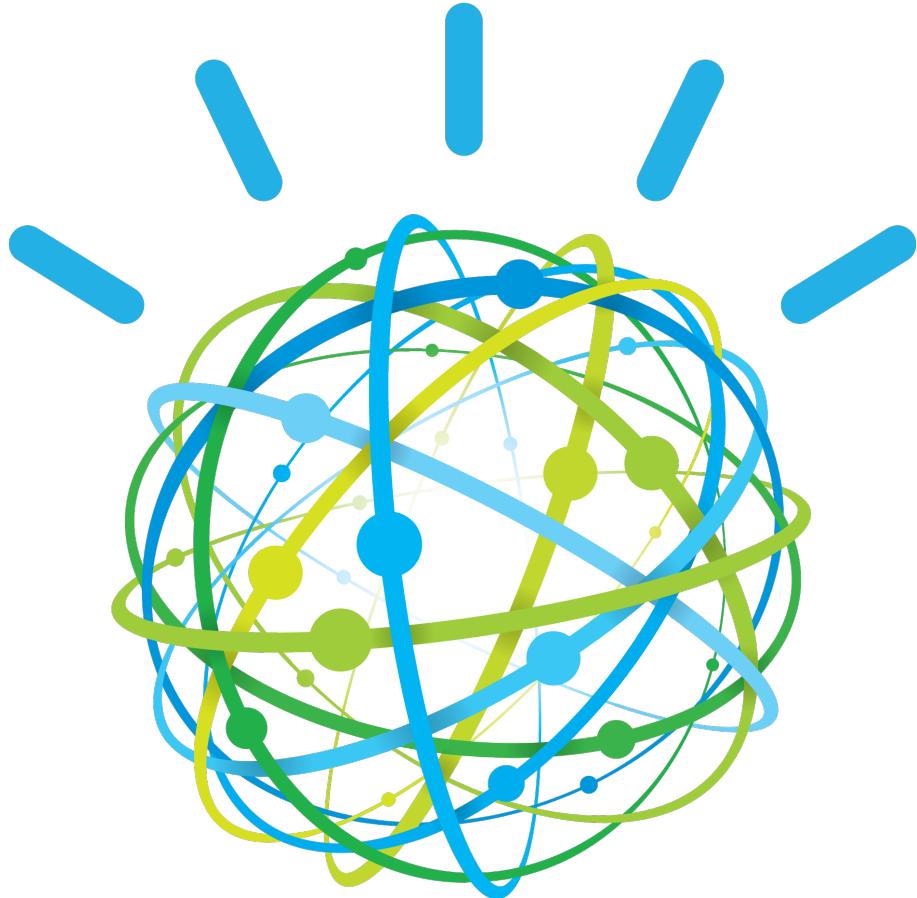


Lab 2 part 1: Building a Dialog

IBM Watson Assistant



Lab Instructions

Laurent Vincent

Content

Let's get started	3
1. Overview	3
2. Objectives	3
3. Prerequisites	3
4. Scenario	4
5. What to expect when you are done.....	4
Update the default settings	5
6. Update default settings	5
Building more natural answers.....	6
7. Update Greeting answers.....	6
8. Test the sequence of the responses.....	7
Welcome statement enhancements	8
9. Add contextual details	8
10. Context variable initialization	11
11. First “Jump to”	13
Human takeover	15
12. Increment the unanswer counter	15
13. Talk to concierge node	16
14. Does the Bot understand the user?	19
Hotel amenities management	23
15. Single node with one response per condition.....	23
16. Single node with multiple conditions / responses	27
17. How to keep the hotel amenity context? (Hotel_hours node)	30
18. How to keep the hotel amenity context? (Hotel Locations node)	35
Find a restaurant! (optional).....	38
19. Find a restaurant branch	38
20. Return responses node	39
21. Capture restaurant & mealtime nodes	41
22. Request missing information nodes	45
23. Loop management	46
24. Navigation node management	46
25. Test your branch.....	48
Enhance Find a restaurant branch (optional).....	51
26. Initialize context variables	51
27. Avoid infinite loop	52
28. Test the enhancements	57

Let's get started

1. Overview

The [IBM Watson Developer Cloud](#) (WDC) offers a variety of services for developing cognitive applications. Each Watson service provides a Representational State Transfer (REST) Application Programming Interface (API) for interacting with the service. Some services, such as the Speech to Text service, provide additional interfaces.

The [Watson Assistant](#) service combines several cognitive techniques to help you build and train a bot - defining intents and entities and crafting dialog to simulate conversation. The system can then be further refined with supplementary technologies to make the system more human-like or to give it a higher chance of returning the right answer. Watson Assistant allows you to deploy a range of bots via many channels, from simple, narrowly focused bots to much more sophisticated, full-blown virtual agents across mobile devices, messaging platforms like Slack, or even through a physical robot.

The **illustrating screenshots** provided in this lab guide could be slightly different from what you see in the Watson Assistant service interface that you are using. If there are colour or wording differences, it is because there have been updates to the service since the lab guide was created.

2. Objectives

Watson Assistant provides several options to manage conditions, and possibility to provide several answers to make your bot look more human. In this lab, you will learn how to manage:

- complex conditions,
- Multiple answers,
- context variables,
- what to do next.

3. Prerequisites

Before you start the exercises in this guide, you will need to complete the following prerequisite tasks:

- Lab 1 – building a conversation lab Instructions
- download the required files from the workshop materials Box.

Reminder of IBM Cloud URL:

Location	URL
worldwide	https://cloud.ibm.com/login

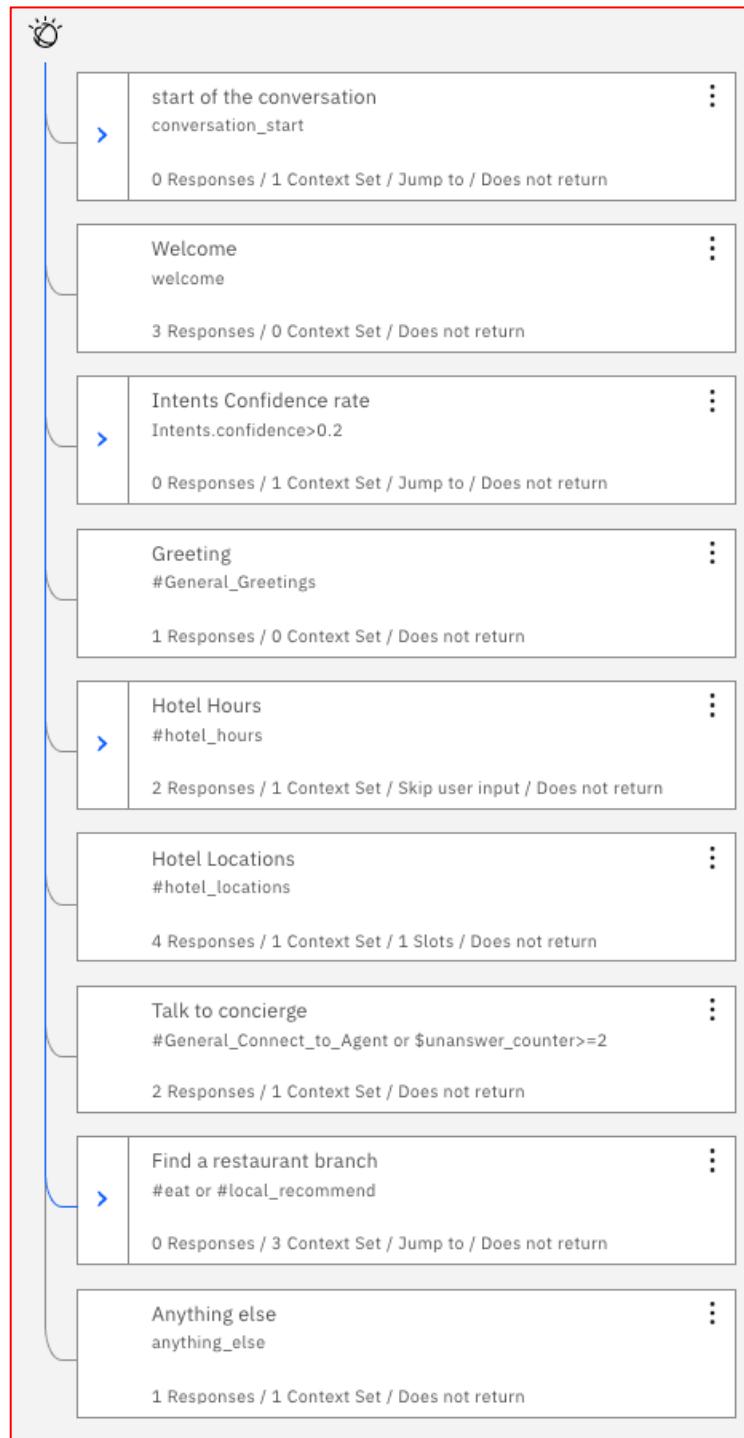
4. Scenario

Use case: A Hotel Concierge virtual assistant that is accessed from the guest room and the hotel lobby.

End-users: Hotel customers

5. What to expect when you are done

At the end of session, you should have a more complex dialog using several conditions and answers in the same node. Your dialog shall look like this:



Update the default settings

6. Update default settings

To facilitate the test of your next implementation, you are going to disable two options:
Disambiguation and Autocorrection

1. Go back to your skill, by clicking the **Skills** menu, then click on your **My Concierge skill** tile.

The screenshot shows the 'My Concierge Skill' tile. It includes the skill's name, type ('Dialog – English (US)'), creation date ('Jan 15, 2021 10:17 AM CET'), update date ('Jan 15, 2021 11:48 AM CET'), and a list of 'LINKED ASSISTANTS (1): Concierge'.

2. Click **options** menu, then **Disambiguation** menu
3. Turn the option off

The screenshot shows the 'Disambiguation' settings page for 'My Concierge Skill'. The 'Options' menu item is highlighted. The 'Disambiguation' section describes its purpose and includes a toggle switch set to 'Off'.

4. Click **Autocorrection** menu
5. Turn the option off

The screenshot shows the 'Autocorrection' settings page for 'My Concierge Skill'. The 'Options' menu item is highlighted. The 'Autocorrection' section describes its purpose and includes a note about looking for an icon in the 'Try it' panel and 'User conversations' page, along with a toggle switch set to 'Off'.

Building more natural answers

7. Update Greeting answers

In WA's response, you can perform any combination of these functions:

- update the dialog context
- provide a text response to the user (as you did in previous lab)
- change the flow of the dialog by using a **Jump to** action.

These three options are always processed in this order, regardless of the order in which you specify them.

Right now, we just want to humanize the bot's answer to make it more natural.

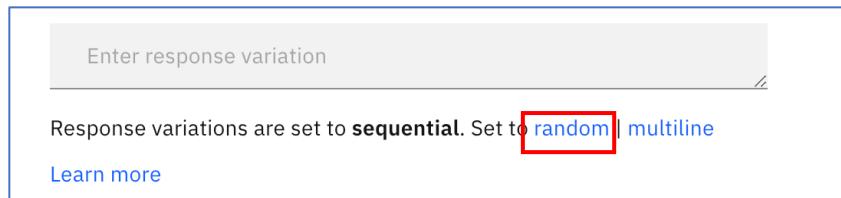
1. Open the **Dialog** page.
2. You are going to update the **Greeting** node and add several answers to make it look less mechanic and more human. Open the **Greeting** node.
3. Add the responses as shown in the table below. When you add a response and press enter, a new empty response field will open for you:

Field	Value
Assistant responds	<i>Hi! How may I help you?</i>
Assistant responds	<i>Hi! What you would like to do?</i>
Assistant responds	<i>Hi! What can I do for you?</i>

The screenshot shows the configuration of a Greeting node in the Microsoft Bot Framework designer. At the top, there is a 'Greeting' node name input field and a 'Customize' button. Below the node name, there is a note: 'Node name will be shown to customers for disambiguation so use something descriptive.' and a 'Settings' link. The main configuration area is divided into two sections: 'If assistant recognizes' and 'Assistant responds'. Under 'If assistant recognizes', there is a list containing '#General_Greetings'. Under 'Assistant responds', there is a 'Text' dropdown set to 'Text' with four response variations listed: 'Hi! What can I do for you?', 'Hi! How may I help you?', 'Hi! What you would like to do?', and 'Enter response variation'. There are also up and down arrows for reordering the responses. At the bottom, a note says 'Response variations are set to sequential. Set to random | multiline' and a 'Learn more' link.

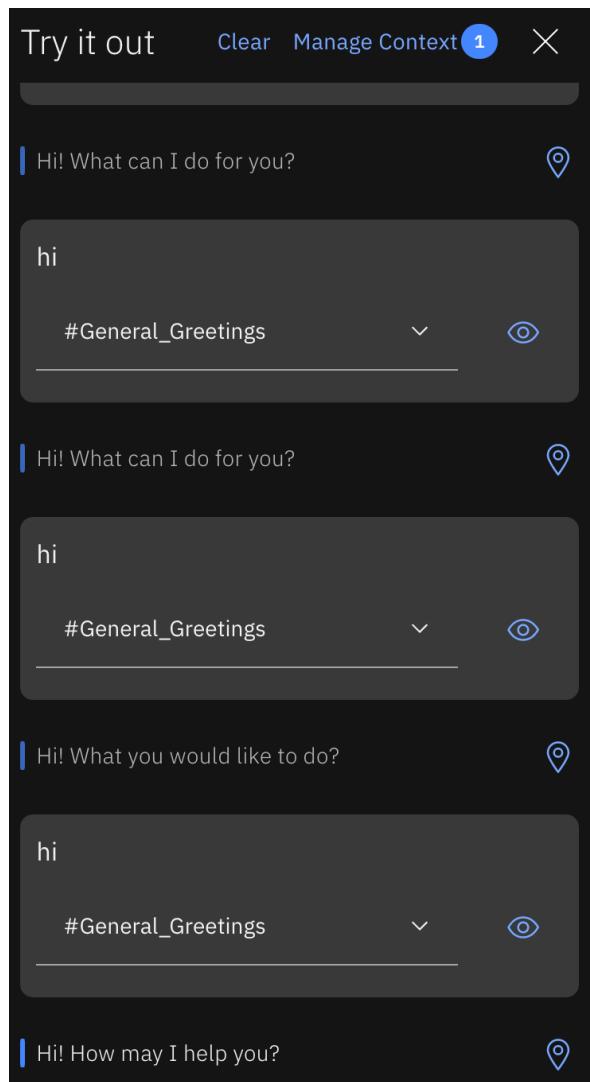
If you want to, you can decide to return these responses sequentially (default) or randomly. In this case, we want to change it to randomly.

4. Click on **random**, at the bottom of the response frame.



8. Test the sequence of the responses

1. Open **Try it out** panel
2. Enter *hi* several times and review the Watson's response. You will see randomly chosen responses from your *Assistant responds* fields.



Welcome statement enhancements

Welcome statement is the first step in the conversation where we want to:

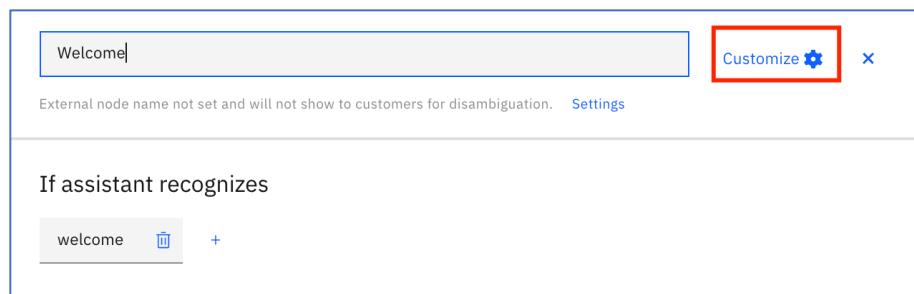
- Engage the user to continue with conversation
- Include information about the role of the solution and its name (if needed)
- Consider using contextual information such as the time or user's name
- Initialize the context variables if required.

