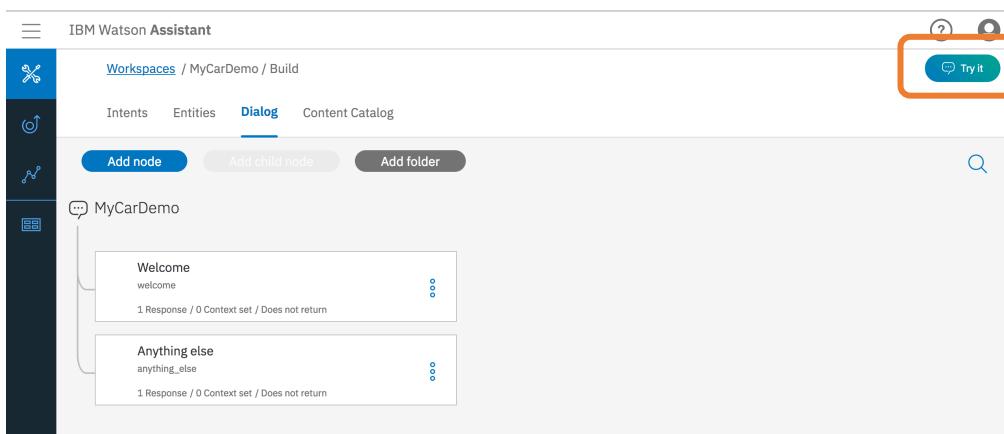
The screenshot shows the IBM Watson Assistant workspace interface. On the left is a sidebar with icons for Home, Entities, Intents, Stories, and Rules. The main area has a header with a back arrow, the entity name '@appliance', and a 'Try it' button. Below the header, there's a 'Last modified a few seconds ago' message, download, delete, and search buttons. A 'Fuzzy Matching BETA' toggle switch is set to 'Off'. The entity details section shows the entity name '@appliance' and its values. The 'Value name' field contains 'music'. Under 'Synonyms', 'radio' and 'track' are listed, with 'song' and 'songs' added below them. An 'Add synonym...' input field and a '+' button are also present. An 'Add value' button is highlighted with an orange box. Below this, an 'Entity values (2)' section lists 'air conditioner' and 'heater', each with a 'Type' column showing 'Synonyms' and their respective synonyms ('ac, cooler, fan' and 'heat, hot air').

Section 2. Create the dialog tree

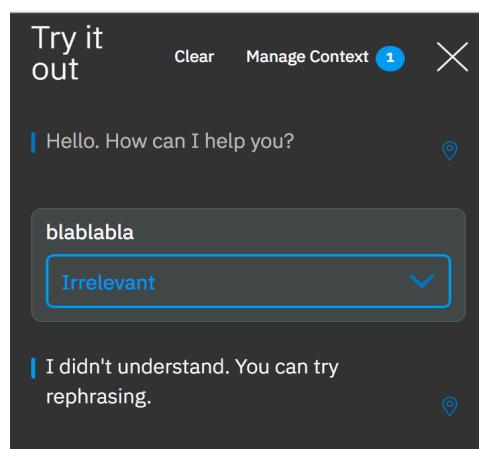
- 17. You are now ready to create the Dialog tree. Click the **Dialog** tab.
- 18. To create a new dialog tree, click the **Create** button.



- 19. You can now see two dialog nodes. They are the two default ones. Click on **Try it**, a new panel will appear on the right side. This is the dialog window that allows you to test your dialog tree.



- 20. The welcome node is executed and the chatbot says “Hello. How can I help you?”. Just type a bunch of letters to see what happens.



So not much happens, because you do not have any other dialog nodes. But the Anything else box opens. That catches all nodes and is always set to true in case the input is not understood.

The nodes have a specific structure, which you can see by clicking on them. The top third is merely a Boolean condition: true or false. The next is the response that has a simple and an advanced view (where you can include JSON objects to add dynamic output and contextual variables for multi-turn conversations). And the last is what the chatbot must do after its response.

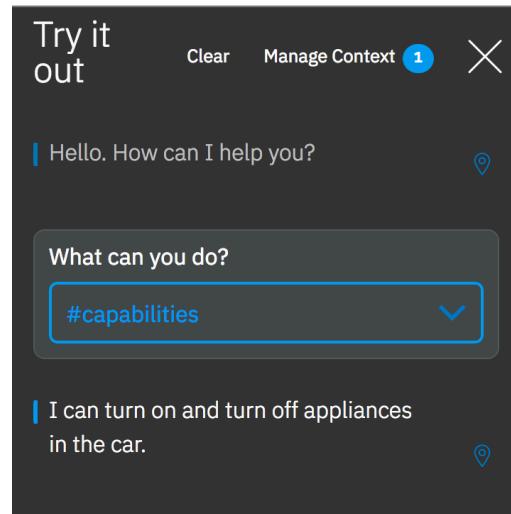
The screenshot shows the IBM Watson Assistant interface in the 'Dialog' tab. A dialog node named 'Welcome' is selected. The 'If bot recognizes:' section contains the intent 'welcome'. The 'Then respond with:' section contains a single response: 'Hello. How can I help you?'. Below it is a link to 'Add a variation to this response'. The 'And finally' section is currently empty. The sidebar on the left shows other nodes like 'Anything else' and 'MyCarDemo'.

Ok, so let's add more dialog nodes. You will now add sibling nodes (sits underneath, different conversation) and children nodes (sits underneath the parent but with indentation).

- ___ 21. Click the **Add node** button to create a sibling node after the Welcome. Type **#capabilities** (# to specify an intent). If you just type **#cap...** the rest appears in the drop down and select the option corresponding.
- ___ 22. In the response section, type: **I can turn on and turn off appliances in the car.**

The screenshot shows the IBM Watson Assistant interface in the 'Dialog' tab. A dialog node named 'Welcome' is selected. A new child node named '#capabilities' has been added below it. The 'If bot recognizes:' section for '#capabilities' contains the intent '#capabilities'. The 'Then respond with:' section for '#capabilities' contains the response 'I can turn on and turn off appliances in the car.'. Below it is a link to 'Add a variation to this response'. The 'And finally' section is currently empty. The sidebar on the left shows other nodes like 'Anything else' and 'MyCarDemo'.

