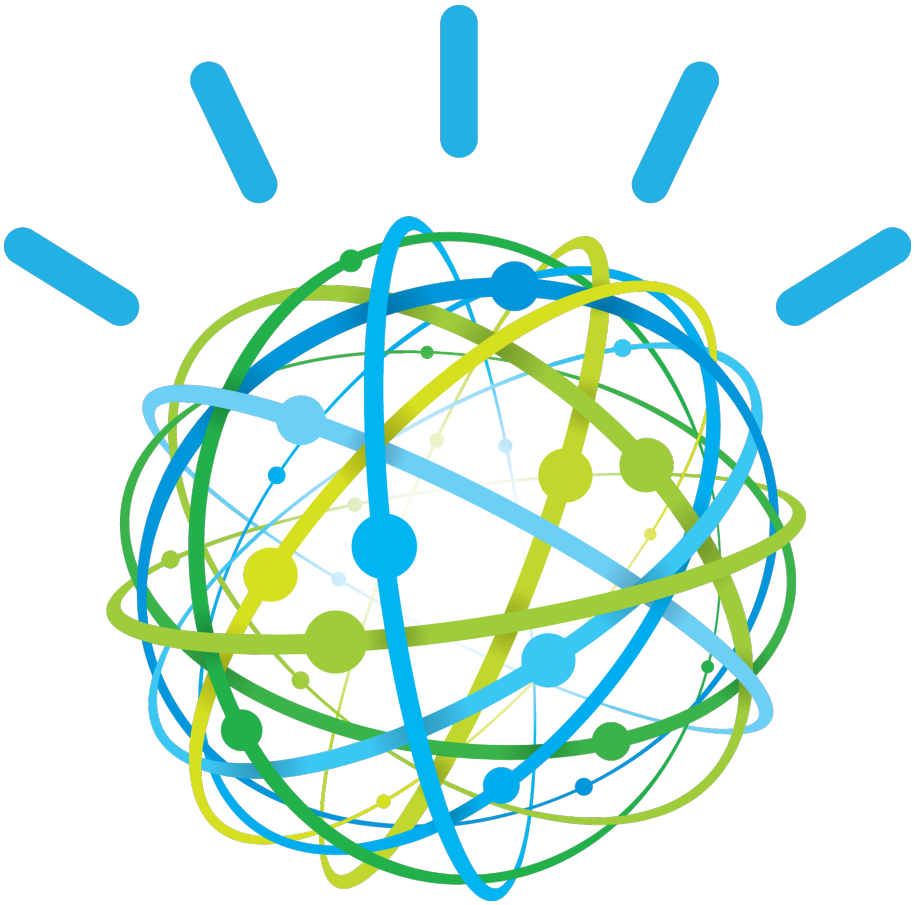
Lab 2 part 1: Building a Dialog

IBM Watson Assistant



Lab Instructions

Laurent Vincent

Content