9. Add contextual details

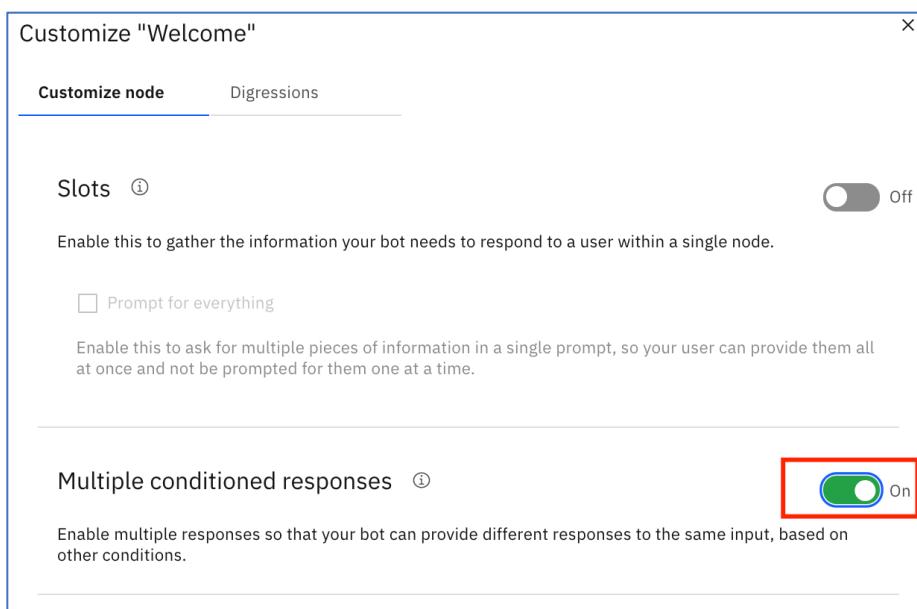
You are going to update the Watson's welcome according to the time of the day.

You can do it by any other criteria which could makes sense for you.

1. Select **Welcome** node which is the first node of your dialog
2. Click **Customize** at the top right of the box



3. Enable **Multiple responses**



4. Click **Apply**

5. Replace the response *Hi! I am Watson...* with *Good morning!*

6. Enter the following condition *now().before('12:00:00')*

You should get the result below. You are using the method 'before()' to evaluate if the current time is before noon. That's why the answer is *Good morning*.

Assistant responds		
	If assistant recognizes	Respond with
1	now().before('12:00:00')	Good morning!
Add response +		

7. Click on **Add response**.

[Add response +](#)

8. Repeat the steps from 6 to 8 with :

condition : *now().after('12:00:00')*

response : *Good Afternoon!*

9. Click on **Add response**

10. Repeat the steps from 6 to 8 with :

condition : *anything_else*

response : *Hi!*

Now you should have:

	If assistant recognizes	Respond with
1	now().before('12:00:00')	Good morning!
2	now().after('12:00:00')	Good Afternoon!
3	anything_else	Hi!

Note: if you need to move up or down a response, you can use the arrows at the left of each selected row.

Assistant responds

If assistant recognizes	Respond with		
1 now().before('12:00:00')	Good Morning!		-
2 now().after('12:00:00')	Good afternoon!		-
3 anything_else	Hi!		-

11. Click **Edit Icon** on the first conditioned response

If assistant recognizes	Respond with	
1 now().before('12:00:00')	Good morning!	

12. Enter the following responses :

Field	Value
Assistant responds	<i>I am Watson. I can answer questions about the Hotel.</i>
Assistant responds	<i>How can I help you today?</i>

13. Click **multiple** option to return all responses in one time to the end-user.

If assistant recognizes

now().before('12:00:00') +

Assistant responds

Text

Good morning!

I am Watson. I can answer questions about the Hotel.

How can I help you today?

Enter response variation

Response variations are set to **sequential**. Set to [random](#) | [multiline](#)

[Learn more](#)

Then you should have this

The screenshot shows a configuration dialog with a blue border. At the top, there is a text input field labeled "Enter response variation". Below it, a message says "Response variations are set to **multiline**. Set to [sequential](#) | [random](#)". A red rectangular box highlights the word "multiline". At the bottom left, there is a link "Learn more".

14. Click **Save**

15. Repeat the 4 last steps with the 2 others conditioned responses.

16. Click **multiple** option to return all responses in one time to the end-user.

10. Context variable initialization

The response you define for the **welcome** node (*welcome* condition) in the dialog is displayed to initiate a conversation from the "Try it out" pane, and from other integrations, like **Web chat** or **Preview link**. However, it is not displayed from the **Slack** and **Facebook** integrations because nodes with the *welcome* special condition are skipped in dialog flows that are started by users. Assistants that are deployed to messaging channels typically wait for users to initiate conversations with them, not the other way around.

Unlike the *welcome* special condition, the *conversation_start* special condition is always triggered at the start of a dialog. You can use a combination of these nodes to manage the start of your dialog in a consistent way and use *conversation_start* to initialize any context variable.

1. Select the **Welcome** node and add a child by clicking on the 3 dots icon and select **Add node above**.
2. Fill the node like this

Field	Value
Node name	<i>start of the conversation</i>
If assistant recognizes	<i>conversation_start</i>
Assistant responds	

We will initialize 1 context variable required for our conversation:

- **\$unanswer_counter:0** This variable will be used to count the number of iterations through the dialog when no correct answer was provided to the user (ie. Anything else node was triggered). Once the counter will reach the defined threshold, our bot will request a human action.

1. Open the **context editor**, by clicking the 3 dots menu, then select **Open context editor**.

start of the conversation

If assistant recognizes

conversation_start +

External node name not set and will not show to customers for disambiguation. [Settings](#)

Customize X

Assistant responds

Text ...

Enter response text

Open JSON editor Open context editor

Response variations are set to **sequential**. Set to [random | multiline](#)

Learn more

2. You will now specify the first variable, fill in the fields like this:

Context variable : *unanswer_counter*

Context value : *0*

Variable	Value
unanswer_counter	0

Add variable +

3. Delete the empty response

Then set context

Variable	Value
unanswer_counter	0

Add variable +

Assistant responds

Text ...

Enter response text

Response variations are set to **sequential**. Set to [random | multiline](#)

Learn more

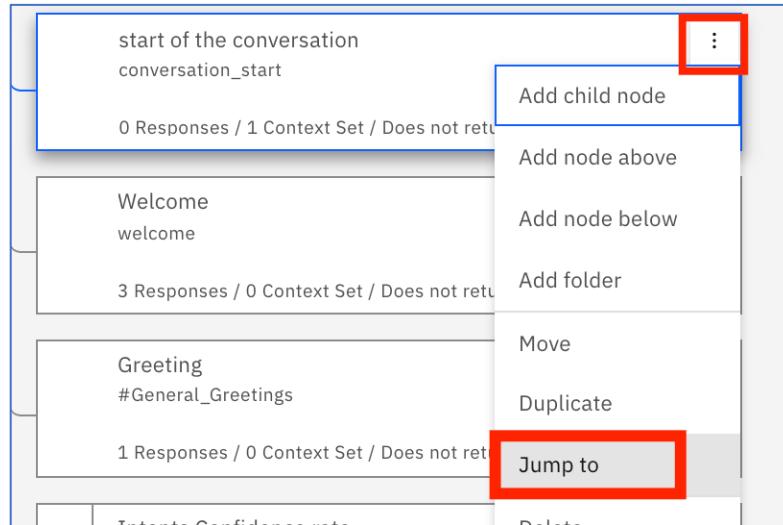
Delete

4. Close the edit page:

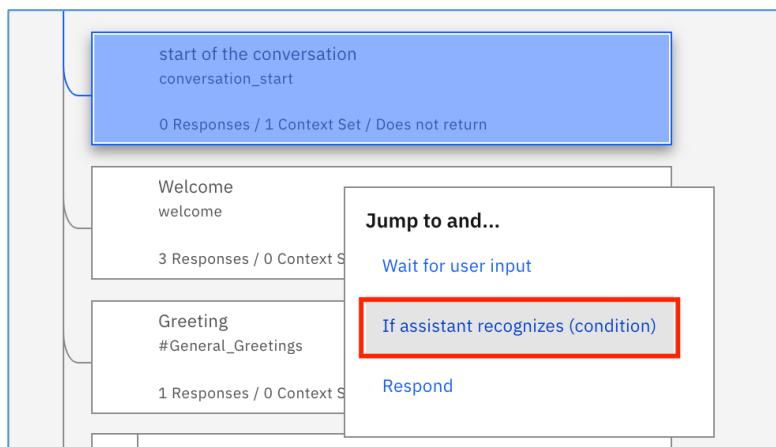
11. First “Jump to”

As this stage, if you are using the **Try it out**, the welcome statement doesn't return any response. You must chain **start of the conversation** and **Welcome** node, to execute them consecutively.

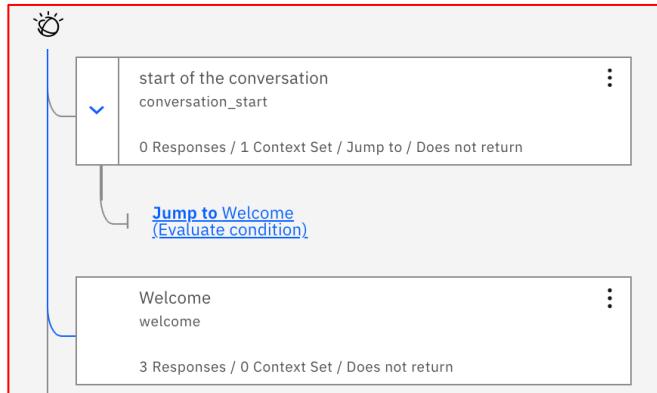
1. Select the **start of the conversation** node
2. In the **contextual menu** select **Jump to**



3. Select **Welcome** node as destination
4. Then select **if assistant recognizes**



Now, you should have the first 2 nodes with following configuration:

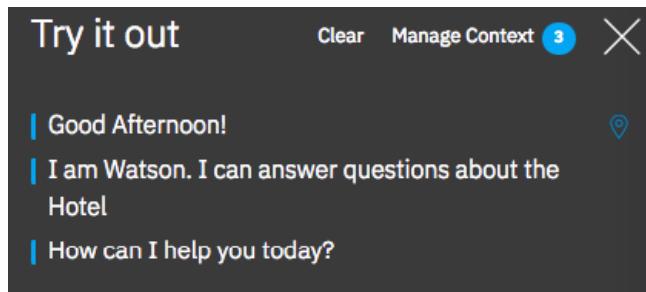


This design results in a dialog that works like this:

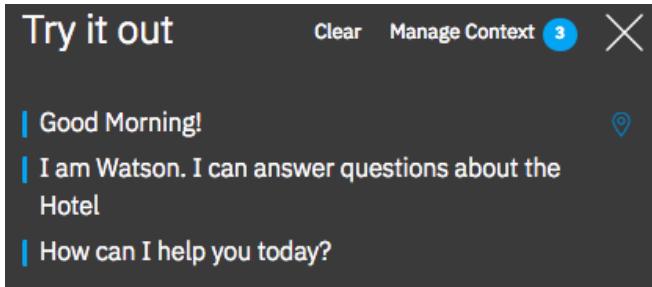
- Whatever the integration type, the `conversation_start` node is processed, which means any context variables that you define in it are initialized.
- In integrations where the assistant starts the dialog flow, the `welcome` node is triggered and its text response is displayed.
- In integrations where the user starts the dialog flow, the user's first input is evaluated and then processed by the node that can provide the best response.

5. You are ready to test it. Open **Try it out** panel and clear previous conversation.

Watson should provide you the following welcome message:



or



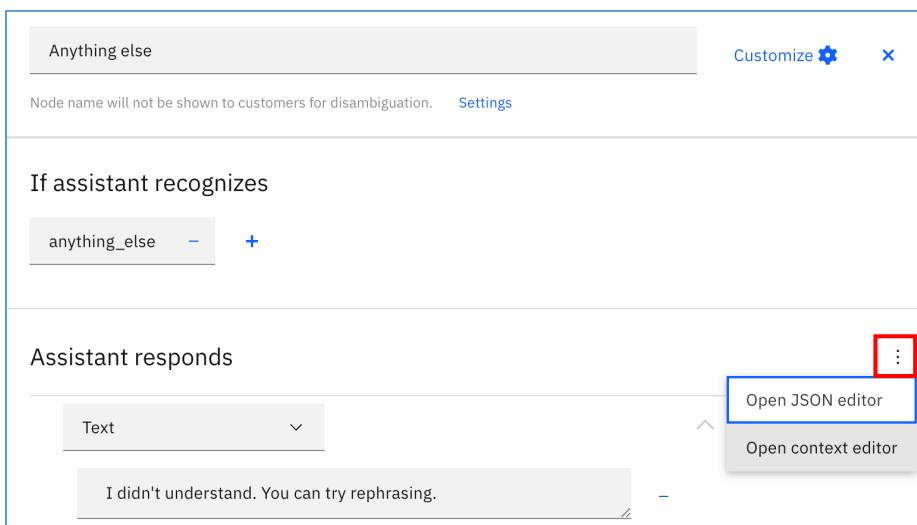
Human takeover

12. Increment the unanswer counter

The **Anything else** node is the overall fallback node. If it gets hit too many times in a row, we must manage this to avoid user's frustration. To do this, we are going to engage a human agent after a number of consecutive triggers of the Anything else node. So, we are going to:

- Increment a counter for every hit of the Anything else node,
- Leverage this counter in a new node to request a human agent action, when the threshold is reached. We decide that 3 iterations without answer is our threshold.

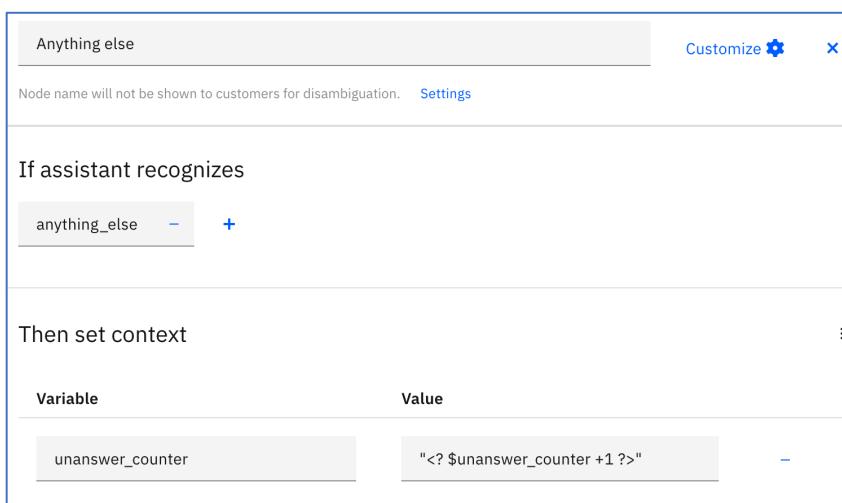
1. Select **Anything else** node to open its editor
2. Open the **context editor**



3. Fill it in like this

Variable : *unanswer_counter*

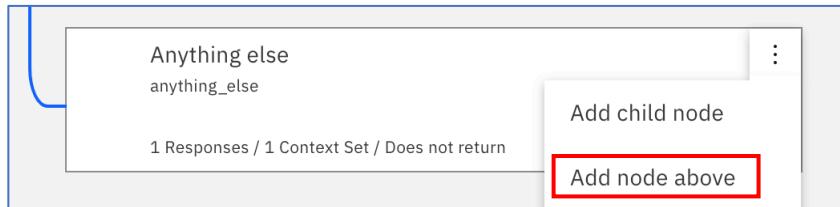
Value : "*<? \$unanswer_counter +1 ?>*"



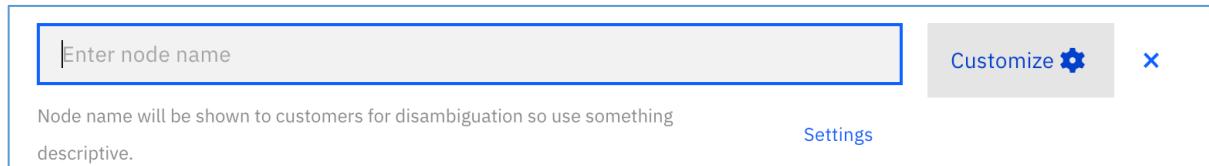
- Close the **Anything else** node editor.