- ___ 23. Click the **Try it button** to test your dialog and type **What can you do?**



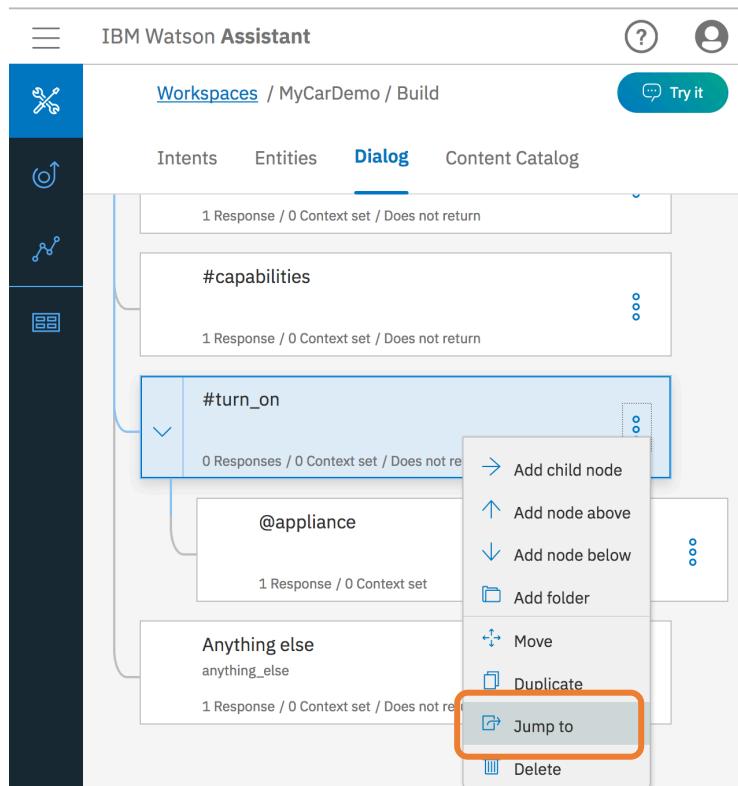
So, the chatbot recognized the intent, the capabilities node had a condition true, so it executed that condition and hence the response that you got.

- ___ 24. Clear the chat box and close it.
 ___ 25. Let's build on this dialog by clicking the **Add node** button and specify the **turn_on** condition.
 ___ 26. Click the **three vertical dots** of the **turn_on** node to **add a child node**, because you want to specify turn on which appliance. Because it might have an Intent but not an Entity, so let's condition it on appliance.

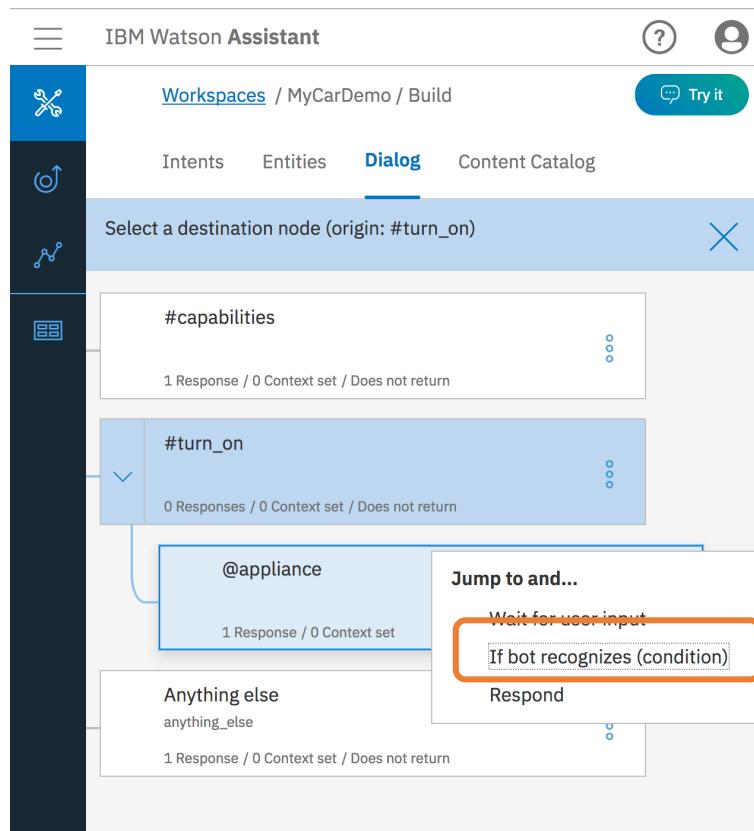
- ___ 27. So, if there is an intent on an appliance, you asked it to turn on something. Type **@appliance** in the condition section, click **any** in the drop-down list, then type in the response section **OK turning on the @appliance** and close the node window.
- ___ 28. Test the dialog and type **turn on the ac**. Notice that it gets the intent and the entity right, but no response.

The reason is that there is a default *wait for user input*. This means, once the *turn_on* node is evaluated for true, then it is going to wait for the user input before moving to the child node *appliance*. You overcome this problem by adding a *Jump to...* This is very useful when you need to move different location in the dialog tree.

- ___ 29. Click the three vertical dots of the *turn_on* node, click **Jump to** and then click on the *appliance* node.



- 30. So, you either jump to and *wait for the user input* or *respond directly* or a *condition*; click on **If bot recognizes (condition)**.



- 31. Let's test it. Click the **Try it** button and type: **Turn on the ac**. Notice the response comes back as expected.

So, consider this: What will happen if there is no appliance entity? It will evaluate the *turn_on* node, then be forced to fall back to the root node and if nothing else matches the *turn_on* condition, then the *Anything else* node will capture it.

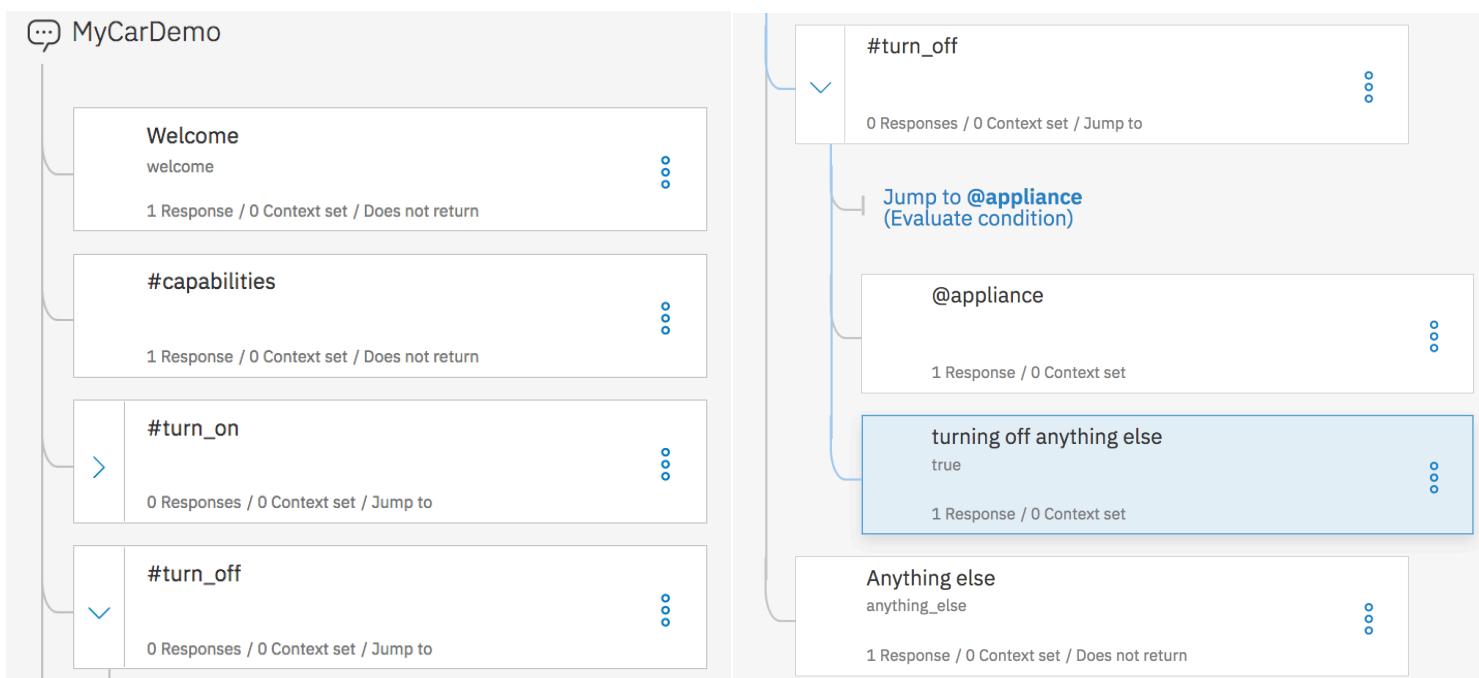
So, let's add a default dialog of our own as a sibling to the *appliance* node, such that if nothing matches, it will capture it with better relevance and not go back to the root.