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

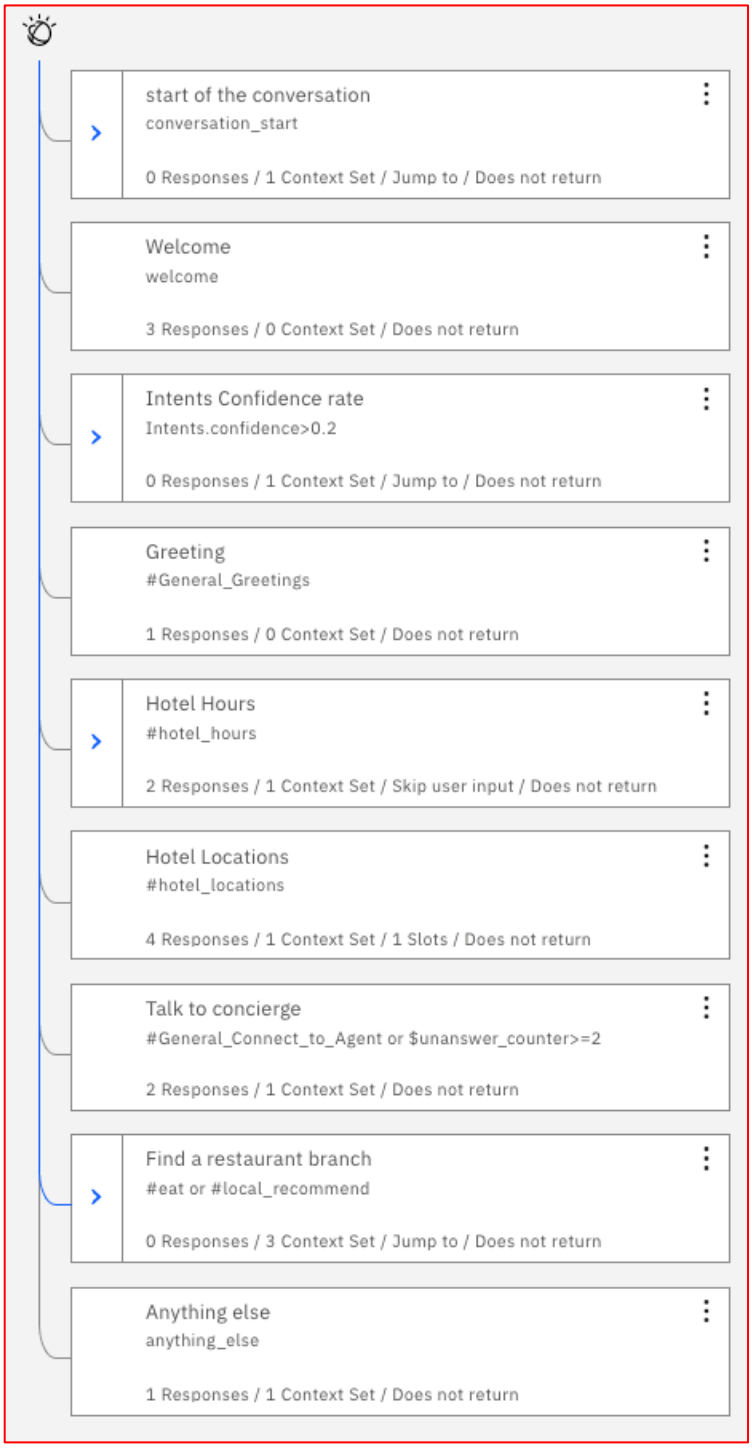
4. Scenario

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:

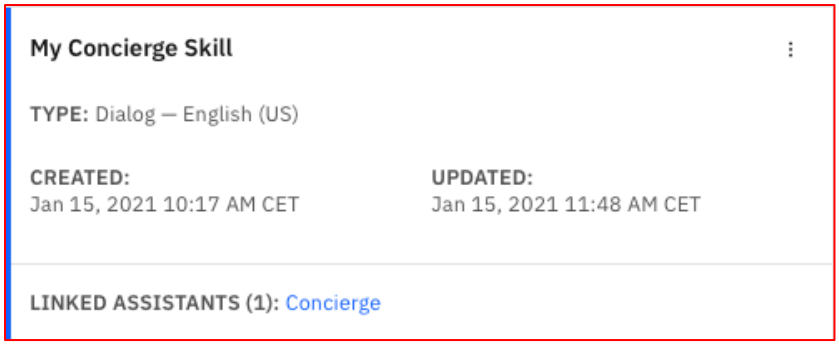


Update the default settings

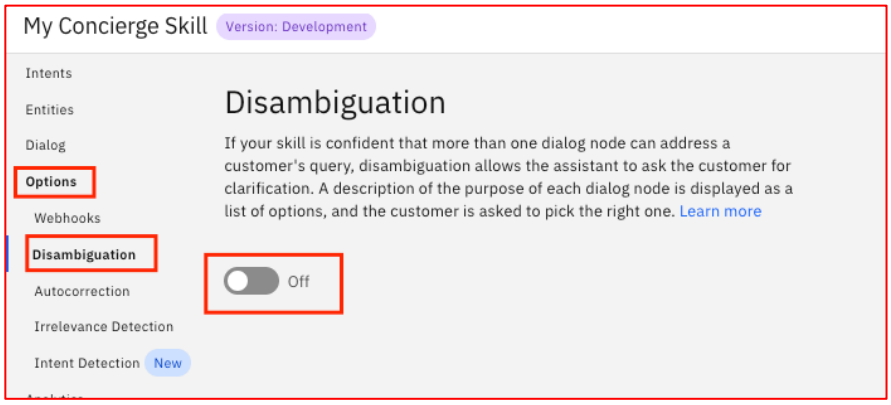
6. Update default settings

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

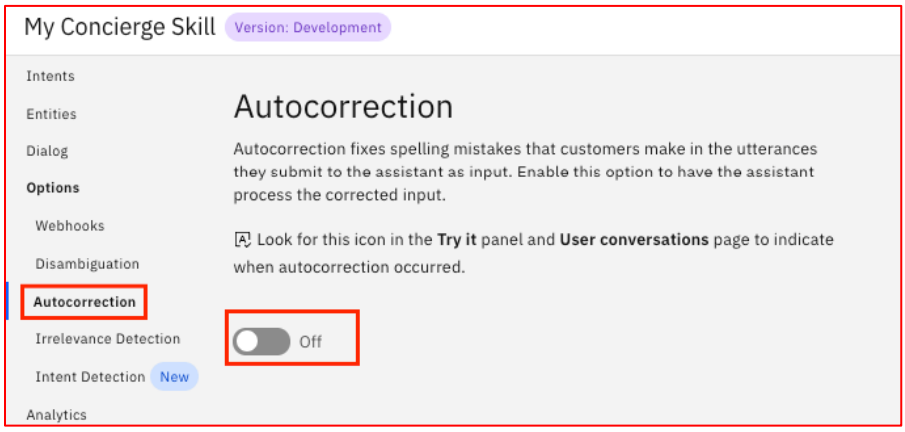
1. Go back to your skill, by clicking the **Skills** menu, then click on your **My Concierge skill** tile.



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



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



Building more natural answers

7. Update Greeting answers

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

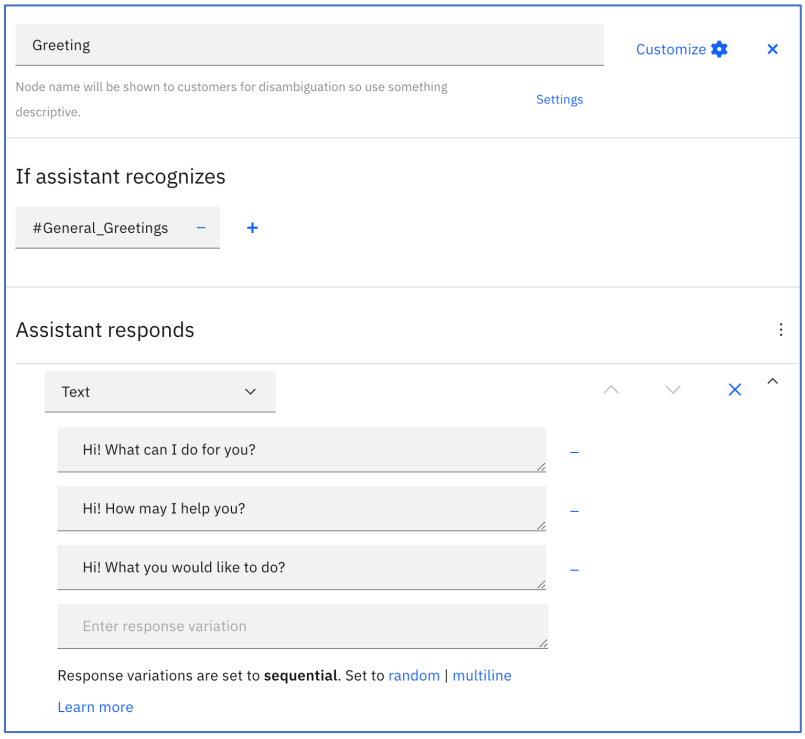
- update the dialog context
- provide a text response to the user (as you did in previous lab)
- change the flow of the dialog by using a **Jump to** action.

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.