13. Talk to concierge node

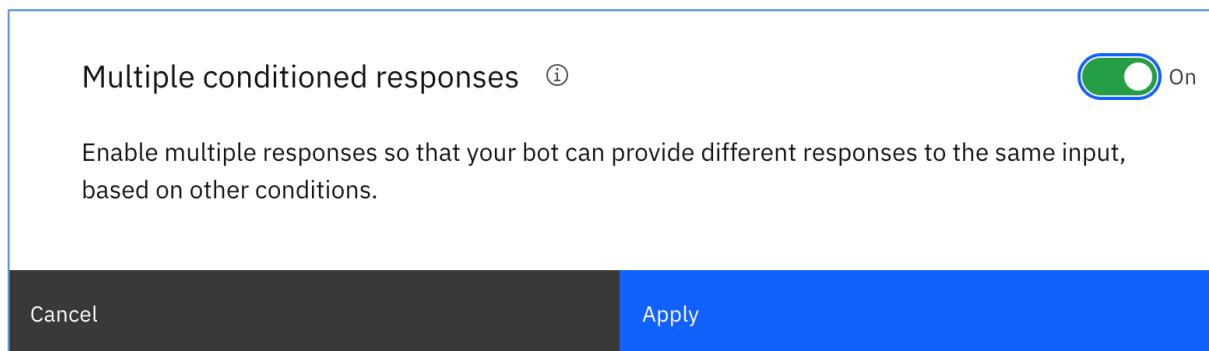
- Add a node above the **Anything else** node



- To support multiple conditions, click on the **customize** button (top right):



- Activate **multiple responses** and click **Apply**:



- Here are the parameters of your new node

Field	Value
Name of the node	<i>Talk to concierge</i>
If assistant recognizes	<i>#General_Connect_to_Agent or \$unanswer_counter>=2</i>
Response 1	
Condition	<i>#General_Connect_to_Agent</i>
Responses	<i>A concierge is going to contact you in less than 2 minutes.</i>
Response 2	
Condition	<i>\$unanswer_counter>=2</i>
Responses	<i>Sorry, I am having difficulty helping you, a concierge is going to contact you in less than 2 minutes.</i>

Your node should look like this:

The screenshot shows a configuration interface for a node named "Talk to concierge". At the top, there's a "Customize" button with a gear icon and an "X" button. Below the name, a note says "Node name will be shown to customers for disambiguation so use something descriptive." with a "Settings" link. The main section is titled "If assistant recognizes" and contains two conditions separated by an "or": "#General_Connect_to_Agent" and "\$unanswer_counter>=2". Below this, under "Assistant responds", there are two rows. Row 1: "If assistant recognizes" is "#General_Connect_to_Agent" and "Respond with" is "A concierge is going to contact you". Row 2: "If assistant recognizes" is "\$unanswer_counter>=2" and "Respond with" is "Sorry, I am having difficulty handling your request". Each row has a "Customize" gear icon and a minus sign.

If assistant recognizes	Respond with
1 #General_Connect_to_Agent	A concierge is going to contact you
2 \$unanswer_counter>=2	Sorry, I am having difficulty handling your request

In this case, the response returned is a text. If you did an integration with a service desk solution, you could use the response type : "[Connect to human agent](#)". It enables to hand over to an operator of the service desk.

To measure containment accurately, WA must be able to identify when a human intervention occurs. The metric primarily uses the : "[Connect to human agent](#)". response type as an indicator. If a user conversation log includes a call to a : "[Connect to human agent](#)" response type, then the conversation is considered to be **not contained**.

However, not all human interventions are transacted by using this response type. as you use an alternate method to deliver additional support, you must take additional steps to register the fact that the customer's need was not fulfilled by the assistant. so, you need to flag these types of redirects so the containment metric is able to pick them up and display them in WA dashboard:



5. Click on the setting icon of the first conditioned response

If assistant recognizes	Respond with
#General_Connect_to_Agent	A concierge is going to contact you ⚙️ ✖️

6. Open context editor

If assistant recognizes

#General_Connect_to_Agent ✖️ +

Assistant responds

Text ⋮

A concierge is going to contact you in less than 2 minutes. ✖️

Enter response variation ✖️

Response variations are set to **sequential**. Set to [random](#) | [multiline](#)

⋮ Open JSON editor Open context editor

7. Add the context variable `context_connect_to_agent` with the value `true`

Then set context ⋮

Variable	Value
context_connect_to_agent	true ✖️

8. Click **Save**

9. Close the **Talk to concierge** node editor.

14. Does the Bot understand the user?

In our case, to manage the bot's inability to provide the right answer, we want to contact a human agent, but we could use any other useful action in order to help the user to go further.

In the example above we have two reasons to contact somebody:

- a direct request of the user, triggered by `#General_Connect_to_Agent`,
- the inability to provide an answer managed by `$unanswer_counter`.

The counter is set to zero in the Welcome node and incremented in the Anything else node. Since we only want to increment the counter with consecutive hits, we need to reset it when our bot does understand the user. So, we will add a single node just below the **Welcome** node that checks the confidence rate and resets the counter to 0 if the confidence is above 20%.

1. Add a node below the **Welcome** node
2. Fill the node as below

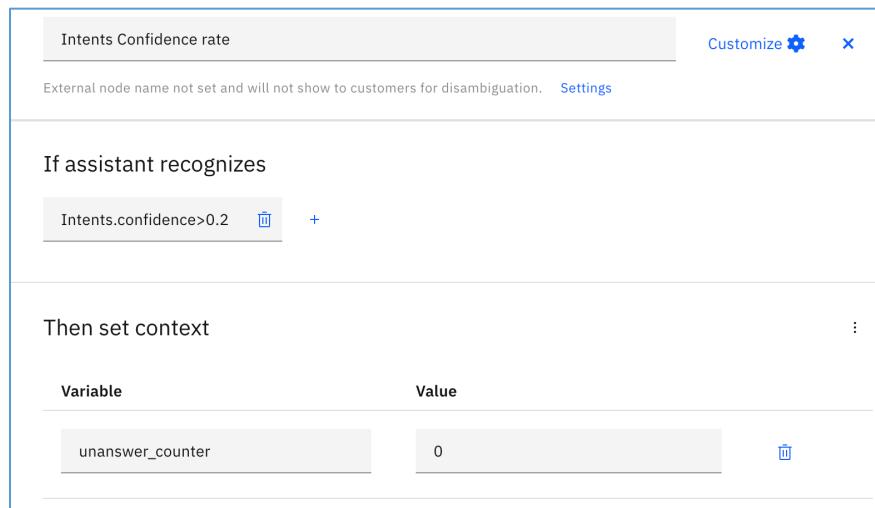
Field	Value
Node name	<i>Intents Confidence rate</i>
If assistant recognizes	<i>Intents.confidence>0.2</i>
Assistant responds	

10. Open the nodes **context editor** (If you can't remember how, scroll up few pages).

11. Fill it in like this

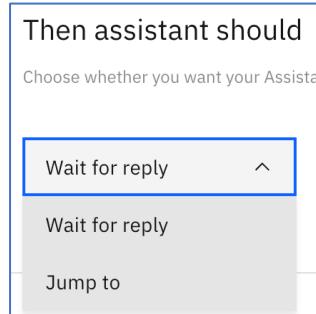
Variable : `unanswer_counter`

Value : `0`



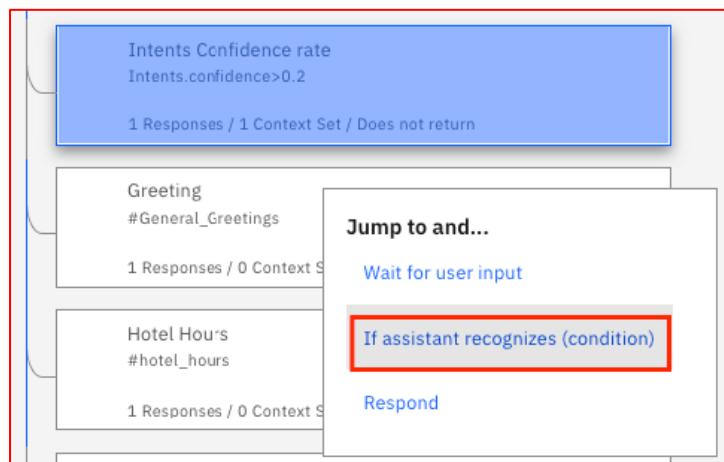
As we don't expect any interaction with the user in this node, the node must jump to the next existing node – the **Greeting** node.

12. Click **Wait for reply** drop box (bottom of the editor page)

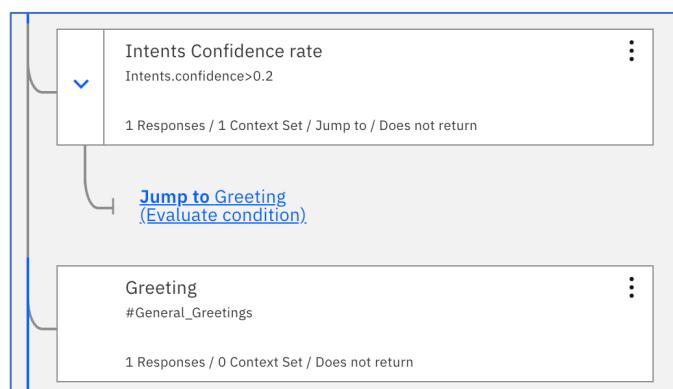


13. Select **Jump to** option

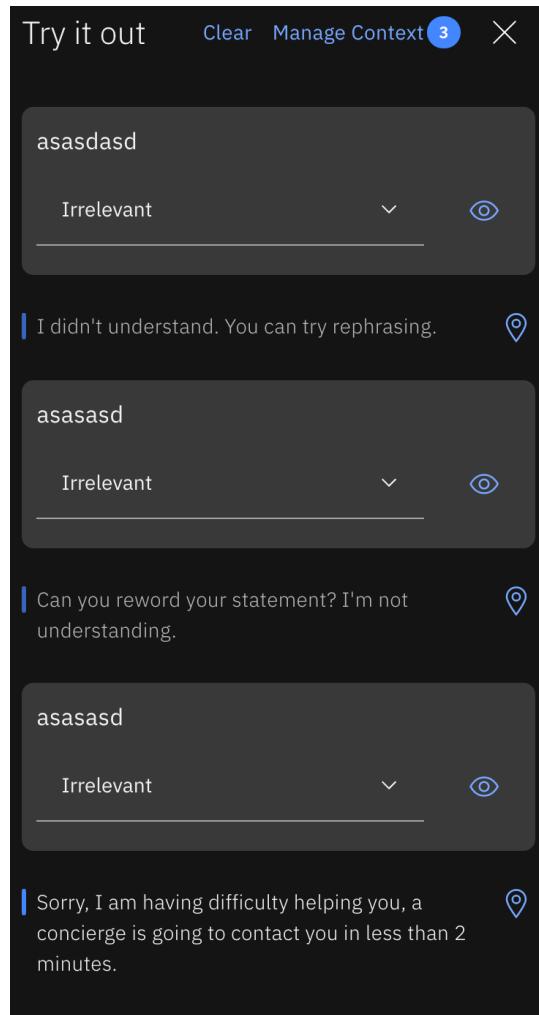
14. Select **Greeting** node, then select **If assistant recognizes (condition)**



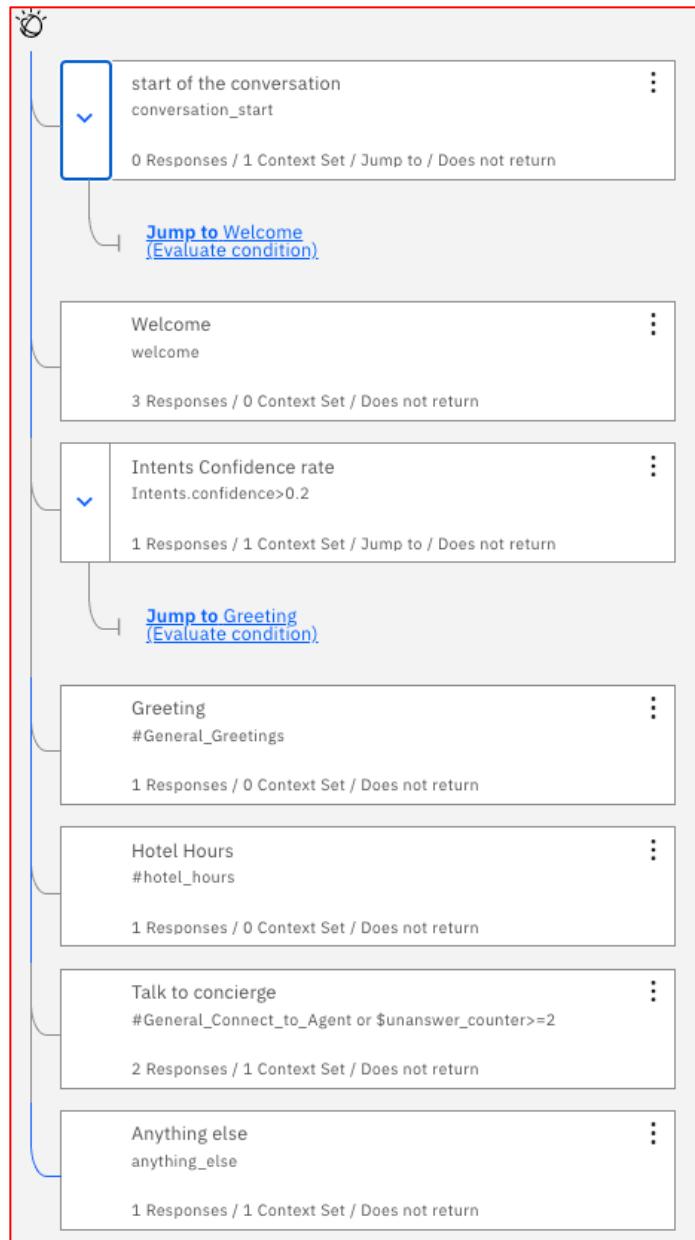
Your partial node tree should now look like this:



15. Go to **Try it out** panel to test the behaviour of your bot. Deliberately type in 3 times some nonsensical input. After the 3rd try, the bot should contact the agent.



Your full dialog should at this point look like on the picture below. Check your own dialog, if it looks differently, go back, find where you made a mistake and fix it.



Hotel amenities management

15. Single node with one response per condition

In this section, you will practice building children nodes using the Jump to function. If you have a difficult time working on your screen due to panel overlap, you can close your **Try it out** panel or increase your browser window size.