- __ 32. Add a sibling node to the *appliance* node, name it “turning on anything else”, set the condition to true (type it in the condition section) and type the following in the response section: **I understood your intent was to turn on something. Try saying turn on the ac.**

This is just some text that the car provides when the user does not provide an appliance to do something with.

- __ 33. Create the **turn_off** node underneath the *turn_on* node.

- Create the child node of **@appliance**.
- Ensure it is *jumped from* the *turn_off* node.
- Add a **true** sibling condition node and Name it like the other.
- Copy paste the responses from the *turn_on* node, except have it say **turn off** instead of turn on.



- __ 34. Test the dialog and ask to **turn off the ac**. Expect to see *OK, turning off the air conditioner.*

Now let's ask the user for a preferred genre of music. This is a special case and we want to handle it differently. Let's say that we have jazz, rock and pop as music genre. Let's go back to Entities.

- ___ 35. Click the **Entities** tab.
- ___ 36. Add a new Entity, name it genre and add values to it: **jazz**, **pop** and **rock**; specify some synonyms, for example, **saxo** for jazz.

The screenshot shows the 'genre' entity configuration in the IBM Watson Assistant workspace. The entity name is '@genre'. It has three values: 'jazz', 'pop', and 'rock', all of which are listed as 'Synonyms' with the value 'saxo'.

Entity values (3)	Type
jazz	Synonyms
pop	Synonyms
rock	Synonyms

- ___ 37. Click on the **blue arrow** to go back to the workspace and go back to the **Dialog** tab.
- ___ 38. Click **Try it** to test with the changes. Notice that the chat is training on the new changes. It will turn green shortly.

Yeap, this is neural nets, deep learning and the name of the algorithm is back propagation. The fact that it takes a few seconds to build patterns and use models (including Support Vector Machines) on a distributed scale is cutting edge technology.

Since you know that you want to turn something on and that is an appliance (music), then let's begin from the *turn_on->appliance* node.

- 39. From the *turn_on->appliance* node, add a child node and specify the condition of **@appliance:music**.
- 40. And if the condition music is true, then add another child to specify the condition genre by adding a child node the **@appliance:music** node and type **@genre**.

The left screenshot shows the initial state of the dialog tree. It starts with an intent node labeled '@appliance' which has a condition 'turning on anything else true'. Below it is a '#turn_off' node. A context menu is open over the '@appliance' node, with the 'Add child node' option highlighted by a red box.

The right screenshot shows the result after step 39. The '@appliance' node now has a child node labeled '@appliance:music' with the condition 'turning on anything else true'. This node also has a context menu with 'Add child node' highlighted.

- 41. Add the following response to the genre node: **OK, playing some @genre music for you.**

The screenshot shows the dialog tree with the '@appliance:music' node expanded. Inside it is a child node labeled '@genre'. To the right, a configuration panel is open for this node. The 'If bot recognizes:' section contains the condition '@genre' (highlighted by a red box). Below it, the 'Then respond with:' section contains the message '1. Ok, playing some @genre music for you.' (also highlighted by a red box). The 'Wait for user input' dropdown is set to 'v'.

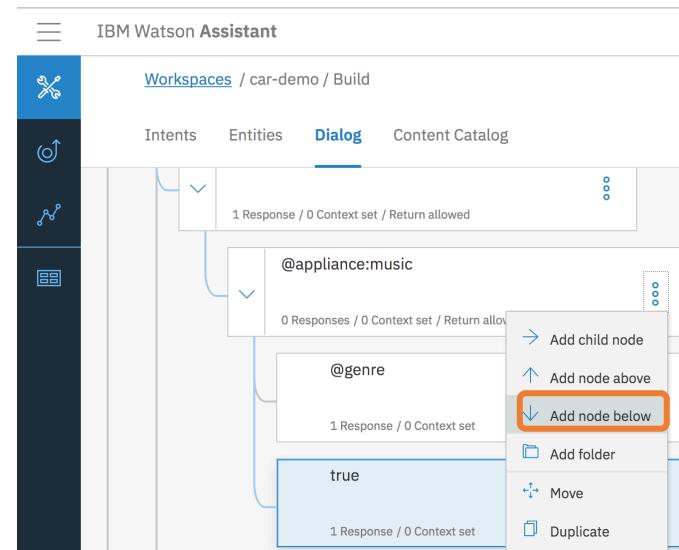
This is just a dummy message, but this where you use some JSON code to interact with the client app and play actual music of a certain genre from Spotify or Amazon Music and so forth.

- 42. Similar to Step 32, if the genre is not understood by the system and the condition is true, then you want to capture a poignant message here.
- 43. Add a sibling node to the *genre* node, set the condition to true (type it in the condition section) and type the following in the response section: **Sorry, I don't understand that genre, I can only play jazz, rock or pop from my collection. Try again...**

The screenshot shows the IBM Watson Assistant interface in the 'Dialog' tab. A context node named '@genre' is selected. A context menu is open to the right of the node, with the 'Add node below' option highlighted by a red box. Other options in the menu include 'Add child node', 'Add node above', 'Add folder', 'Move', and 'Duplicate'.

The screenshot shows the IBM Watson Assistant interface in the 'Dialog' tab. A new node named 'true' has been added as a sibling to the '@genre' node. The 'true' node is currently selected. In the right-hand panel, there are fields for 'Name this node...', 'If bot recognizes:', and 'Then respond with:'. Under 'Then respond with:', there is a list item: '1. Sorry, I don't understand that genre, I can only play jazz, rock or pop from my collection.' Below this, there is a field for 'Add a variation to this response'. At the bottom of the panel, there is a 'Customize' button and a 'X' button.