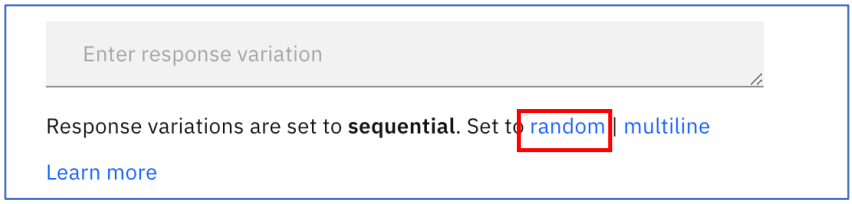
1. Open the **Dialog** page.
2. You are going to update the **Greeting** node and add several answers to make it look less mechanic and more human. Open the **Greeting** node.
3. Add the responses as shown in the table below. When you add a response and press enter, a new empty response field will open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

Field	Value
Assistant responds	*Hi! How may I help you?*
Assistant responds	*Hi! What you would like to do?*
Assistant responds	*Hi! What can I do for you?*



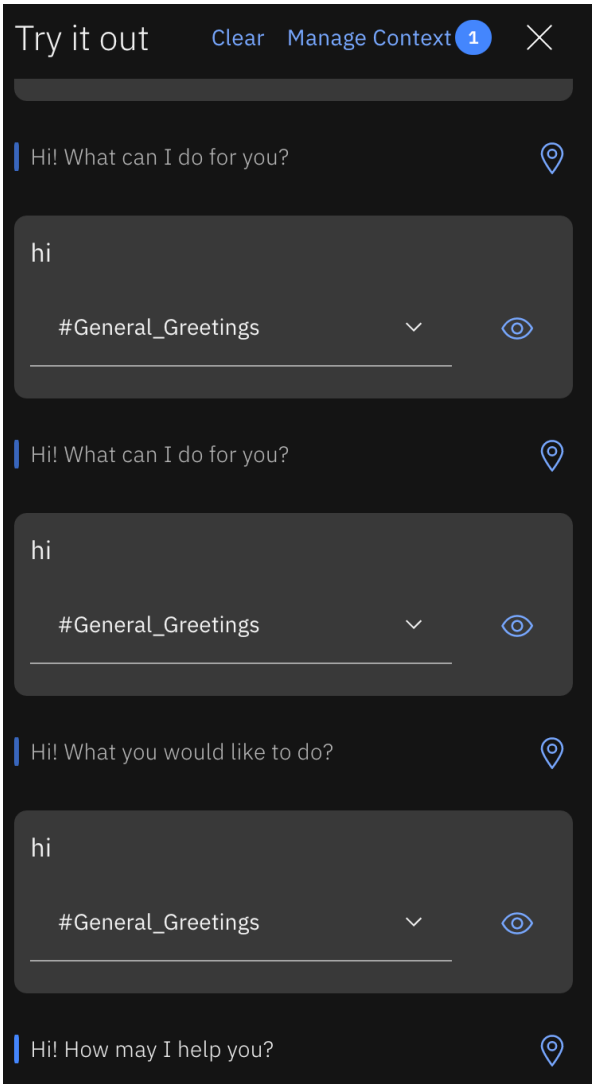
If you want to, you can decide to return these responses sequentially (default) or randomly. In this case, we want to change it to randomly.

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:

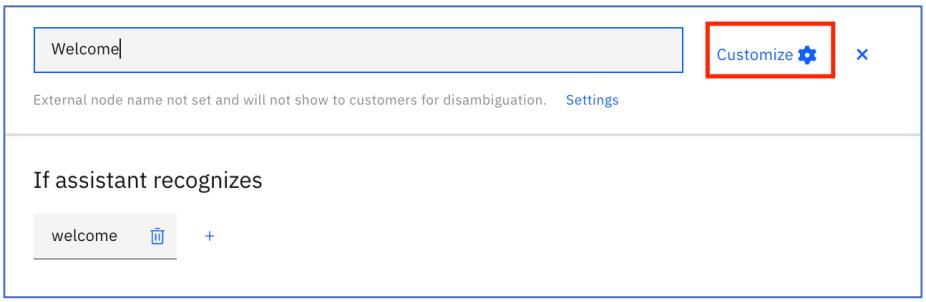
- Engage the user to continue with conversation
- Include information about the role of the solution and its name (if needed)
- Consider using contextual information such as the time or user's name
- Initialize the context variables if required.