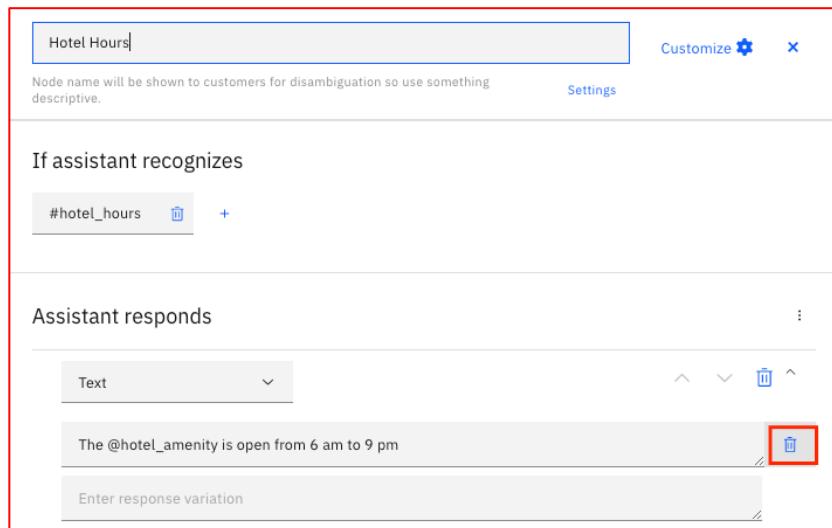
1. Click the **Hotel Hours** node to highlight it
2. Click on the contextual menu and click **Add child node**



3. In the editor of the new node, fill it like this:

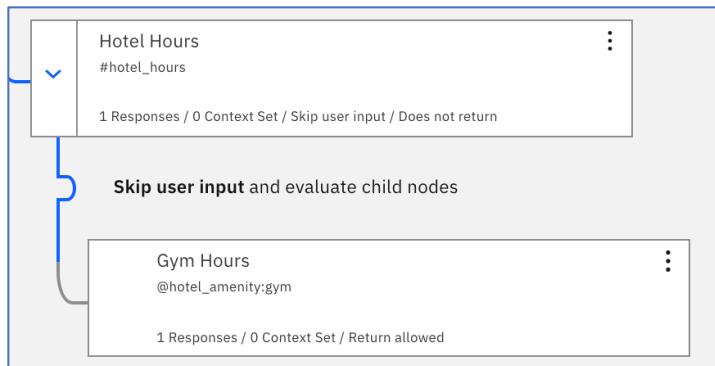
Field	Value
Node name	<i>Gym Hours</i>
If assistant recognizes	<i>@hotel_amenity:gym</i>
Assistant responds	<i>The @hotel_amenity.value is open from 5am to 11pm.</i>

4. Go back and click the **Hotel Hours** node to highlight it
5. Remove the existing response by clicking on the **delete** icon



- At the bottom of the page, click **Wait for reply** drop box and select **Skip user input** option. This will tell Watson to move straight to the child node.

Now, you should have this :



7. Repeat the step 1 to 3 to add 4 more children, with the following inputs

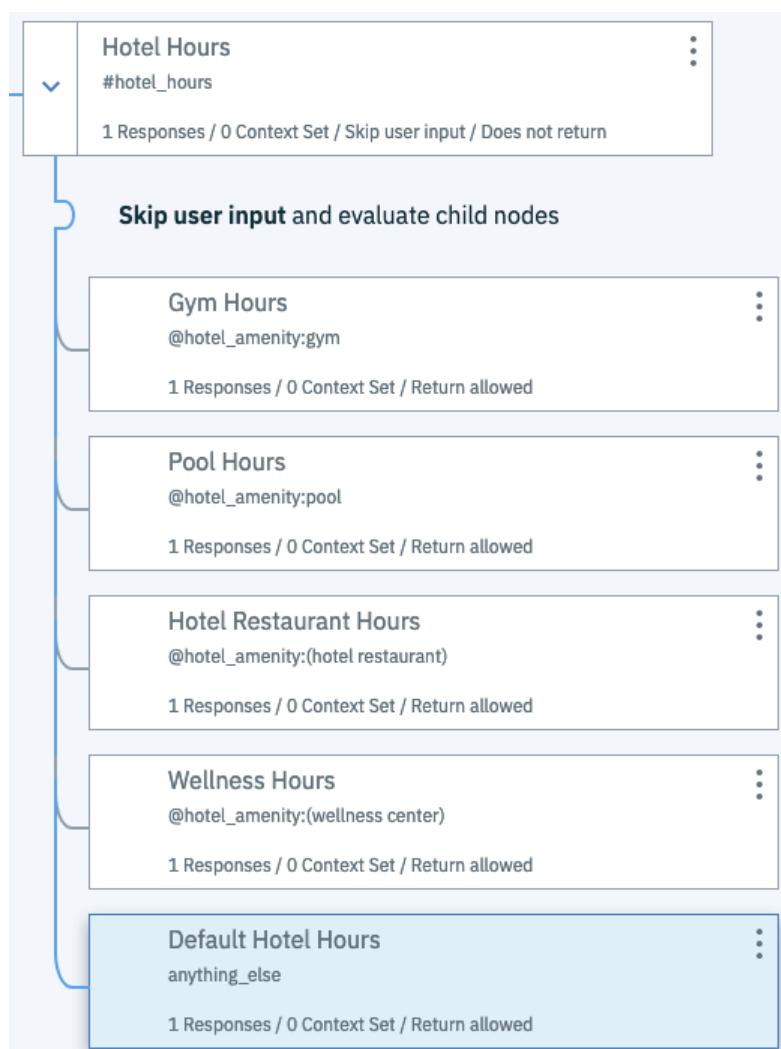
Field	Value
Node name	<i>Pool Hours</i>
If assistant recognizes	<i>@hotel_amenity:pool</i>
Assistant responds	<i>The @hotel_amenity.value is open from 5am to 8pm.</i>

Field	Value
Node name	<i>Hotel Restaurant Hours</i>
If assistant recognizes	<i>@hotel_amenity:(hotel restaurant)</i>
Assistant responds	<i>The @hotel_amenity.value is open from 5am to 2pm and from 6pm to 1am.</i>

Field	Value
Node name	<i>Wellness Hours</i>
If assistant recognizes	<i>@hotel_amenity:(wellness center)</i>
Assistant responds	<i>The @hotel_amenity.value is open from 10am to 9pm.</i>

Field	Value
Node name	<i>Default Hotel Hours</i>
If assistant recognizes	<i>anything_else</i>
Assistant responds	<i>I understand that you would like to learn more about our hours of operation. You can ask me specifically what hours you are looking for (gym, restaurant, pool, sauna), or you may call the front desk for more information.</i>

Your chat flow should look similar to this:



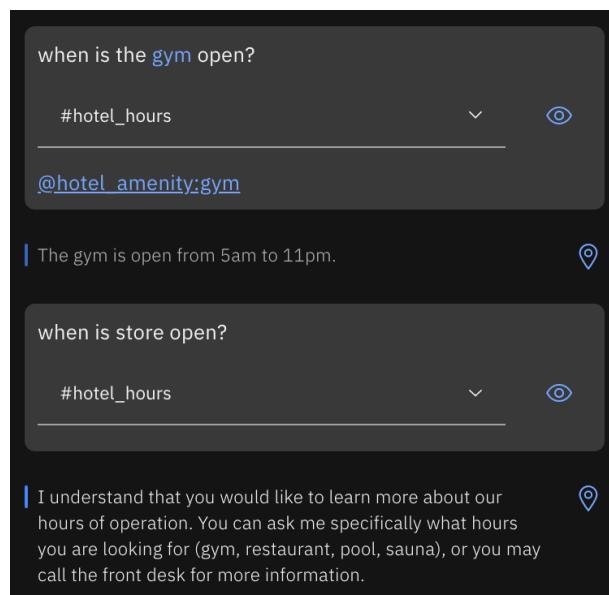
8. Open the **Try it out** panel and clear previous conversation.

9. Type *when is the gym open?*

Watson should return the response that is matched to `#hotel_hours` and
`@hotel_amenity:gym`

10. Type *when is store open?*

Watson should return the answer that you put in the **Default Hotel Hours** Response. This is because Watson was able to recognize that you are asking for the hours when a location is open, but it did not recognize the *store* as an entity that was used as a condition in any of the child nodes

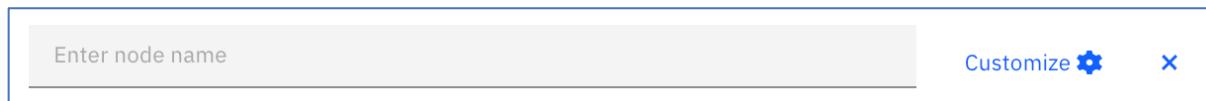


16. Single node with multiple conditions / responses

You have created children nodes that have one response per condition. It is possible for a single dialog node to have multiple conditions that trigger a different response, called a multiple condition response. This approach enables you to simplify your dialog tree.

As you already created a such node for **Welcome**, instead of giving you specific step-by-step instructions for this section, you will be given a list of conditions with the paired responses. There is a screen shot of the final dialog node after the instructions.

1. Create a new dialog node below the **Hotel Hours** node. (This should be a root node in the main dialog tree)
2. To support multiple conditions, click on the **customize** button (top right):



The screenshot shows a dialog node configuration window. At the top right are two buttons: a blue 'Customize' button with a gear icon and a white 'X' button. Below the buttons is a text input field with the placeholder 'Enter node name'.

3. Activate **multiple conditioned responses** and click **Apply**:
4. Here are the parameters of your new node

Field	Value
Node name	<i>Hotel Locations</i>
If assistant recognizes	<i>#hotel_locations</i>
Response 1	
Condition	<i>@hotel_amenity:pool or @hotel_amenity:(wellness center)</i>
Responses	<i>The @hotel_amenity.value is on the second floor.</i>
Response 2	
Condition	<i>@hotel_amenity:gym</i>
Responses	<i>The @hotel_amenity.value is in the front lobby.</i>
Response 3	
Condition	<i>@hotel_amenity:(hotel restaurant)</i>
Responses	<i>The @hotel_amenity.value is located on the ground floor of the hotel.</i>
Response 4	
Condition	<i>anything_else</i>
Responses	<i>I understand that you'd like to locate something in the hotel. Please ask me for the specific amenity you are looking for, (gym, restaurant, pool, sauna) or you may call the front desk for directions.</i>

Hotel Locations

Node name will be shown to customers for disambiguation so use something descriptive.

If assistant recognizes

#hotel_locations - +

Assistant responds

If assistant recognizes	Respond with
1 @hotel_amenity:pool or @hot	The @hotel_amenity.value is c
2 @hotel_amenity:gym	The @hotel_amenity.value is i
3 @hotel_amenity:(hotel restaura	The @hotel_amenity.value is l
4 anything_else	I understand that you'd like to

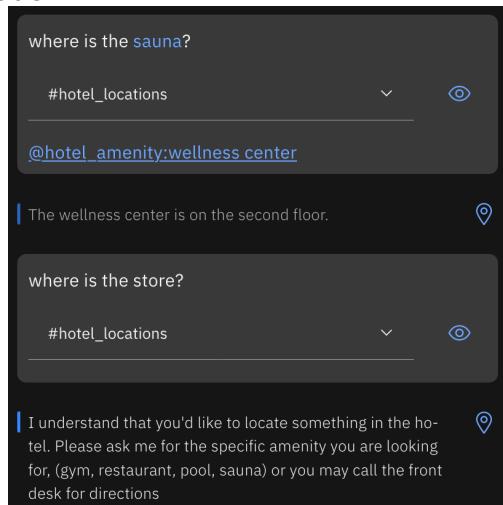
5. Close the node editor and open the **Try it out** panel

6. Type *where is the sauna?*

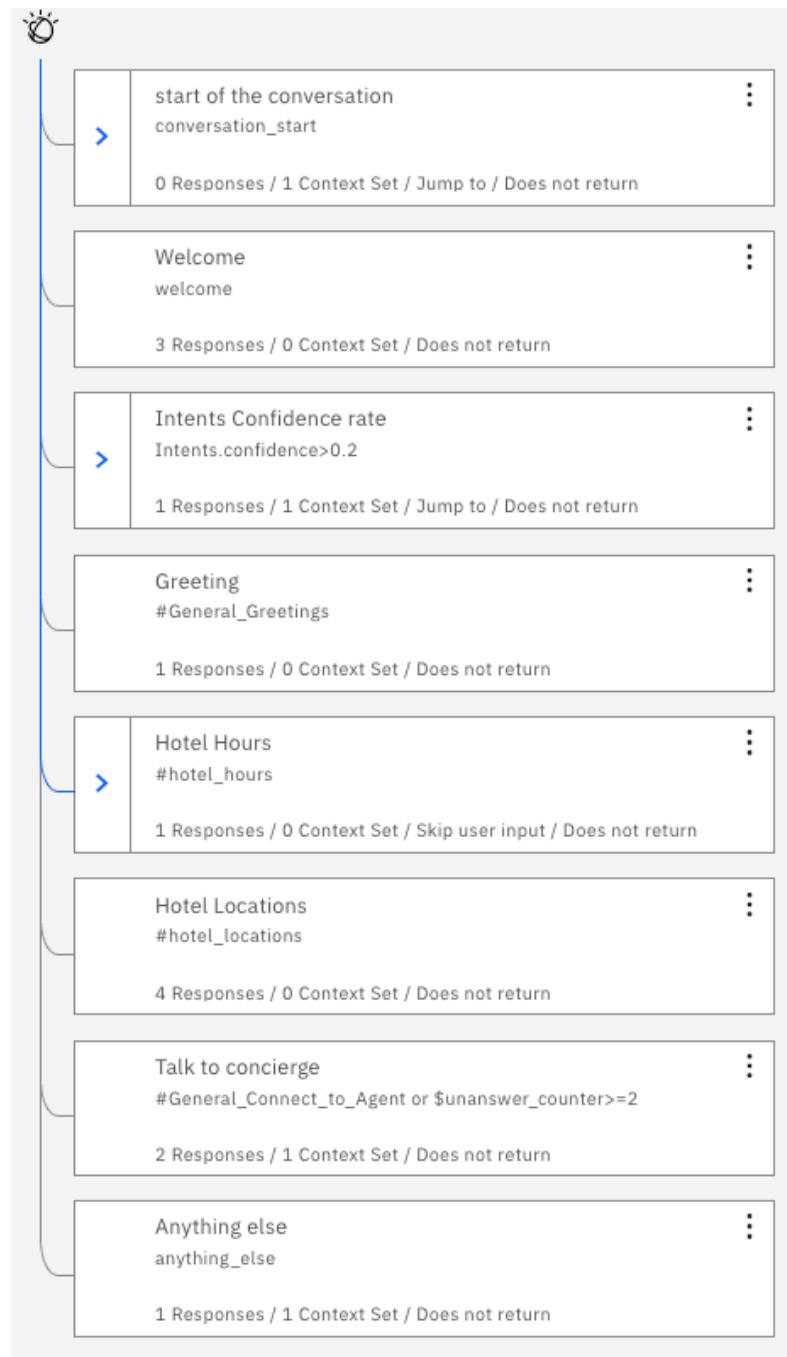
Watson should return *The wellness center is on the second floor.*

7. Type *where is the store?*

Watson should return the answer that you put without any condition (the last one). This is because Watson was able to recognize that the input was related to asking for location, but it did not recognize the *store* as an entity that was used as a condition in your node



At this stage, your dialog looks like this:



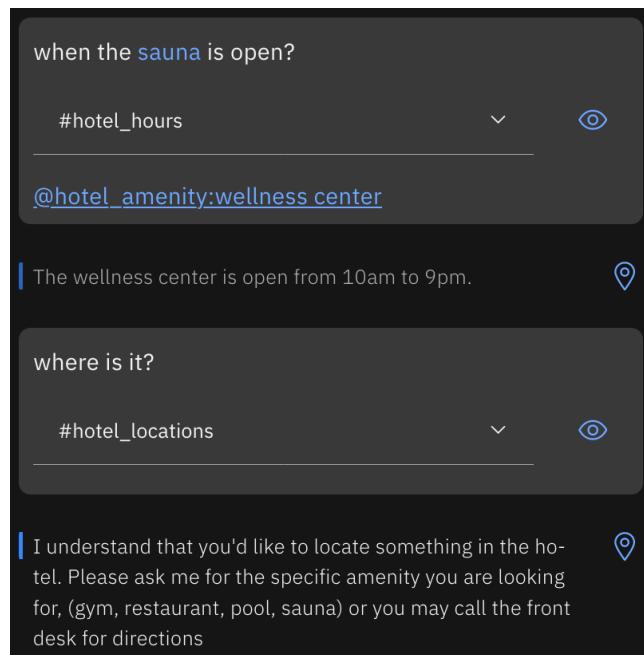
17. How to keep the hotel amenity context? (Hotel_hours node)

1. Open the **Try it out** panel
2. Type *when is the sauna open?*

Watson should return *The wellness center is open from...*

3. Type *where is it?*

In a natural conversation, you should expect that Watson keep in mind that the question is related to the sauna! Unfortunately, Watson returns the default answer because the entities are not persistent during the conversation.