- 44. Repeat the above step and create another **true** condition for the **appliance:music** node.



This means, that the appliance:music is true and the appliance is not music it reaches the **true** node you just created. Since the intent was to turn on something, (maybe you want to turn on the ac or the heater) the let's move the response from the appliance node to this **true** condition node.

- 45. Cut (not copy) and paste the *response* from the appliance node to the **true** node that is a sibling of the **appliance:music** node. And give it a name like this:

The screenshot shows the IBM Watson Assistant interface with the 'Dialog' tab selected. The 'true' node under '@genre' now has a response 'turning on anything else'. A variation of this response, 'turning on something other than music', is highlighted with an orange box.

- 46. And if the appliance:music is true and the system understood (that is, we asked for the correct genre), then the system response for the appliance music node is: **I'm happy to play some music what genres do you like?**

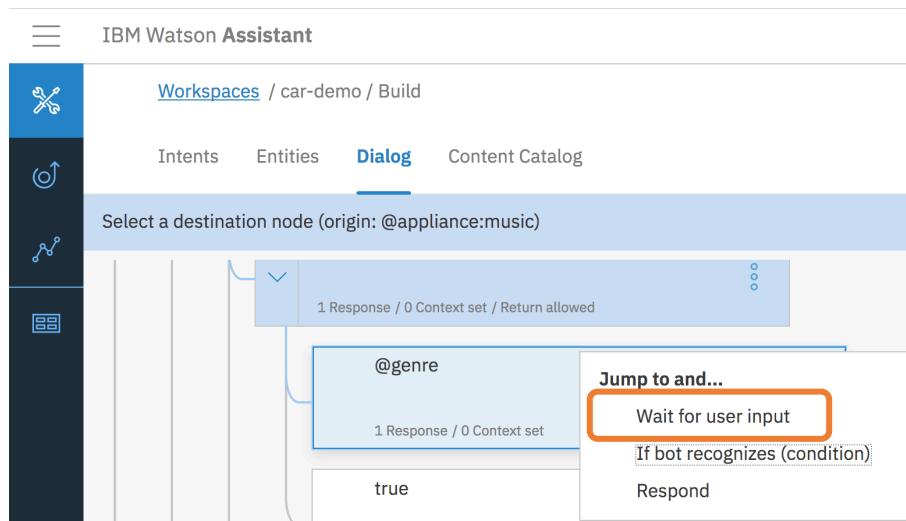
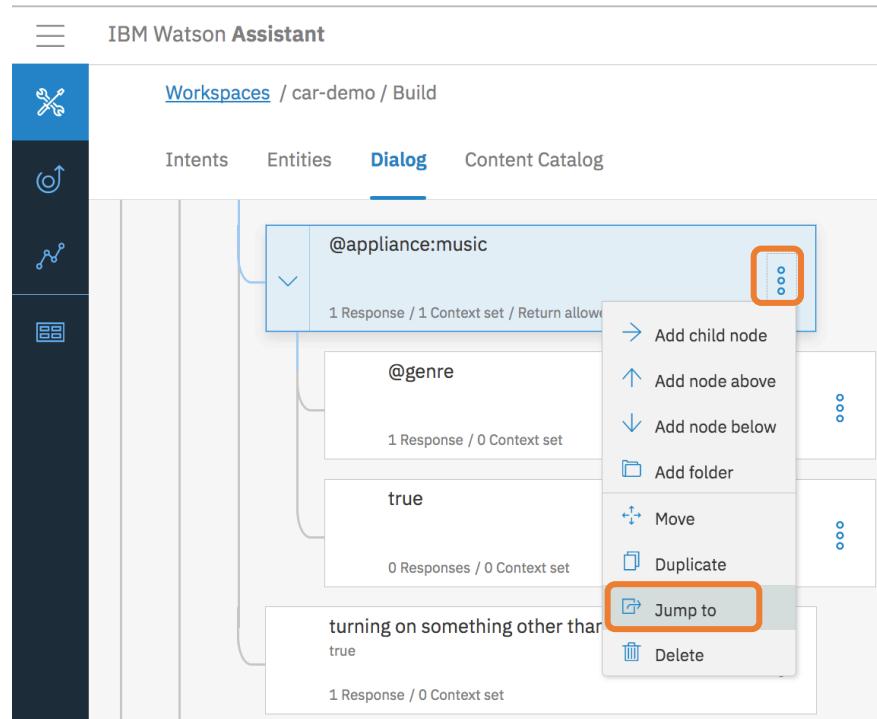
The screenshot shows the IBM Watson Assistant interface in 'Dialog' mode. On the left, a tree view shows nodes: '@appliance' (1 Response / 0 Context set / Return allowed) which branches into '@appliance:music' (1 Response / 0 Context set / Return allowed), '@genre' (1 Response / 0 Context set), and 'true' (1 Response / 0 Context set). The 'true' node is highlighted. To the right, a configuration panel has a text input 'Name this node...' with 'Customize' and 'X' buttons. Below it, 'If bot recognizes:' contains '@appliance:music' with a minus sign and a plus sign. Under 'Then respond with:', there is a list item '1. I'm happy to play some music what genres do you like?' with a minus sign, and a note 'Add a variation to this response'.

- 47. From the appliance node, click the three vertical dots and add the **jump to** the @appliance:music node and click on **condition**.

The left screenshot shows the IBM Watson Assistant interface with a context menu open over the '@appliance' node. The 'Jump to' option is highlighted with a red box. The right screenshot shows the 'Jump to and...' dialog box, which includes fields for 'Wait for user input', 'If bot recognizes (condition)', and 'Respond'.

This is so once the appliance is captured we want to present a prompt to the user. Jump from where you ask?

- 48. Click the three vertical dots from the appliance:music node and select the **jump to** the genre node but click **wait for user input**.



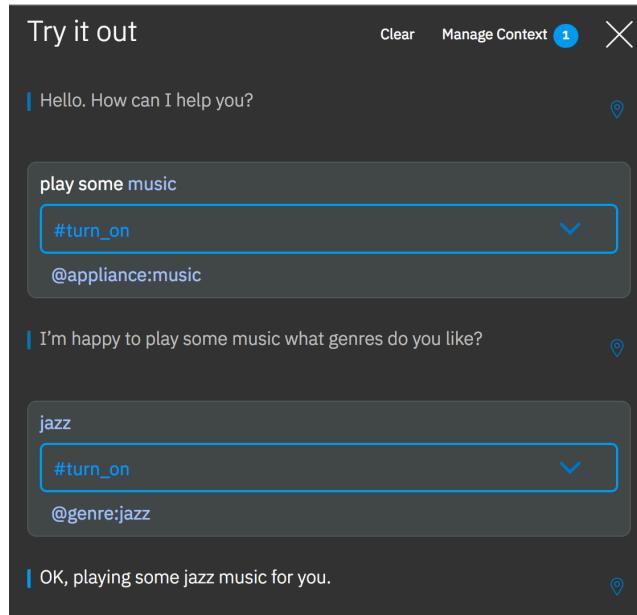
This is because we know there was an appliance and we want to condition on what type of appliance was asked by the user and also capture the answer of the user after the prompt was presented.