9. Add contextual details

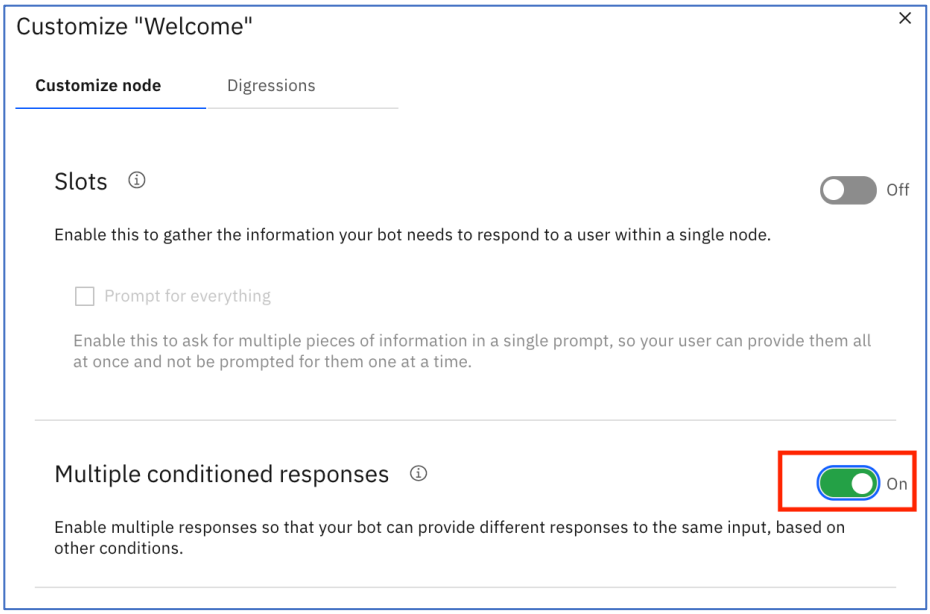
You are going to update the Watson's welcome according to the time of the day.

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



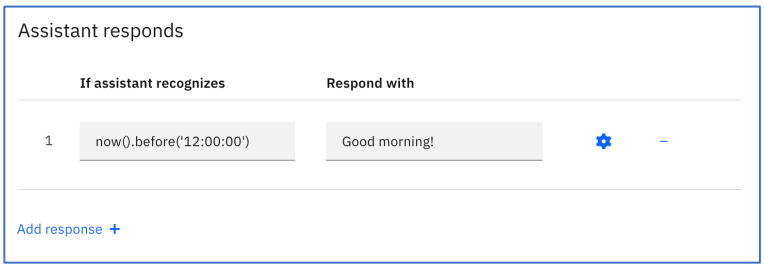
3. Enable **Multiple responses**



4. Click **Apply**

5. Replace the response *Hi! I am Watson*… with *Good morning!*
6. Enter the following condition *now().before('12:00:00')*

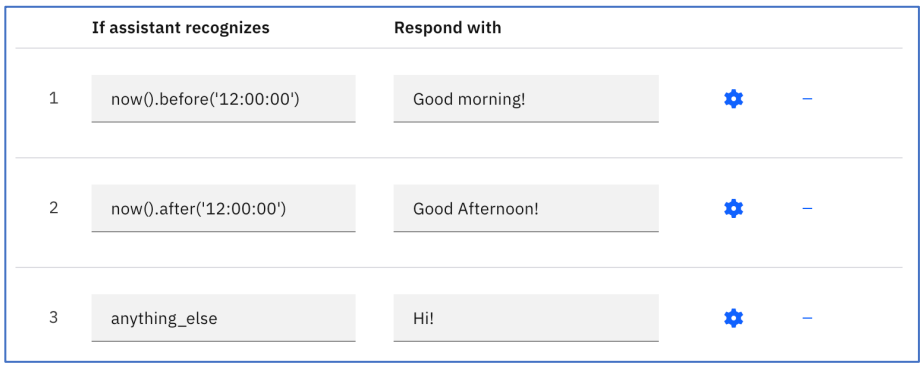
You should get the result below. You are using the method 'before()' to evaluate if the current time is before noon. That's why the answer is *Good morning*.