To store data, you must use context variables. Context variable is also the mechanism for passing data between WA and the application that submits the user's input to Watson.

In this case, you are going to use context variable `$hotel_amenity`.

There are two methods to access it:

- Shorthand: `$hotel_amenity`
- Full syntax: `context.hotel_amenity`

4. Click the **Hotel Hours** node to edit it and click **Customize** button, then enable the **multiple conditioned responses** and click **Apply**.
5. In the first row, enter `@hotel_amenity` as condition and click on the **wheel** (**Customize response** icon) on the right of condition line

If assistant recognizes

If assistant recognizes	Respond with
#hotel_hours	Enter a response

Assistant responds

If assistant recognizes	Respond with
1 @hotel_amenity	Enter a response

Customize response

6. After a new window opens, open the **context editor** (Click on the three dots on the right of condition line then select **Open context editor**)

Configure response 1

If assistant recognizes

@hotel_amenity

Assistant responds

Text

Enter response text

...
Open JSON editor
Open context editor

7. Fill in the context like this

Variable : `hotel_amenity`

Value : `"@hotel_amenity"`

Then set context

Variable	Value
hotel_amenity	"@hotel_amenity"

That's the way to store the entity value of `@hotel_amenity` in the context variable `$hotel_amenity` when the `@hotel_amenity` is captured by Watson

You can do the same thing using the JSON editor. If you open it (click 3 dots, then click **Open JSON editor**), you should see this in the editor:

Assistant responds

```
1  {
2    "output": {
3      "generic": [
4        {
5          "values": [],
6          "response_type": "text",
7          "selection_policy": "sequential"
8        }
9      ]
10     },
11     "context": {
12       "hotel_amenity": "@hotel_amenity"
13     }
14 }
```

8. Click **Save**

9. Click **Add response + (bottom)**

If assistant recognizes

#hotel_hours - +

Assistant responds

If assistant recognizes

1 @hotel_amenity

Respond with

Enter a response



Add response +

10. Fill in the second row like this:

condition : *anything_else*

response :

Assistant responds		
If assistant recognizes		Respond with
1	@hotel_amenity	Enter a response -
2	anything_else	Enter a response -

11. In each child of **Hotel Hours** node, you must replace

`@hotel_amenity` with `$hotel_amenity` as condition
`@hotel_amenity.value` with `$hotel_amenity` in the response

Note: pay attention to all @ and \$ signs!

For example, in the first node (Gym Hours), you should get the following screen:

The screenshot shows the Dialogflow interface for creating a node named "Gym Hours". The node name is displayed at the top left. On the right, there are "Customize" and "X" buttons. Below the node name, a note says "Node name will be shown to customers for disambiguation so use something descriptive." with a "Settings" link. The main area is titled "If assistant recognizes" and contains a list item "\$hotel_amenity:gym" with a minus sign and a plus sign next to it. Below this, under "Assistant responds", there is a "Text" dropdown menu set to "Text" with a downward arrow. To the right of the dropdown are three small arrows pointing up, down, and left, and a blue "X" button. A text input field contains the response "The \$hotel_amenity is open from 5am to 11pm." with a minus sign to its right.

After you updated each child node, you should get the dialog below:



The **Hotel Hours** node, now, uses 2 responses and set one Context

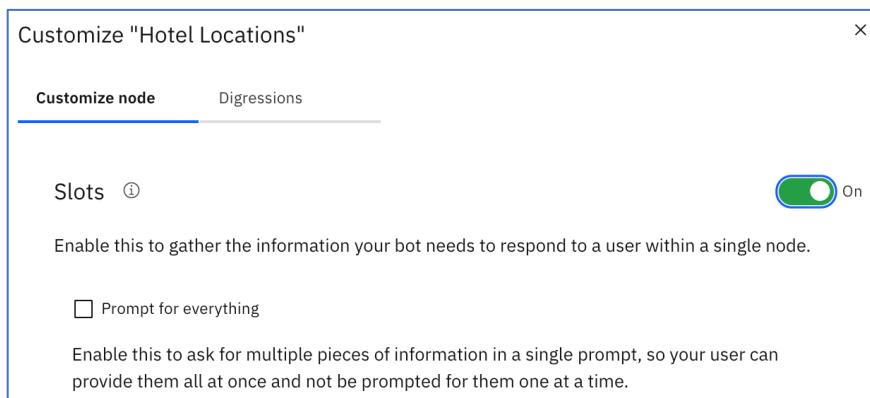
If you test these updates now, the behaviour of your bot didn't change yet

18. How to keep the hotel amenity context? (Hotel Locations node)

The next step is to update the node Hotel Locations to set the context variable `$hotel_amenity` and manage it correctly.

To set the context variable you are going to use a very simple slot.

1. Select **Hotel Locations** node to edit it
2. Click **Customize** (upper right of the window)
3. Enable **Slots** option



4. Click **Apply**.
5. Fill the slot like this:
 - Check for : `@hotel_amenity`
 - Save it as : `$hotel_amenity`

If assistant recognizes

- +

Then check for 0 Manage handlers

Check for	Save it as	If not present, ask	Type
1	<input type="text" value="@hotel_ameni"/>	<input type="text" value="\$hotel_amenit"/>	Enter prompt Optional ⚙️ -

6. In each response of this node, you must replace
`@hotel_amenity` with `$hotel_amenity` as condition
`@hotel_amenity.value` with `$hotel_amenity` in the response

Note: pay attention to all @ and \$ signs!

Assistant responds

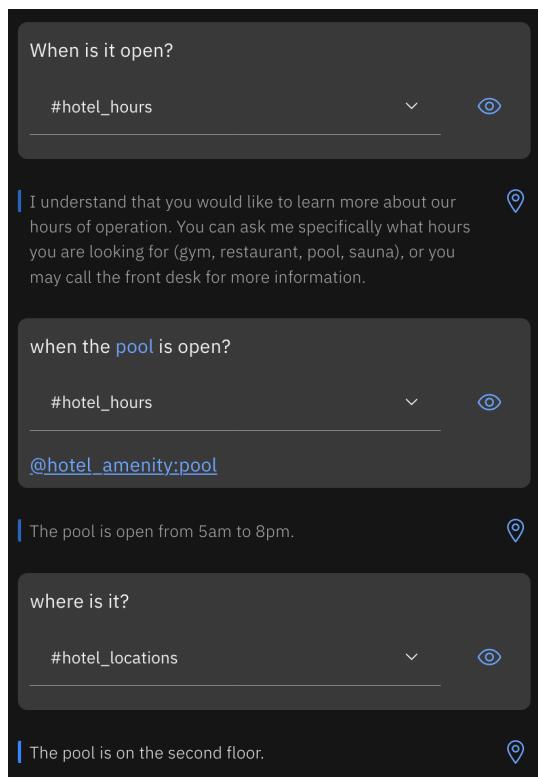
If assistant recognizes	Respond with	
1 <code>\$hotel_amenity:pool or \$hotel_amenity:sauna</code>	The <code>\$hotel_amenity</code> is on the second floor.	⚙️
2 <code>\$hotel_amenity:gym</code>	The <code>\$hotel_amenity</code> is in the front entrance.	⚙️
3 <code>\$hotel_amenity:(hotel restaurant)</code>	The <code>\$hotel_amenity</code> is located on the first floor.	⚙️
4 anything_else	I understand that you'd like to look around.	⚙️

7. Let's test it, open the **Try it out** panel, enter successively:

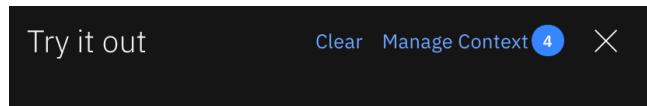
When is it open?

when the pool is open?

where is it?



At the top of the Try it out panel, you can see that Watson manages few context variables.



8. Open the **Manage Context** panel

You will see that the `$hotel_amenity` variable is correctly set to *pool*.

A screenshot of the Manage Context panel. It has a title "Context variables" at the top left and a close "X" button at the top right. Below the title is a search bar starting with "\$" and placeholder text "Enter variable name". Underneath is a list of variables. The first variable is "\$hotel_amenity" with a value of "'pool'" and a circular "⊖" icon to its right. There is also a small "⊕" icon to the left of the variable name.

9. Close the **Manage Context** panel and click on **Clear** to reinitialize the context

10. On the **Try it out** panel, enter successively:

Where is it?

where is the pool?

when is it open?

You will get the following behaviours:

A screenshot of the Try it out panel showing three interactions. 1. The first interaction shows the user asking "Where is it?" and the system responding with a dropdown menu "#hotel_locations" and a location pin icon. 2. The second interaction shows the user asking "where is the pool?" and the system responding with a dropdown menu "@hotel_amenity:pool" and a location pin icon. 3. The third interaction shows the user asking "when is it open?" and the system responding with a dropdown menu "#hotel_hours" and a location pin icon. Each interaction includes a text input field and a response message below it.

Find a restaurant! (optional)

Let's find a restaurant for two potential intents:

- you want to eat something
- you are looking for a specific cuisine (such as French, Italian etc.).

Watson needs 2 data: the type of restaurant and when you want to go there. Sometimes, the entity details can be captured in other nodes or the user might just have forgotten to explicitly determine the object of this intent (entity). In such case, you must request more details if required.

You will manage to provide the right information to the end-users and request the required details to do it.

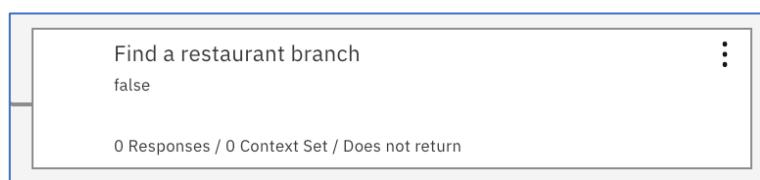
19. Find a restaurant branch

1. Add a new node above the **Anything else** node of the main tree.
2. Fill the node as below:

Field	Value
Name of the node	<i>Find a restaurant branch</i>
If assistant recognizes	<i>false</i>
Assistant responds	(Delete the response field)

The screenshot shows the configuration for the 'Find a restaurant branch' node. At the top, the node name is displayed. Below it, the 'If assistant recognizes' field is set to 'false'. Under 'Assistant responds', there is a 'Text' input field with a red box highlighting the delete icon (a trash can) next to it. A tooltip for this icon says 'Delete response type'. At the bottom, a note states 'Response variations are set to sequential. Set to random | multiline'.

The new node looks like this:



For the time being, this node cannot be triggered, it is just a placeholder to connect the new branch which we will create later. This is why it has no response and the

condition is false.

20. Return responses node

As a follow up, we check if `$restaurant` and `$mealtime` variables have been stored and complete the information gathering if they have. Initially, of course, we won't have this information, so this node will be skipped.

1. Add a child to **Find a restaurant branch** node and fill in the node as below (don't forget to enable **Multiple conditioned responses** option):

Node name: *Return responses*

condition: *\$restaurant and \$mealtime*

resp1 condition: *\$restaurant:french_restaurant*

resp1 response: *The Paris restaurant is located on 1st street and open for \$mealtime.*

resp2 condition: *\$restaurant:japanese_restaurant*

resp2 response: *The Tokyo restaurant is located on 2nd street and open for \$mealtime.*

resp3 condition: *\$restaurant:pizza_restaurant*

resp3 response: *The Rome restaurant is located on 3rd street and open for \$mealtime.*

resp4 condition: *anything_else*

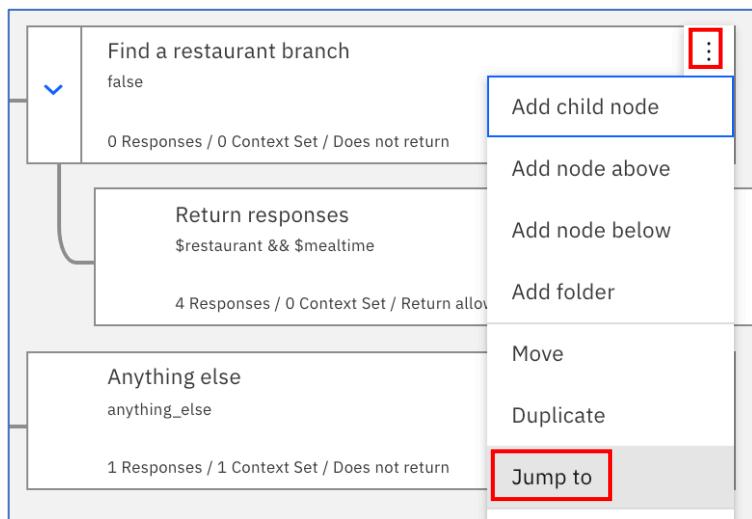
resp4 response: *I understand that you would like to learn more about local restaurant. you may call the front desk for more information.*

The screenshot shows the configuration of a 'Return responses' node. At the top, there's a header bar with the node name 'Return responses', a 'Customize' button, and a close 'X' button. Below the header, there's a descriptive text field containing placeholder text: 'Node name will be shown to customers for disambiguation so use something descriptive.' To the right of this text is a 'Settings' link. The main configuration area has three sections: 'Condition', 'Responses', and 'Else'. In the 'Condition' section, there are three input fields: '\$restaurant', 'and', and '\$mealtime', each with a minus '-' button to its right. In the 'Responses' section, there are four entries, each consisting of a condition and a response, followed by a settings icon and a minus '-' button. The entries are:

Condition	Response	Settings	Remove
1 \$restaurant:french_restaurant	The Paris restaurant is located	⚙️	-
2 \$restaurant:japanese_restaurant	The Tokyo restaurant is locate	⚙️	-
3 \$restaurant:pizza_restaurant	The Rome restaurant is locate	⚙️	-
4 anything_else	I understand that you would li	⚙️	-

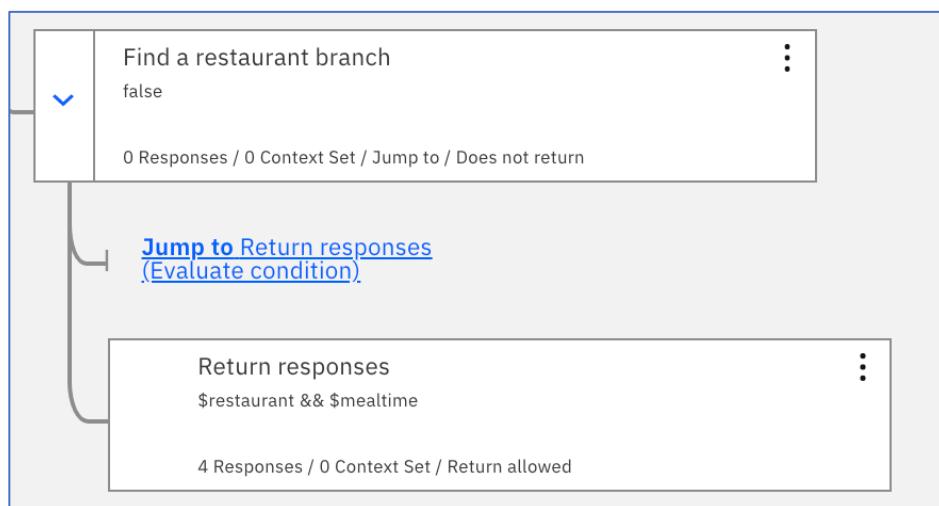
You can add more responses for the other types of restaurants, for this lab 3 are enough.