Let's test our work.

- ___ 49. Click **Try it** button and ask the system to: **play some music**.

It understood our intent of wanting to turn on an appliance and that appliance is music. It is going to prompt us for specific genre of music.

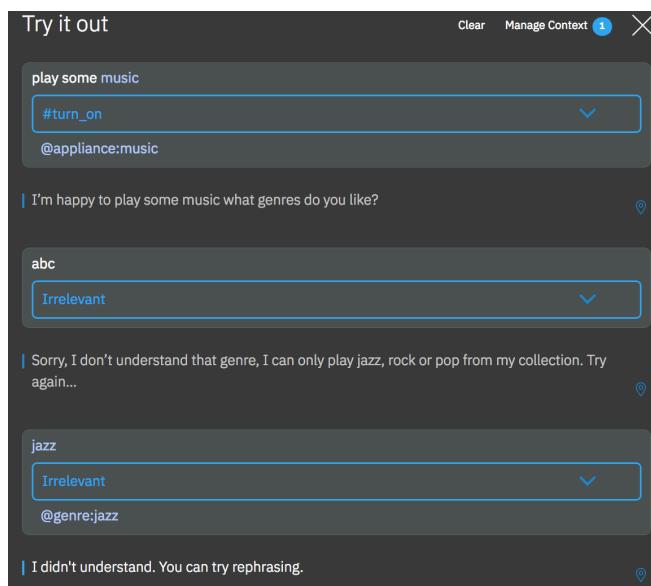
- ___ 50. Type: **jazz**



And so it does, well, at least it says it would.

Let's try a slightly different flow.

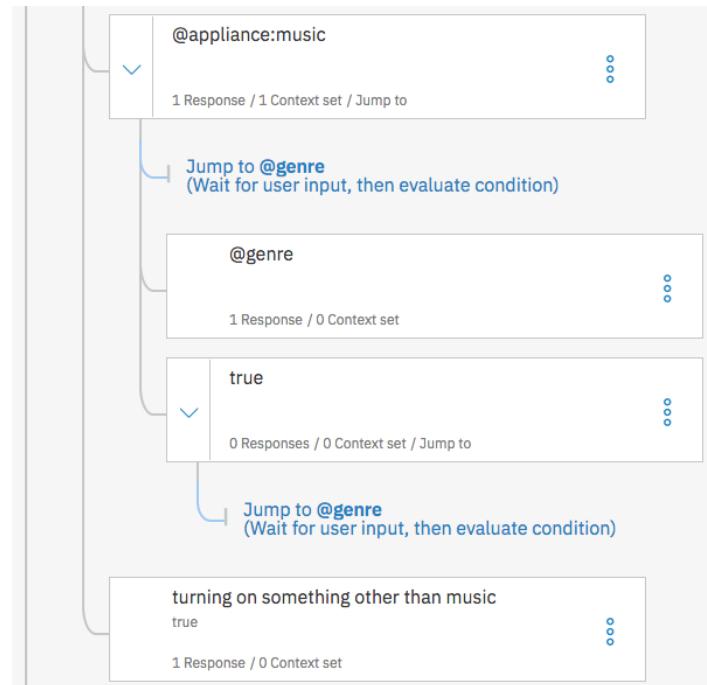
- ___ 51. Type: **play some music**.
- ___ 52. For subsequent response, type: **abc**
- ___ 53. Since *abc* is not in its collection, then type: **jazz**



Notice, it got nothing right and it lost the context. It went to the leaf node genre -> *true* and by default it went back to the root. So, you want to include a **Jump to** (from the *true* leaf node back to the *genre* node) and you want it to continue from a *user input*.

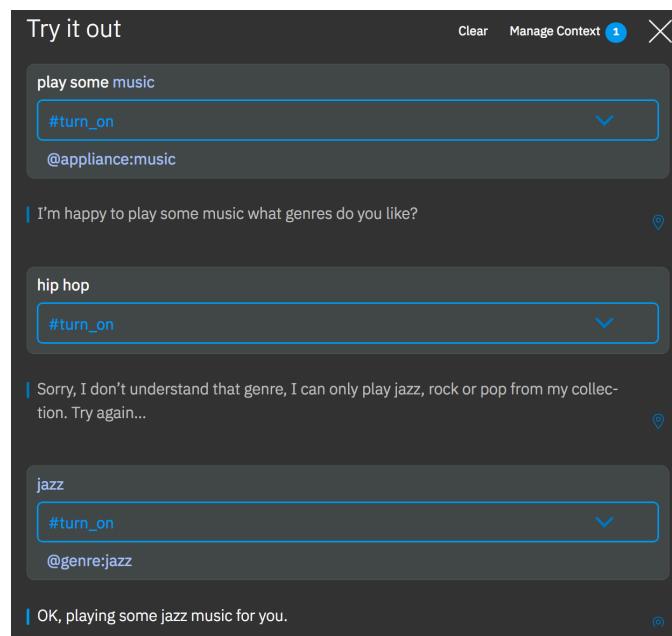
- ___ 54. From the *true* node sibling of *genre* node, click the three vertical dots and select **Jump to**, click on the *genre* node and click **wait for user input**.

You want to continue from a user input. This means that the system has prompted you for user input and the *jump to* user input means that the system wants to come back for additional user input and condition the dialog at this granular level not from the root node.



Test the chat flow with similar dialog:

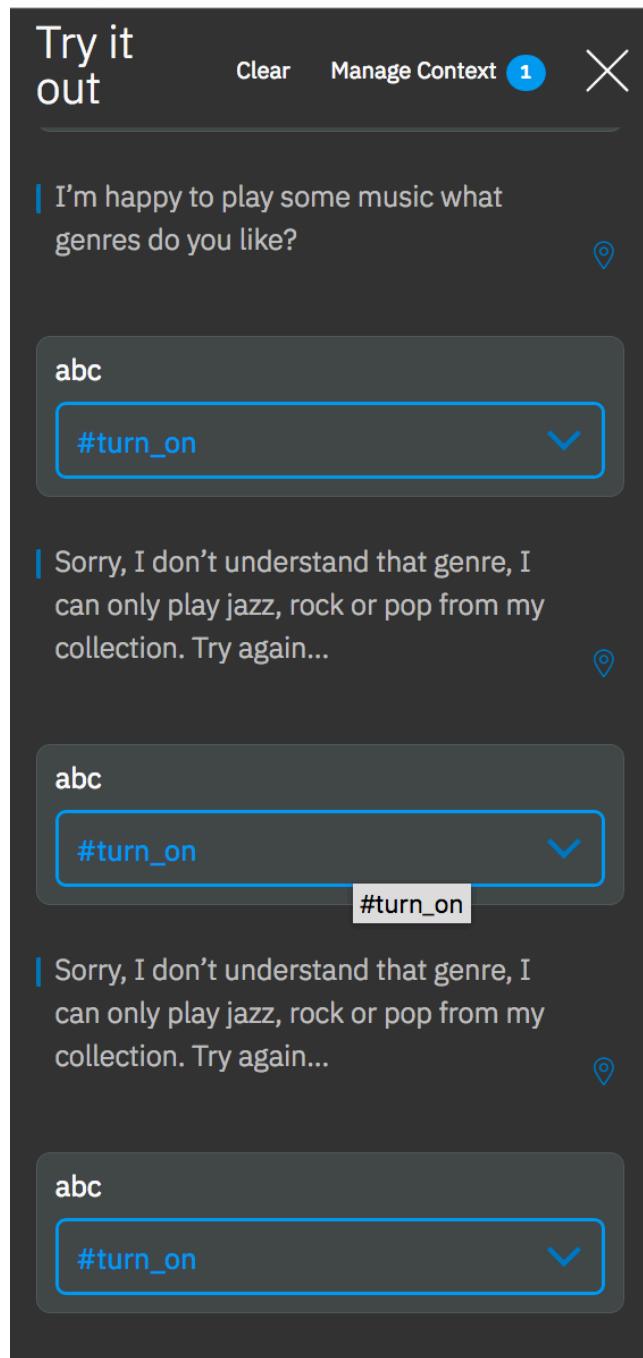
- ___ 55. Type: **play some music**
 ___ 56. Type: **hip hop**
 ___ 57. Type: **jazz**



Notice that it did not lose the context of the conversation and did not go all the way back to the *turn_on* condition (node) and get stuck there.

Now let's explore another user input:

- ___ 58. Clear the chat window
- ___ 59. Type: **play music**
- ___ 60. Type: **abc**
- ___ 61. Again, type: **abc**



Notice that it is stuck in a loop. Let's fix that. The best practice to remove this loop is to add a context variable.

We are going to include a context variable that measures how often does the genre -> **true** node variable gets hit.

We want to initialize the **true** node one level higher before it is called by the condition.

- 62. Click on the *appliance:music* node (this is the one level up from the *true* node to its lower right) to modify it, click the three vertical dots in the response section and select **Open context editor**.

If bot recognizes:

@appliance:music  

Then respond with:

1. I'm happy to play some music what genres do you like?

Add a variation to this response

Open JSON editor

Open context editor

And finally

Jump to...  "Ok, playing some @genre music for you." (User input) 

- 63. Add a context variable in the context editor and instantiate it to 0 :

Then set context:

Variable	Value	More
\$ counter	0	
 Add variable		

And now you must update the context response part in the *true* node.

- 64. Open the *true* node, click on the three vertical dots in the response section and select **Open context editor**.
- 65. Change the condition section of this *true* node to `$counter<1` and edit the context to look like this:

If bot recognizes:

\$counter<1

Then set context:

Variable	Value
\$ counter	"<\$counter+1>"

+ Add variable

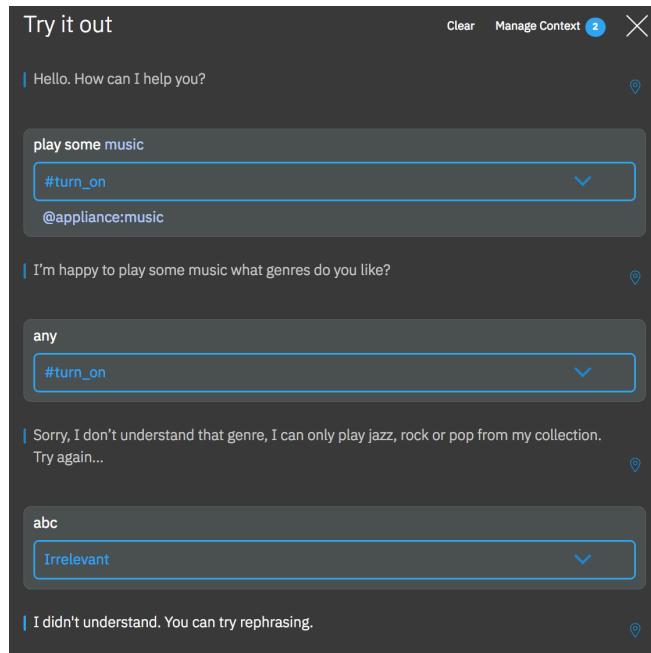
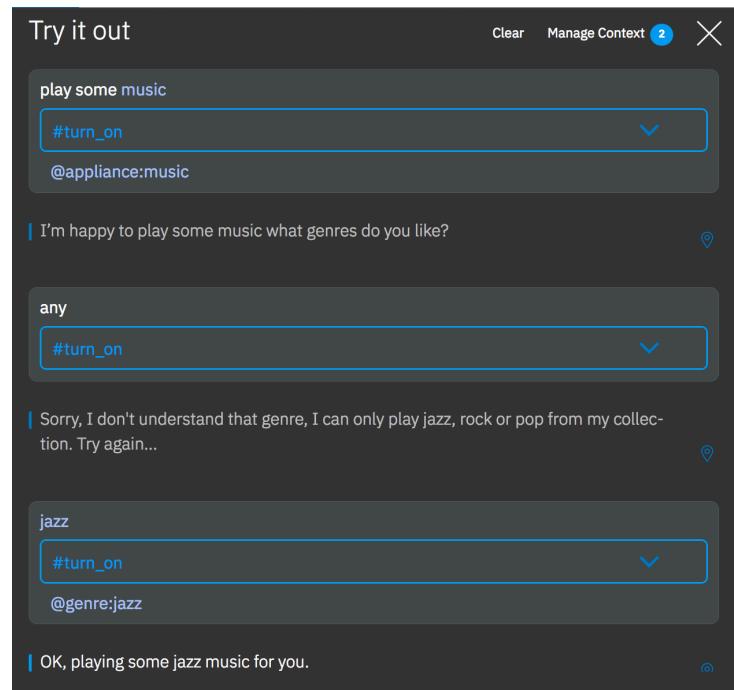
The “`<$counter+1>`” is the syntax inside a JSON editor where you want to evaluate some expression. This means inside the context variable, set the variable to counter plus one.

This means, the first time the condition hits the appliance:music node it will set the variable to zero and the subsequent times it will increment it by one. We want to have a count of how many times we hit the genre node with same “not true” response so it does not get stuck in a loop... it has a count of our hits; less than one means if I type genre music that is not in the collection over and over again, then do not fall for my trap again and again.

Let's test the dialog flow again:

- ___ 66. Type: **play some music**
- ___ 67. Type: **any**
- ___ 68. Type: **jazz**

OK so that was normal operation, now let's make it fall into the loop trap by keep typing irrelevant genre of music.