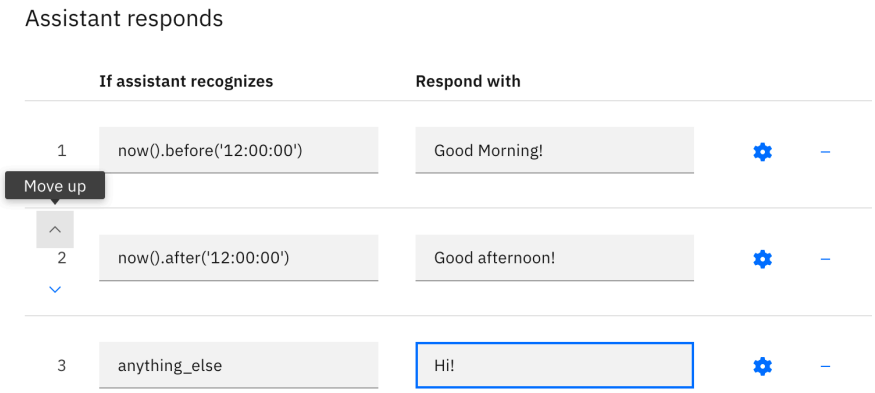
7. Click on **Add response.**

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!*

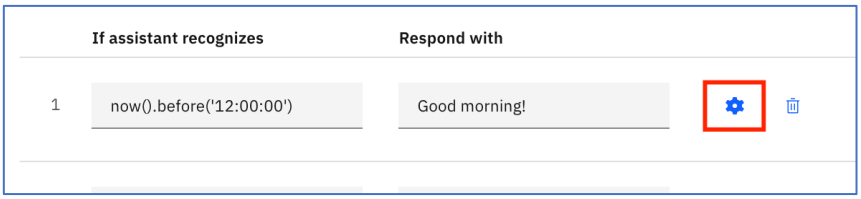
Now you should have:



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



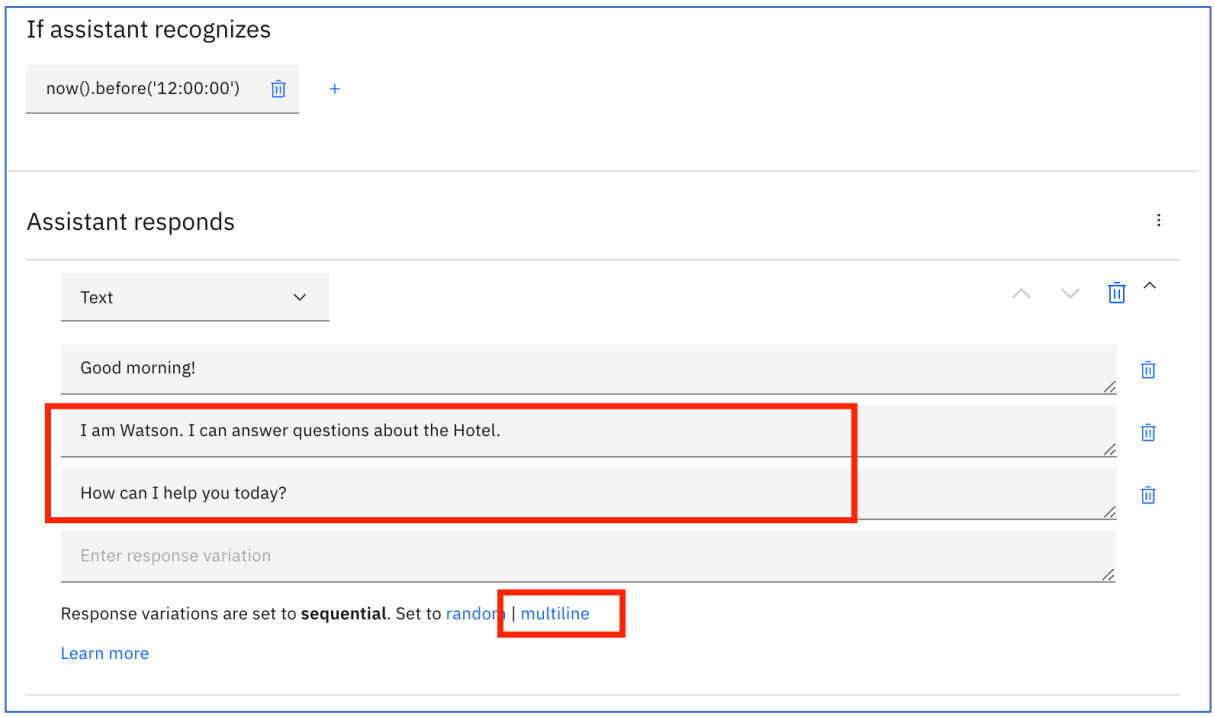
11. Click **Edit Icon** on the first conditioned response