3. As we want to execute this node without waiting for any input from the user, we will use the **Jump to** capability. Select **Find a restaurant branch** node.
4. Click **Jump to** on the 3 dots menu.



5. Select the **Return responses** node.
6. As we want to evaluate the condition, select **If assistant recognizes (condition)**

Now you should have:



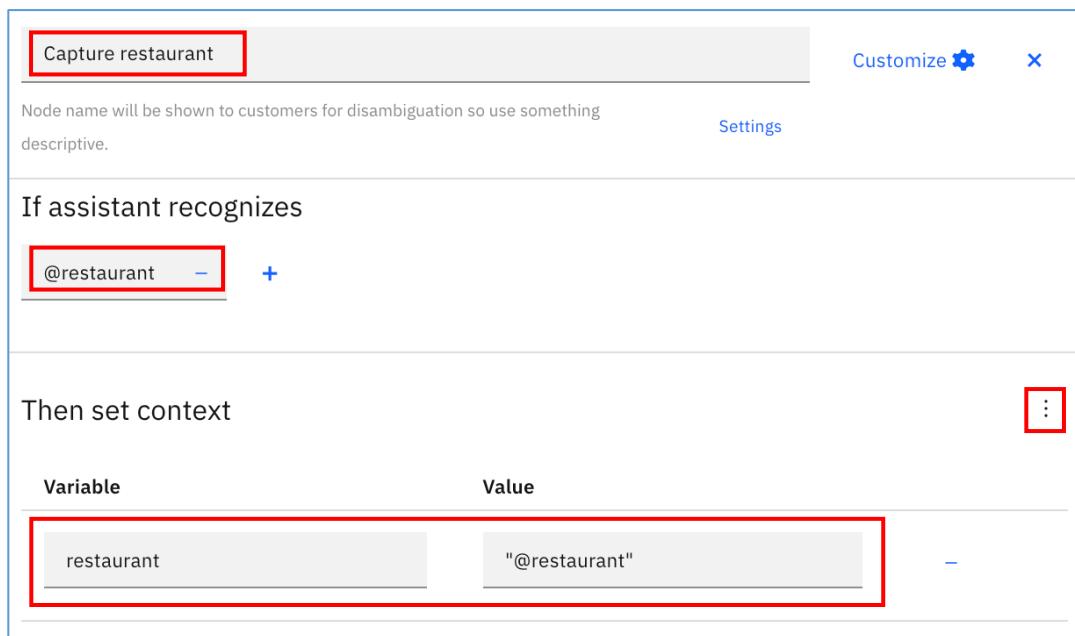
Next, we check to see if the user has provided the type of restaurant. If they did, we will store it in the context and continue from the next node.

21. Capture restaurant & mealtime nodes

1. Select **Return responses** node and add a new node below it by clicking **Add node below**.

2. Fill the node as below

name: *Capture restaurant*
condition: *@restaurant*
context variable: *restaurant*
context value: *@restaurant*



If this node will be executed, it means Watson got the required information so we can go back to the first node to display the answer to the user.

Now, you must capture the mealtime.

3. Select **Capture restaurant** node and add a new node below it by clicking **Add node below**. Enable **multiple conditioned responses**.

4. Fill the node as below

name: *Capture mealtime*
condition: *@mealtime or @sys-time*
resp1 condition: *@mealtime*
resp1 context Variable: *mealtime*
resp1 context Value: *@mealtime*
resp2 condition: *@sys-time>('15:00:00')*
resp2 context Variable: *mealtime*
resp2 context Value: *dinner*
resp3 condition: *@sys-time>('11:00:00')*
resp3 context Variable: *mealtime*
resp3 context Value: *lunch*
resp4 condition: *anything_else*
resp4 context Variable: *mealtime*
resp4 context Value: *breakfast*

Now you should have:

The screenshot shows the configuration of a node named "Capture mealtime". The node's name is displayed at the top left. To the right are "Customize" and "X" buttons. Below the name is a descriptive note: "Node name will be shown to customers for disambiguation so use something descriptive." A "Settings" link is also present. The main configuration area contains a search bar with the placeholder "@mealtime" and an "or" operator, followed by another search bar with the placeholder "@sys-time". Below this, the "Assistant responds" section lists four conditions with their corresponding responses:

If assistant recognizes	Respond with
1 @mealtime	⚙️ -
2 @sys-time>('15:00:00')	⚙️ -
3 @sys-time>('11:00:00')	⚙️ -
4 anything_else	⚙️ -

Note: When you click on each row, **configure** (the wheel icon), then **Open context editor**, you should see something like the below 4 screenshots. *Delete the response option for each row.*

Configure response 1

If assistant recognizes

@mealtime - +

Then set context

Variable	Value
mealtime	"@mealtime"

Add variable +

Assistant responds

Add response type +

Configure response 2

If assistant recognizes

@sys-time>('15:00:00') - +

Then set context

Variable	Value
mealtime	"dinner"

Add variable +

Assistant responds

Add response type +

Configure response 3

If assistant recognizes

@sys-time>("11:00:00") - +

Then set context

Variable	Value
mealtime	"lunch"

Add variable +

Assistant responds

Add response type +

Configure response 4

If assistant recognizes

anything_else - +

Then set context

Variable	Value
mealtime	"breakfast"

Add variable +

Assistant responds

Add response type +

22. Request missing information nodes

Now we will check for missing information. If the user didn't give us complete information, we will prompt for it. After prompting, we continue from the user input.

1. Select **Capture mealtime** node and add a new node after it by clicking **Add node below**.
2. Fill the node as below (don't forget to enable **Multiple conditioned response** option)

name: *Request restaurant*

condition: *!\$restaurant or !\$mealtime*

resp1 condition: *!\$restaurant*

resp1 response: *What kind of restaurant or what food would you like?*

Resp2 condition: *!\$mealtime*

Resp2 response: *When would you like to go to the restaurant?*

The screenshot shows the configuration for a node named "Request restaurant".

Node Name: Request restaurant

Description: Node name will be shown to customers for disambiguation so use something descriptive.

If assistant recognizes:

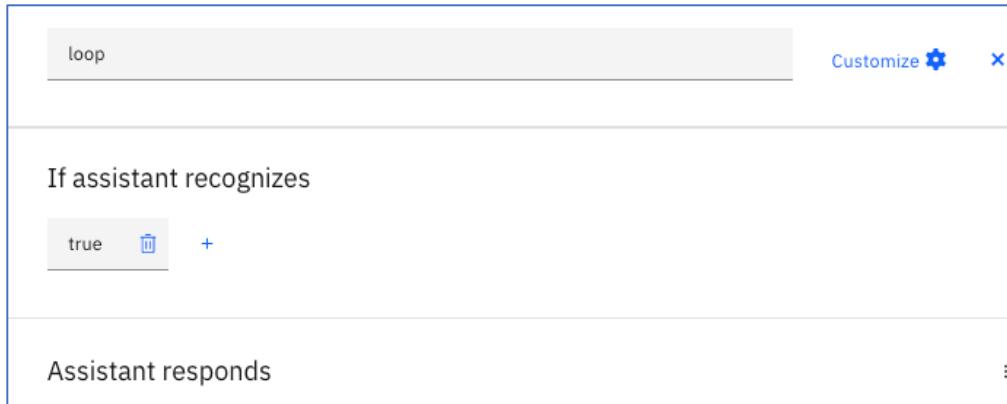
Condition: !\$restaurant — or — !\$mealtime +

Assistant responds:

If assistant recognizes	Respond with
1 !\$restaurant	What kind of restaurant or wh ⚙️ -
2 !\$mealtime	When would you like to go to t ⚙️ -

23. Loop management

1. Select **Request Restaurant** node and add a new node below it by clicking **Add node below**.
2. Fill in the node as below:
name: *Loop*
condition: *true*
3. Delete the response type.



24. Navigation node management

Now we are going to manage the navigation between nodes of the branch.

As we want to check *all required information in one time*, we will use **Jump to**.

1. To do this, we will add **Jump to** and evaluate condition:

from **Capture restaurant** node to **Capture mealtime** node

from **Capture mealtime** node to **Request restaurant** node.

If we don't get all required data, then the bot prompts a question, it waits the answer and returns to the node which evaluates them (**Return responses** node).

2. To do this we will add **Jump to** and **wait for user input** then evaluate condition

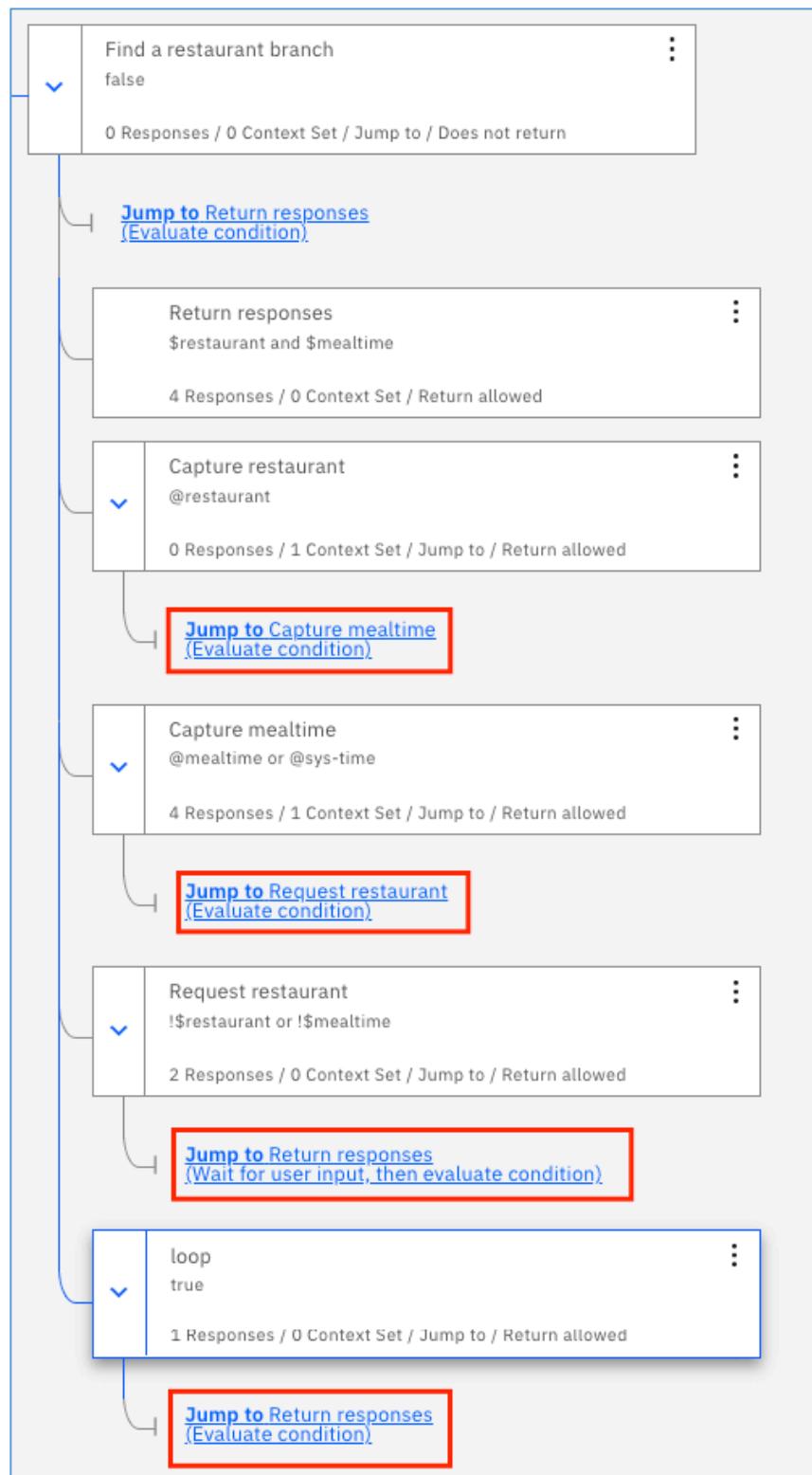
from **Request restaurant** node to **Return responses** node.

3. To finish we will add **Jump to** and evaluate condition

from **Loop** node to **Return responses** node.

Note: if assistant recognizes (condition) = evaluate condition

Now you should have this branch:

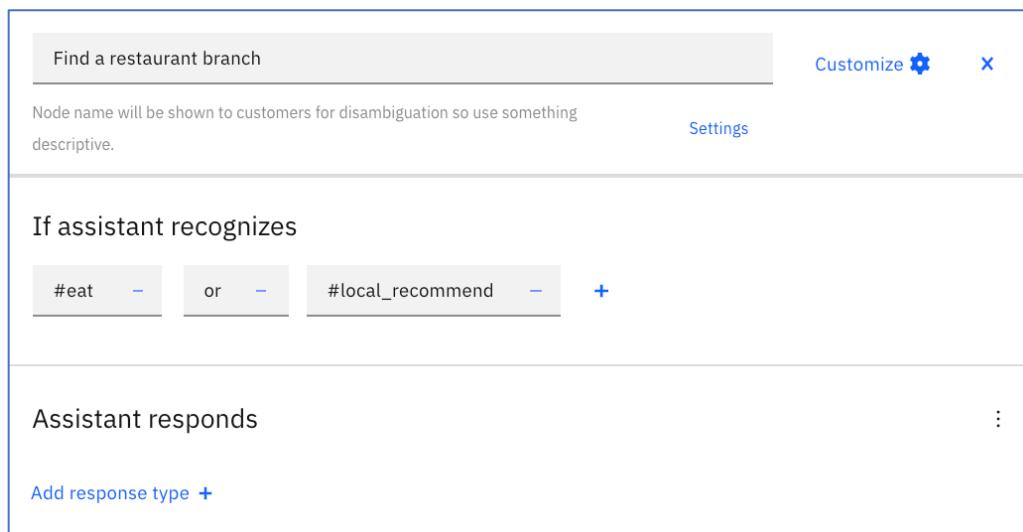


25. Test your branch

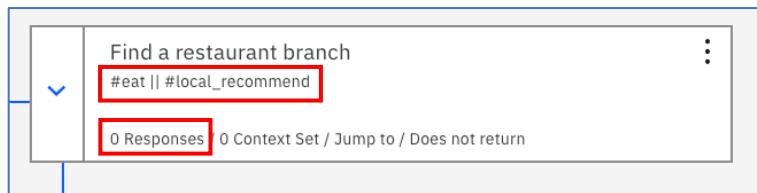
I remind you, this branch should be leveraged by 2 nodes triggered by the intent `#eat` and `#local_recommend`. Their behaviours can be different.