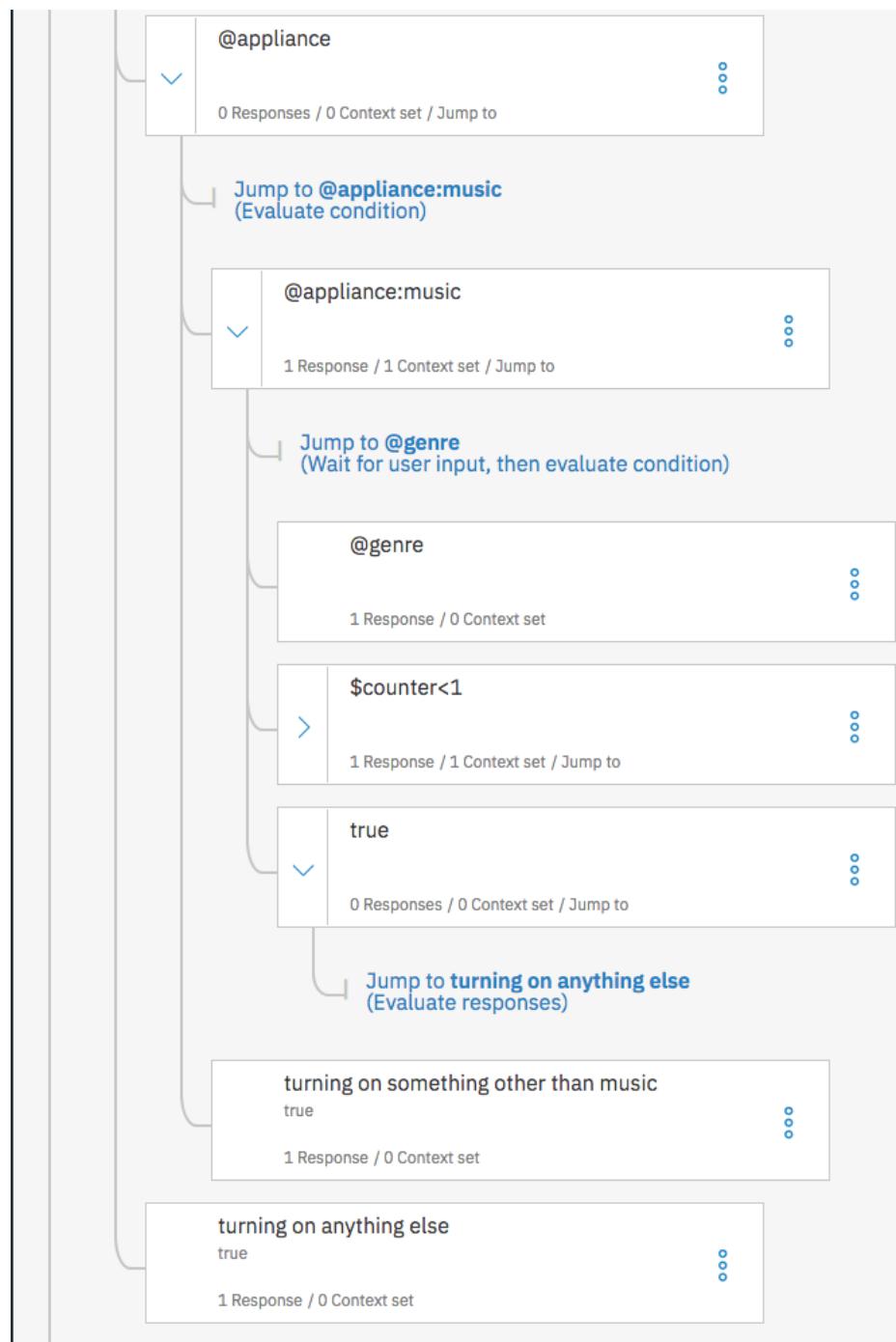
- ___ 69. Clear the chat and type: **play music**

- ___ 70. Type: **any**
- ___ 71. Type: **abc**

Notice that it now goes back to root, but it is not what we want to do. We want to say to the user that we understand that he's trying to turn on something but he's not doing it well, so it should go back to the *true* node, sibling the `@appliance` node.

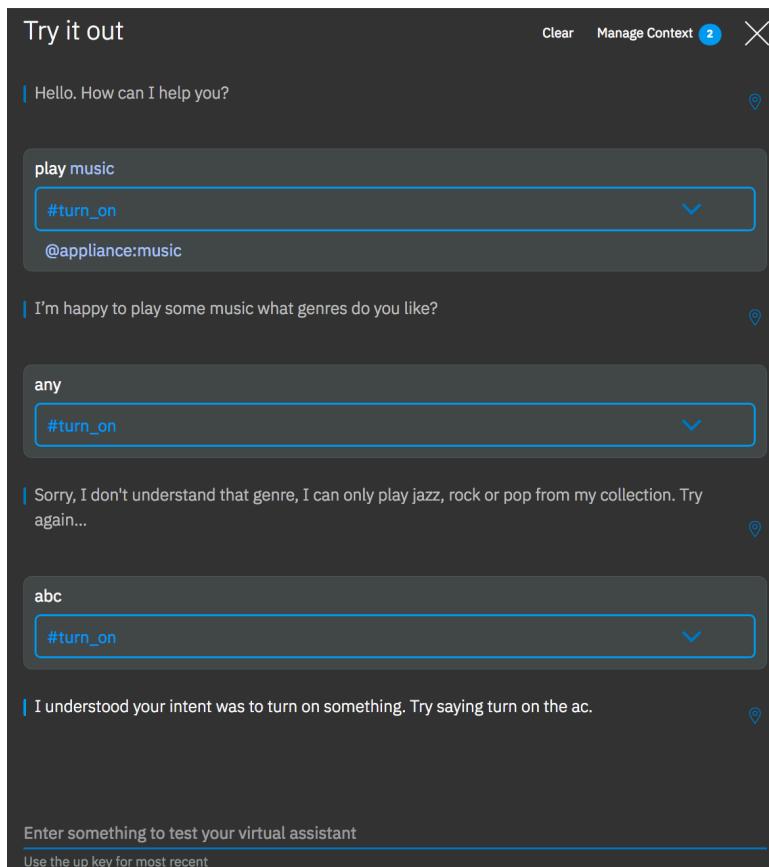
To do that, you must add a new **true child node** to the `@appliance:music` node.

- 72. Add a **child node** to the `@appliance:music` node and type **true** in the condition section.
- 73. Click the three vertical dots of this new *true* node, select **Jump to...**; click on the *turning on anything else* node, sibling of the `@appliance` node and select **Respond**.