12. Enter the following responses :

Field	Value
Assistant responds	*I am Watson. I can answer questions about the Hotel.*
Assistant responds	*How can I help you today?*

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



Then you should have this

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.

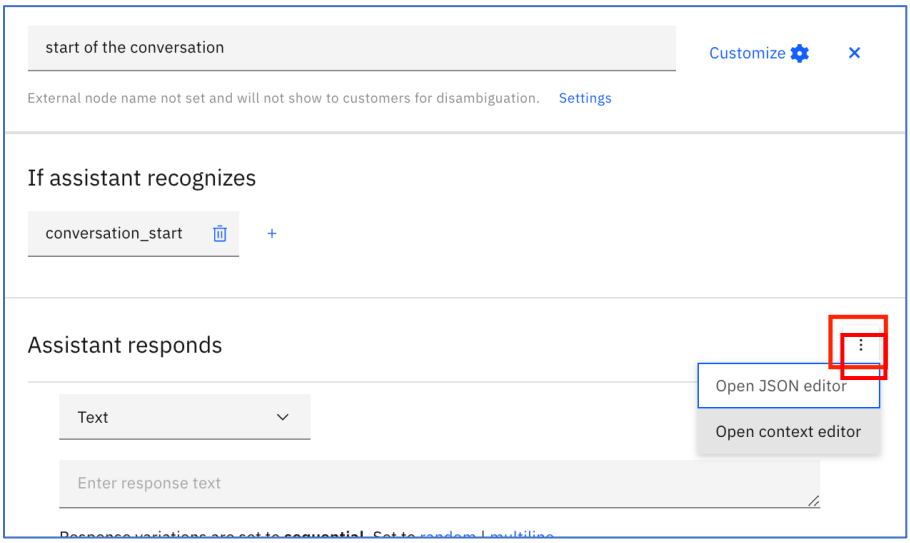
1. Select the **Welcome** node and add a child by clicking on the 3 dots icon and select **Add node above**.
2. Fill the node like this

Field	Value
Node name	*start of the conversation*
If assistant recognizes	*conversation_start*
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.

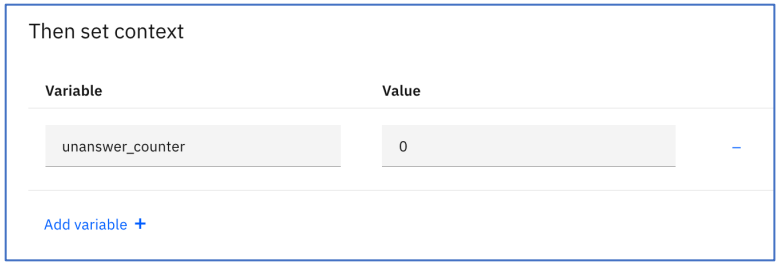
1. Open the **context editor**, by clicking the 3 dots menu, then select **Open context editor.**