For the test, we will simplify the dialog.

1. Select **Find a restaurant branch** node.
2. Replace the condition `false` with `#eat or #local_recommend`
3. If you had a response, remove the empty response by clicking on the **remove** icon.



Now we should have:



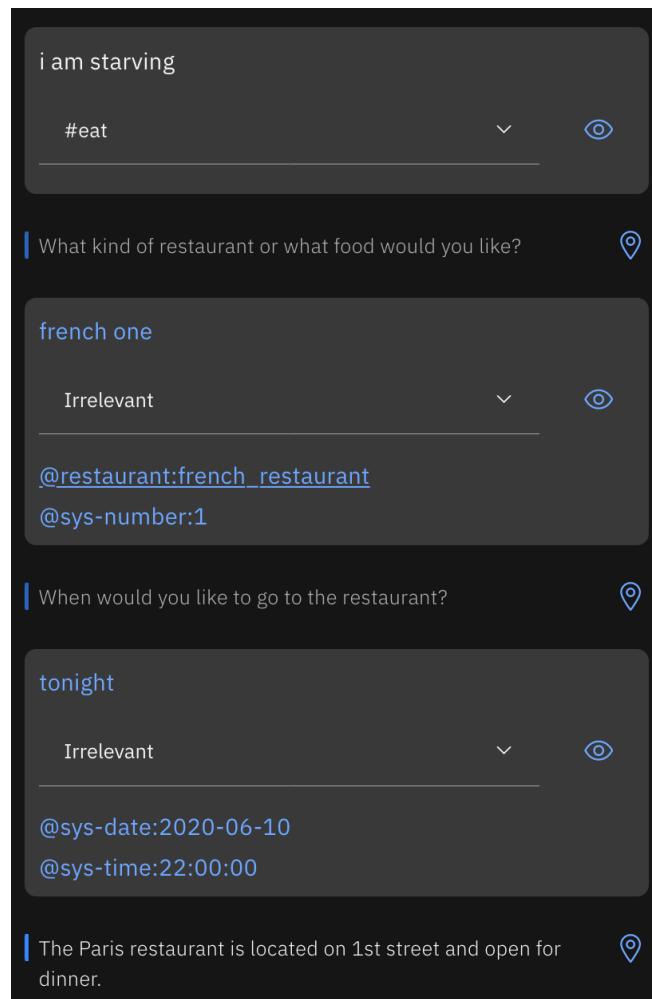
4. Open **Try it out** panel

5. Type in consecutively

I am starving

French one

tonight



Watson works as expected and requests the missing information.

6. Click **Clear**.

7. Type in: *I am looking for a restaurant right now*

8. Type in: *sushi*

I am looking for a restaurant right now

#eat

@places:restaurant
@hotel_amenity:hotel restaurant
@sys-date:2020-06-10
@sys-time:16:05:54

What kind of restaurant or what food would you like? 📍

sushi

#local_recommend

@restaurant:japanese_restaurant

The Tokyo restaurant is located on 2nd street and open for dinner. 📍

Watson determines when you want to eat and requests the missing information

9. Click **Clear**.

10. Type in *I want to eat French for dinner tomorrow*

I want to eat French for dinner tomorrow

#local_recommend

@restaurant:french_restaurant
@mealtime:dinner
@sys-date:2020-06-11

The Paris restaurant is located on 1st street and open for dinner. 📍

You can continue your test. Don't forget to click **Clear** between each test to reinitialize the context variable.

Enhance Find a restaurant branch

(optional)

In this section, we will do some enhancements to the Find a restaurant branch.

If your user looks for a restaurant several times during the conversation, the `$restaurant` and `$mealtime` variables must be re-initiated to take into account his new choices. This can be done by the application or by Watson Assistant. In this lab, we are going to reinitiate the context variable in the WA Dialog.

Also, if your user doesn't provide the right information, Watson will repeat the question in this branch again and again. So, you must use the counter context variable in the branch to stop the loop.

26. Initialize context variables

1. Select the node **Find a restaurant branch** to edit it
2. Open the context editor in this node (now you know how to do it) and fill in the context variables like this:

context Variable: `counter`

context Value: `0`

context Variable: `restaurant`

context Value: `null`

context Variable: `mealtime`

context Value: `null`

The screenshot shows the Watson Assistant interface for editing the 'Find a restaurant branch' node. At the top, there's a header with the node name and options to 'Customize' or 'X'. Below the header, a note says 'Node name will be shown to customers for disambiguation so use something descriptive.' with a 'Settings' link. The main area is divided into sections: 'If assistant recognizes' and 'Then set context'. Under 'If assistant recognizes', there are three input fields: '#eat', 'or', and '#local_recommend'. Under 'Then set context', there's a table where context variables are mapped to values. The table has two columns: 'Variable' and 'Value'. There are three rows:

Variable	Value
counter	0
restaurant	null
mealtime	null

27. Avoid infinite loop

1. Select **Return Responses** node and add a new node below by clicking **Add node below**.

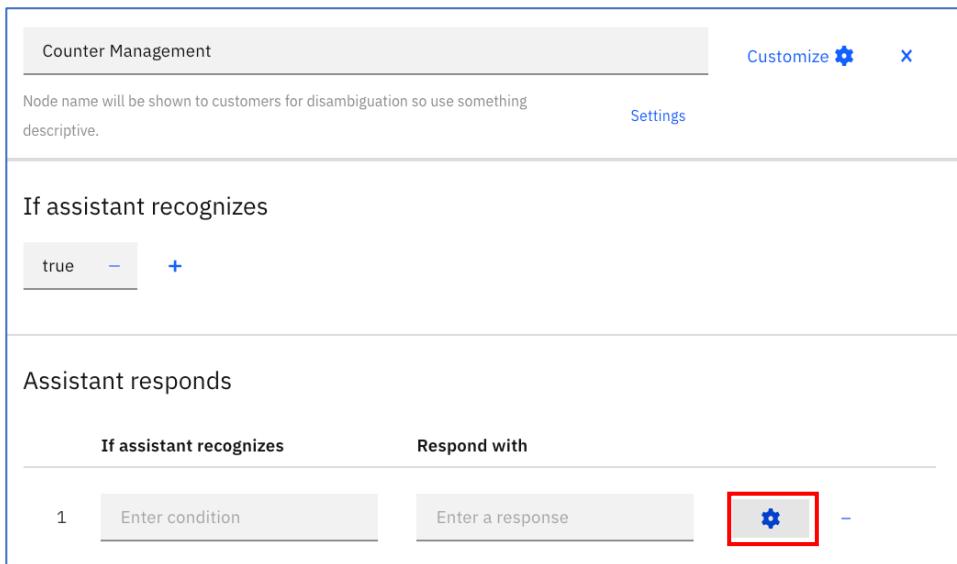
2. Fill the node as below

name: *Counter Management*

condition: *true*

3. Click **Customize** and switch on **Multiple conditioned responses**.

4. On the first row, click on the configure (wheel icon).



5. Click on the 3 dots menu and open the context editor.

6. Fill the node as below to reinitialize the counter when we get an expected input.

resp1 condition: *@restaurant*

resp1 context Variable: *counter*

resp1 context Value: *0*

7. At the bottom, click on **Default to node settings** and select **Jump to**.

The screenshot shows the 'Configure response 1' dialog box. In the 'Then set context' section, there is a table with 'Variable' and 'Value' columns. A row for 'counter' with a value of '0' is highlighted with a red box. Below this, there is an 'Add variable +' button. In the 'Assistant responds' section, there is a 'Text' dropdown and an 'Enter response text' input field. A note says 'Response variations are set to sequential. Set to random' with a 'Learn more' link. Below this, there is an 'Add response type +' button. In the 'Then assistant should' section, a dropdown menu is open, showing 'Default to node settings ~' (highlighted with a red box) and 'Default to node settings' and 'Jump to' as options.

8. Select **Capture restaurant** node.

9. Select the **condition** option. You should get:

The screenshot shows the 'Then assistant should' section with a dropdown menu. The 'Jump to' option is selected, and the value 'Capture restaurant_(Evaluate condition)' is shown next to it.

10. Click **Save**.

11. Repeat the steps 4 to 9 for the 3 other responses with the following parameters:

- to reinitialize the counter when we get an expected input.

resp2 condition: `@mealtime || @sys-time`

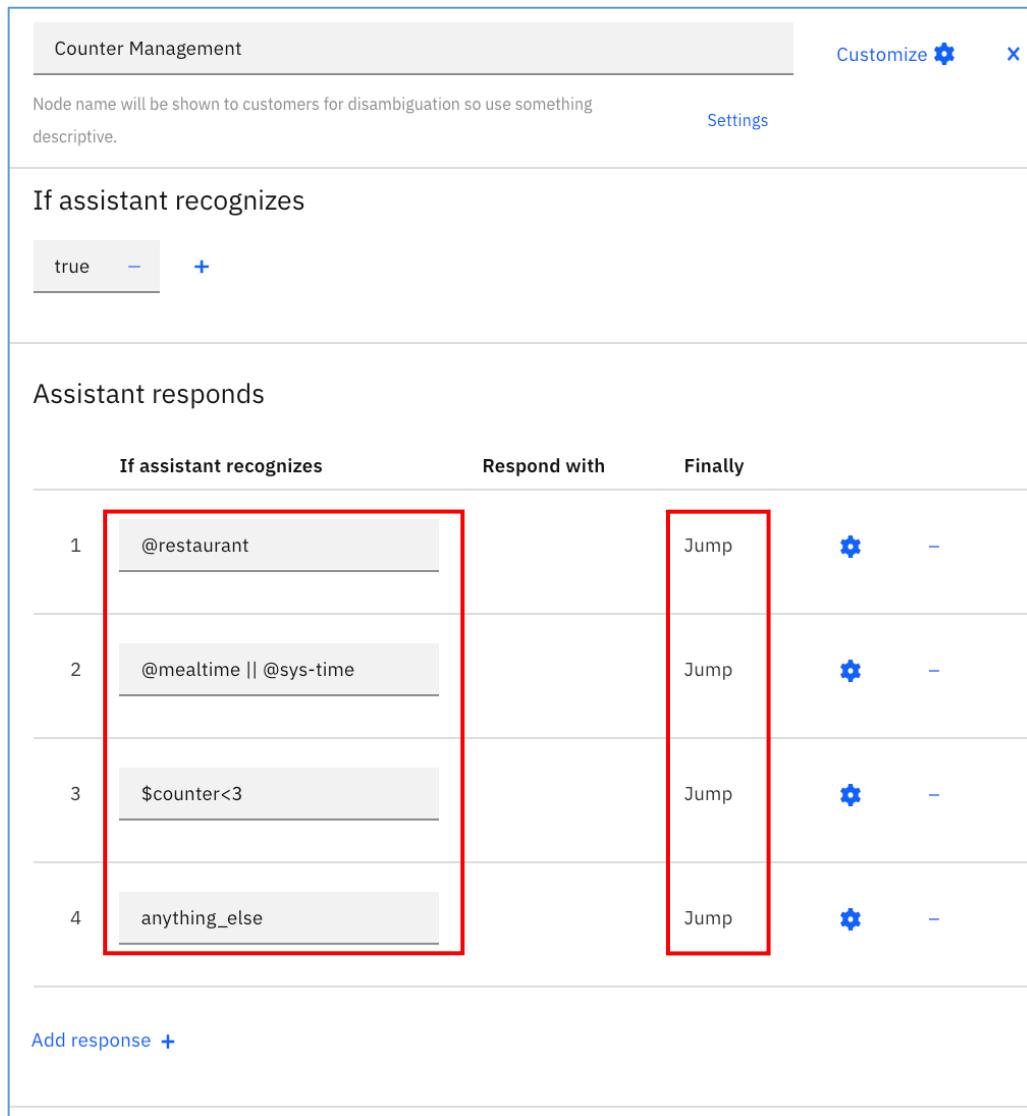
resp2 context Variable: `counter`

resp2 context Value: `0`

resp2 Jump to condition node: `Capture mealtime`

- to increment the counter when we don't get any expected input.
 resp3 condition: `$counter<3`
 resp3 context Variable: `counter`
 resp3 context Value: "`<? context.counter +1 ?>`"
 resp3 Jump to condition node: `Request restaurant`
- to close the loop after too many iterations.
 resp4 condition: `anything_else`
 resp4 context Variable: `mealtime`
 resp4 context Value: "nodef"
 resp4 context Variable: `restaurant`
 resp4 context Value: "nodef"
 resp4 Jump to condition node: `Return responses`

Now, you should have in your node:



The screenshot shows the 'Counter Management' node configuration. At the top, there's a header with 'Counter Management', a 'Customize' button, and a close 'X'. Below the header, a note says 'Node name will be shown to customers for disambiguation so use something descriptive.' with a 'Settings' link. The main area is divided into two sections: 'If assistant recognizes' and 'Assistant responds'.

If assistant recognizes: A table with four rows. Row 1 contains '@restaurant'. Row 2 contains '@mealtime || @sys-time'. Row 3 contains '\$counter<3'. Row 4 contains 'anything_else'. The first three rows are highlighted with a red box.

	If assistant recognizes	Respond with	Finally
1	@restaurant		Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
2	@mealtime @sys-time		Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
3	\$counter<3		Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
4	anything_else		Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>

Assistant responds: A table with four rows, each containing a 'Jump' action. The entire column of 'Finally' actions is highlighted with a red box.

	If assistant recognizes	Respond with	Finally
1	@restaurant	Jump	Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
2	@mealtime @sys-time	Jump	Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
3	\$counter<3	Jump	Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>
4	anything_else	Jump	Jump <input checked="" type="checkbox"/> - <input type="checkbox"/>

At the bottom left, there's a 'Add response +' button.