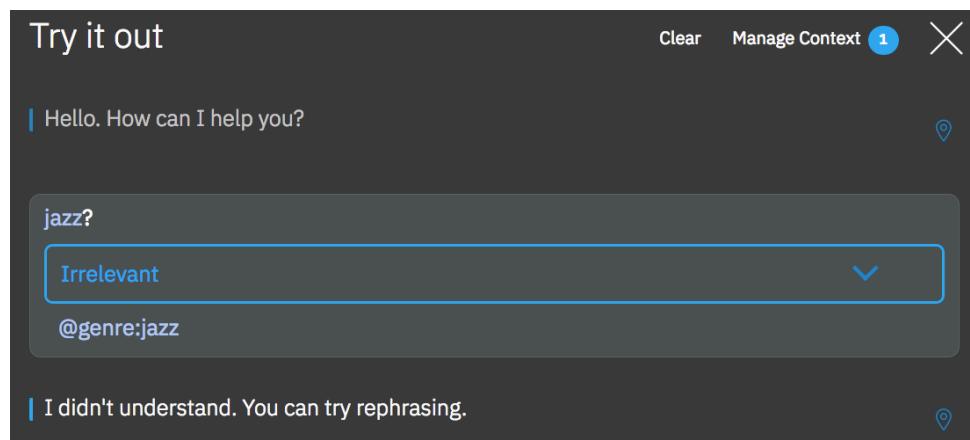
Let's now test the dialog again:

- ___ 74. Clear the chat and type: **play music**
- ___ 75. Type: **any**
- ___ 76. Type: **abc**



No more loop, right? Otherwise, check your JSON code inside the response sections of both nodes you edited. Usually the biggest gotchas are the curvy double quotes.

- 77. Let's try one more scenario. Clear the chat window and at the very start of the conversation type **jazz?**.

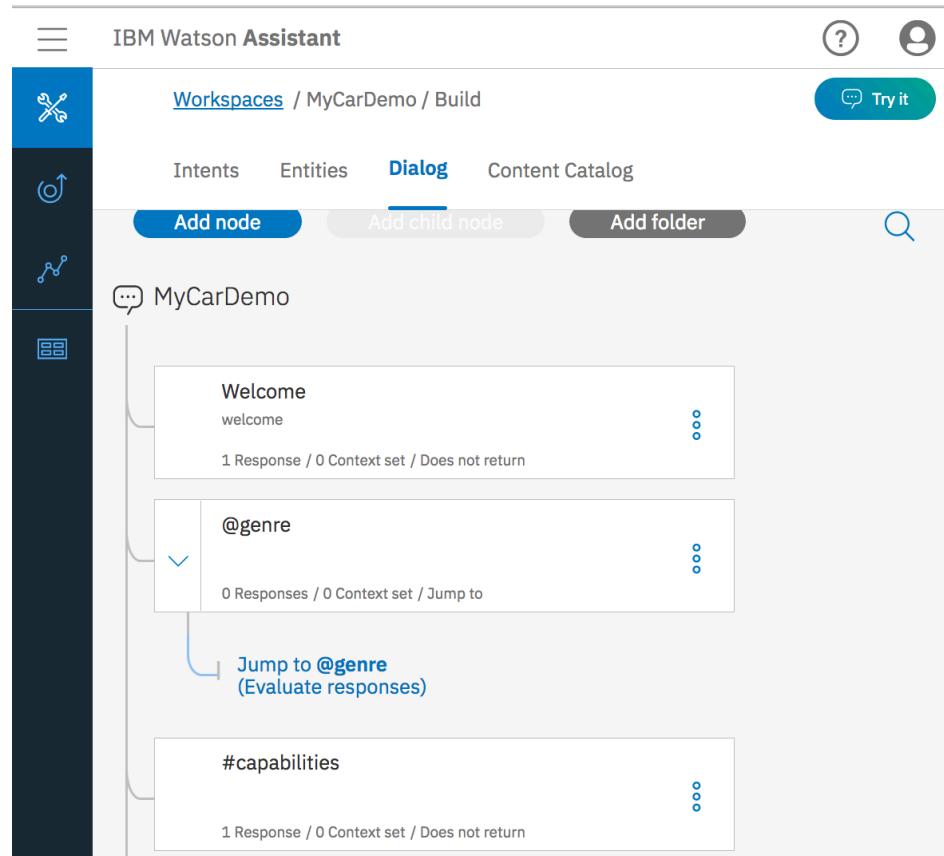


Notice that it did not understand the classification (Intent) but got the correct entity, then it does not know where to go yet and what if you want it to play jazz from your first response.

In other words, can you start the conversation by conditioning on an entity instead of intent first? The answer is YES!

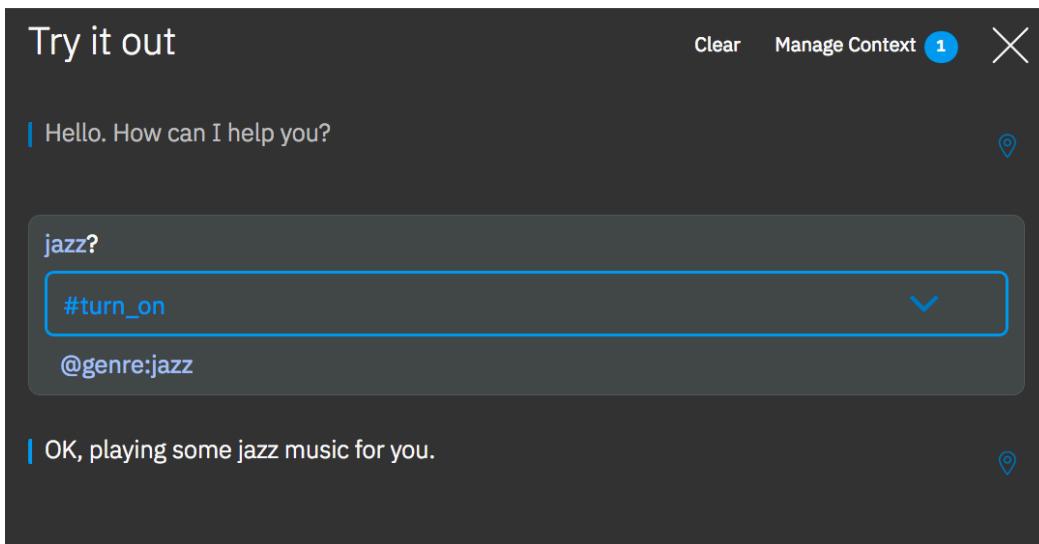
- 78. Create a sibling node underneath the **Welcome** node and set the condition to the entity by typing **@genre**.
- 79. Click the three vertical dots of this new node, select **Jump to**; click on the other **@genre** node all the way from the *turn_on* node and select **Respond**.

Because you want to display the “OK, playing some @genre music for you” from the very start.



Test it.

___ 80. Clear the chat window and from the start of the conversation, type: **jazz?**



Notice that it jumps straight to the entity (it is conditioned on entity instead of intent).

Congratulate yourself. In 81 steps you performed some advanced dialog creation techniques.

And here is the answer to a burning question that you may have: How do I handle chitchat: greetings and goodbyes and off topic conversation?

The best way to do that is to have a separate workspace that just understands chitchat and understands related intents and responds accordingly and then have the client app to talk to both workspaces; if the main workspace is not able to answer the question, then it can point to the adjacent workspace.