With for the 3 last responses:

Configure response 2

If assistant recognizes

@mealtime — or — @sys-time — +

Then set context

Variable	Value
counter	0

Add variable +

Assistant responds

Add response type +

Then assistant should

Jump to [Capture mealtime \(Evaluate condition\)](#)

Configure response 3

If assistant recognizes

\$counter<3 — +

Then set context

Variable	Value
counter	"<? context.counter +1 ?>"

Add variable +

Assistant responds

Add response type +

Then assistant should

Jump to [Request restaurant \(Evaluate condition\)](#)

Configure response 4

If assistant recognizes

anything_else	-	+
---------------	---	---

Then set context

Variable	Value
mealtime	"nodef"
restaurant	"nodef"

[Add variable +](#)

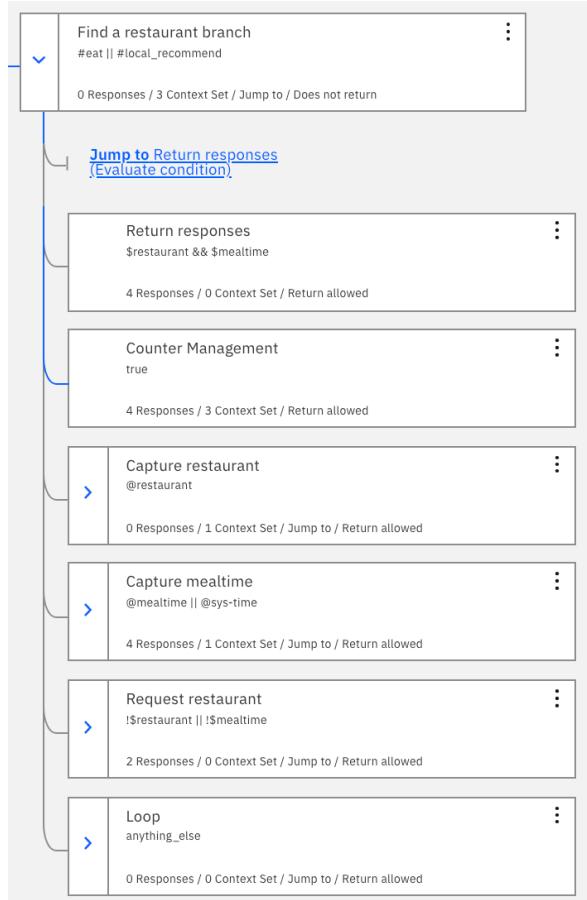
Assistant responds

[Add response type +](#)

Then assistant should

Jump to	Return responses (Evaluate condition)	x
---------	---	---

Now you should have the following branch:



28. Test the enhancements

1. Open Try it out panel, click on **Clear**

2. Enter successively

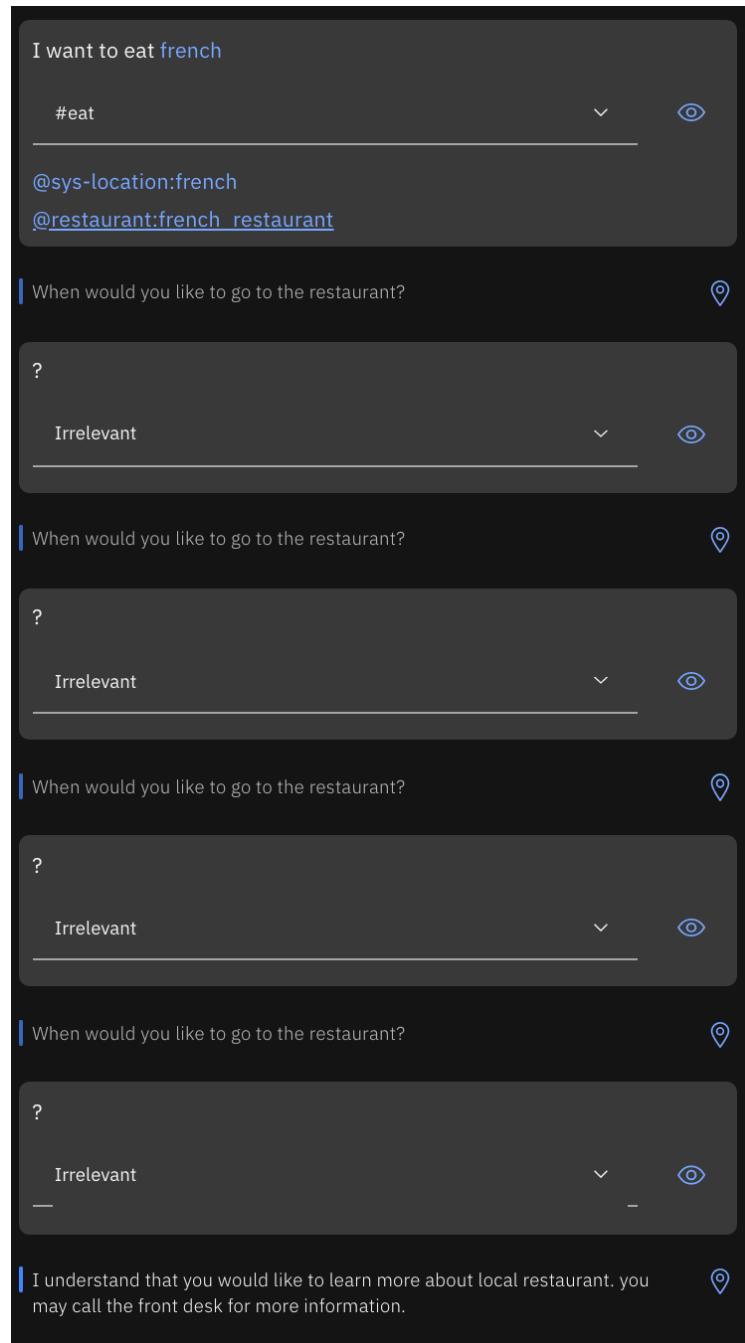
I want to eat french

?

?

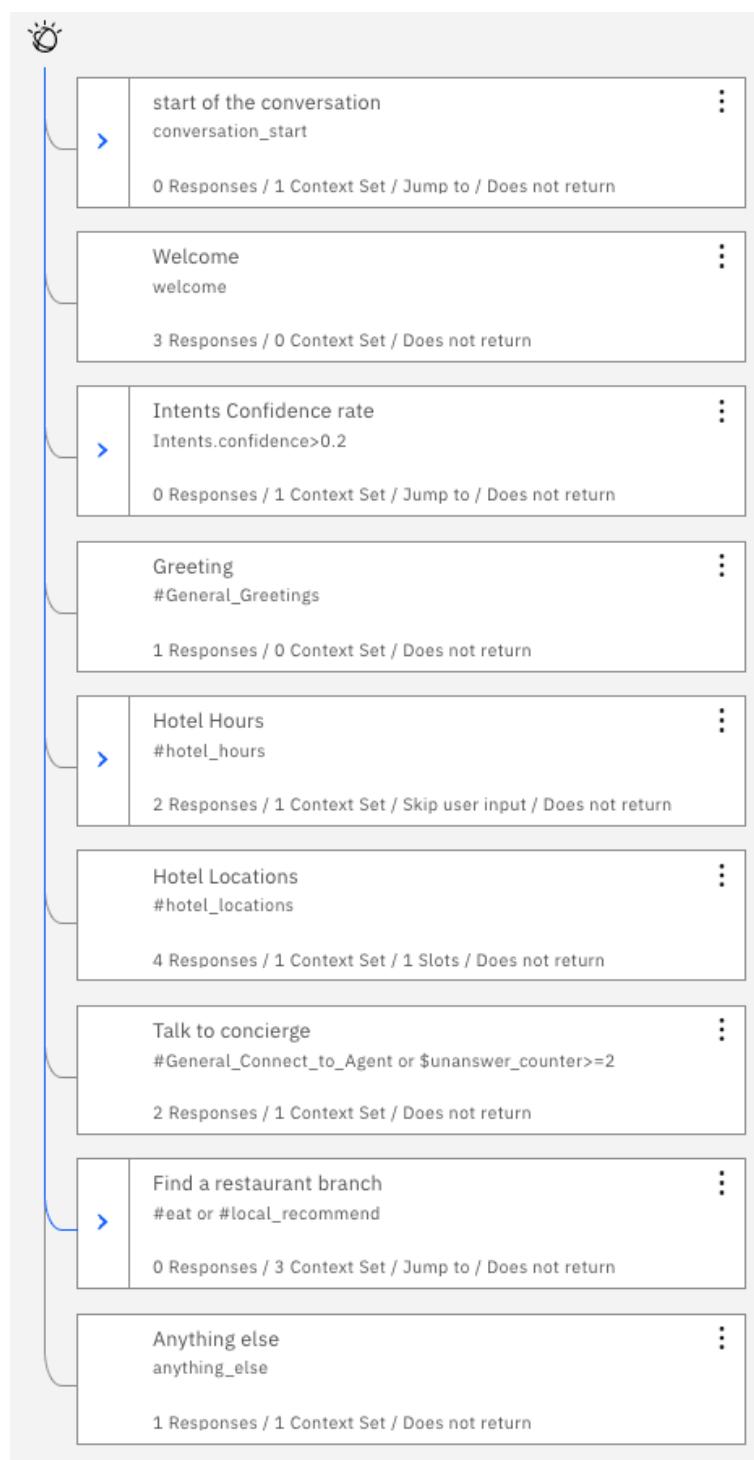
?

?



Watson is trying to determine when you want to eat and requests the missing information. Since you did not provide it, Watson suggested that you engage with the front desk.

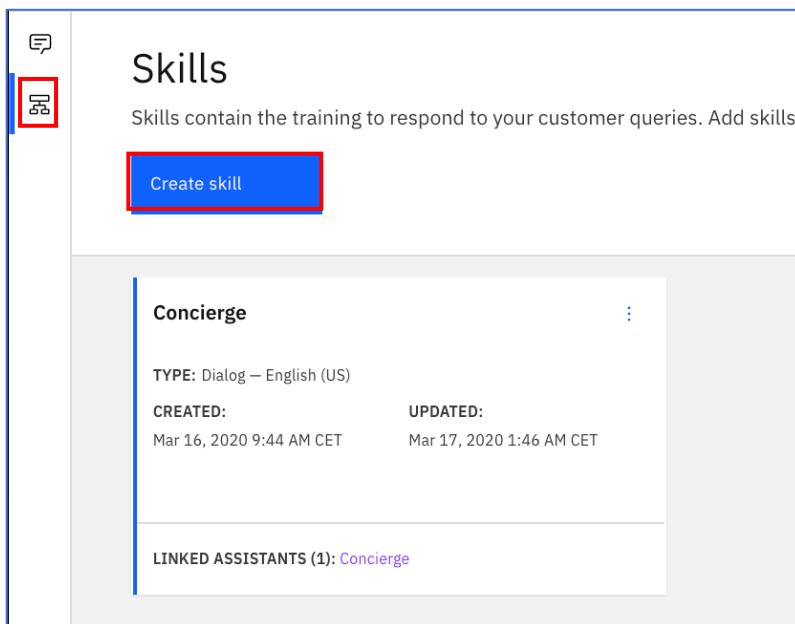
The final dialog:



Note: if you want to compare your dialog with the one we created, feel free to import into your Assistant our Skill as it was after the end of this exercise. Our Skill is in the file [skill-My-Concierge-Skill-Lab2part1.json](#).

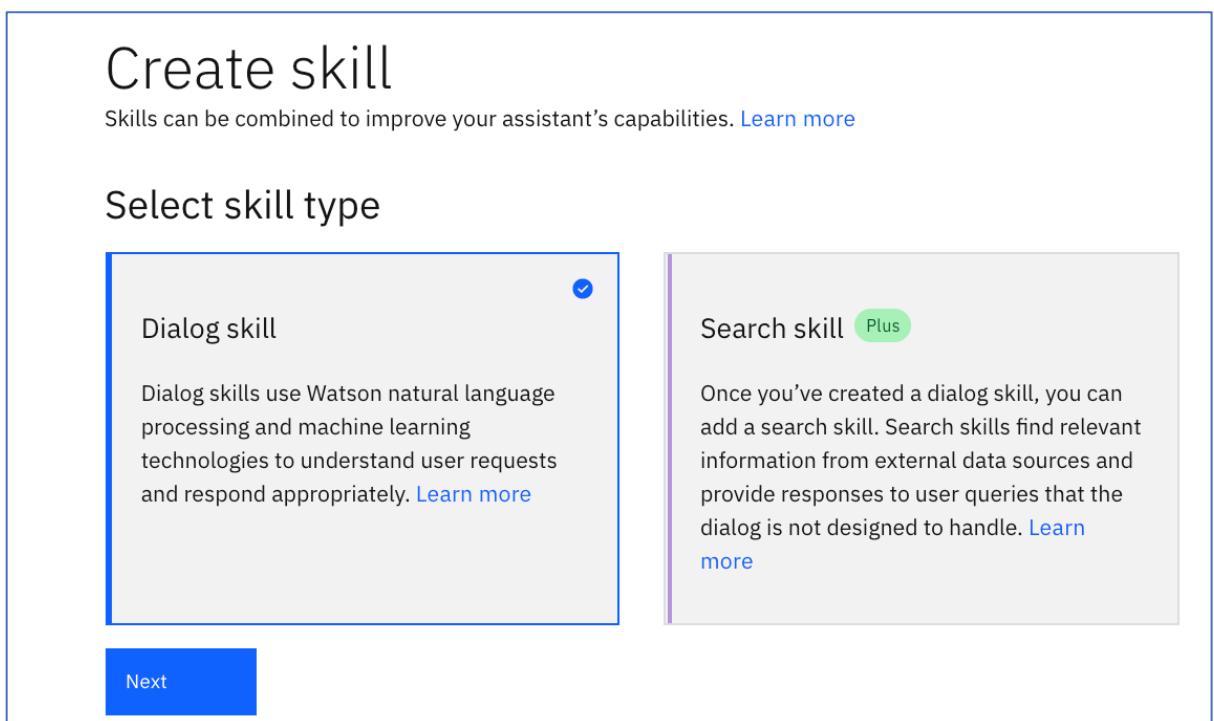
You can import a Skill into your Assistant this way:

1. On Skills tab, click Create skill.



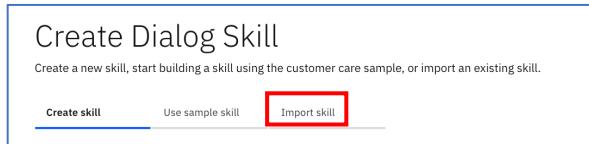
The screenshot shows the 'Skills' page in the Watson Assistant interface. At the top left is a navigation icon with a speech bubble and a gear. Below it is a red-bordered 'Create skill' button. The main area displays a single skill card for 'Concierge'. The card includes the skill name, type ('Dialog – English (US)'), creation date ('Mar 16, 2020 9:44 AM CET'), update date ('Mar 17, 2020 1:46 AM CET'), and a link to 'LINKED ASSISTANTS (1): Concierge'.

2. Select **Dialog skill** and click **Next**.

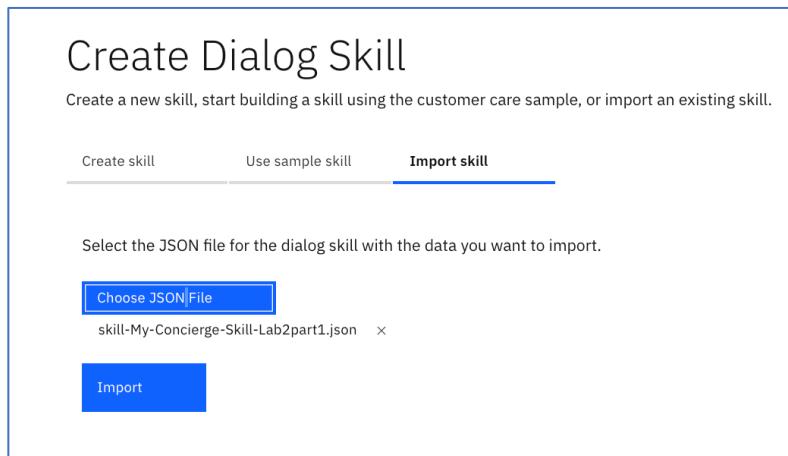


The screenshot shows the 'Create skill' wizard. The title is 'Create skill' and a sub-instruction says 'Skills can be combined to improve your assistant's capabilities. [Learn more](#)'. The section 'Select skill type' contains two options: 'Dialog skill' (selected) and 'Search skill'. The 'Dialog skill' box contains a description: 'Dialog skills use Watson natural language processing and machine learning technologies to understand user requests and respond appropriately.' Below the boxes is a 'Next' button.

3. Click on **Import skill**.



4. Click **Choose JSON File**, then select the file with the exported skill.



5. Click **Import**.

6. Now, you can see the skill you just imported on your **Skills** page.

The screenshot shows the 'Skills' page. On the left, there is a sidebar with a 'Skills' icon (highlighted with a red box) and a 'Create skill' button. The main area lists skills: 'Concierge' and 'My Concierge Skill'. The 'My Concierge Skill' card is highlighted with a red box. Both cards show details like type (Dialog – English (US)), creation date (Mar 16, 2020 9:44 AM CET / Mar 17, 2020 2:00 AM CET), update date (Mar 17, 2020 1:46 AM CET / Mar 17, 2020 2:00 AM CET), and linked assistants (1: Concierge).