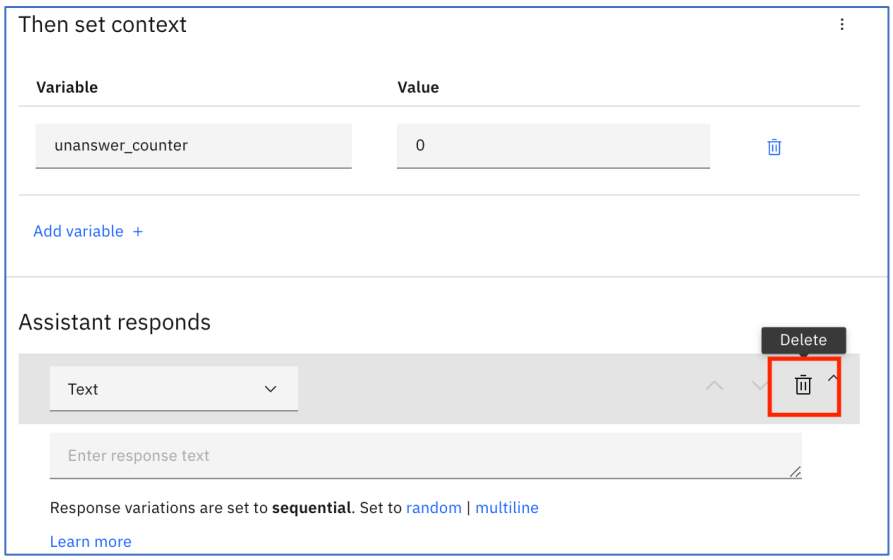
2. You will now specify the first variable, fill in the fields like this:

   Context variable : *unanswer_counter*

   Context value : *0*



3. Delete the empty response

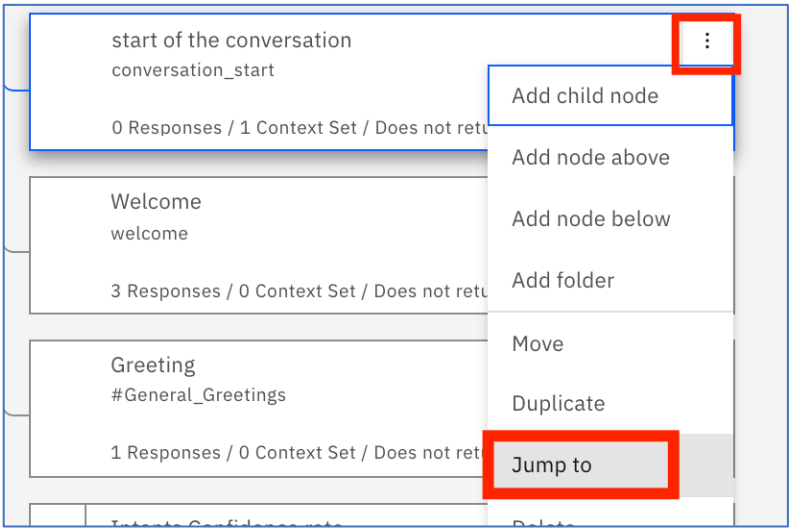


4. Close the edit page:

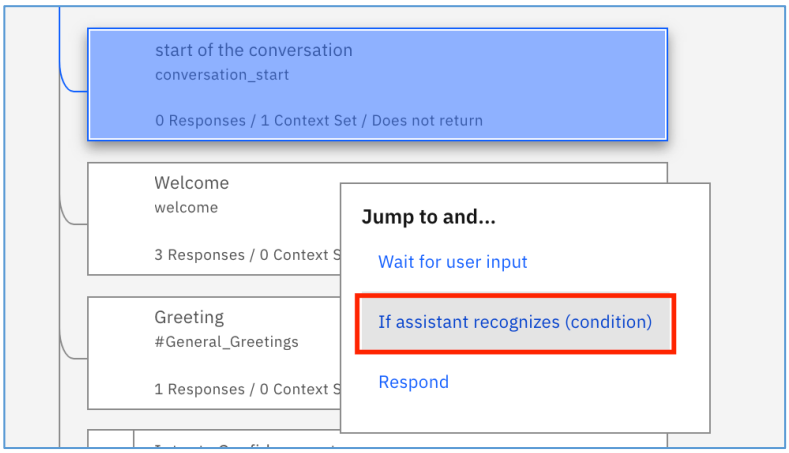
11. First “Jump to”

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.

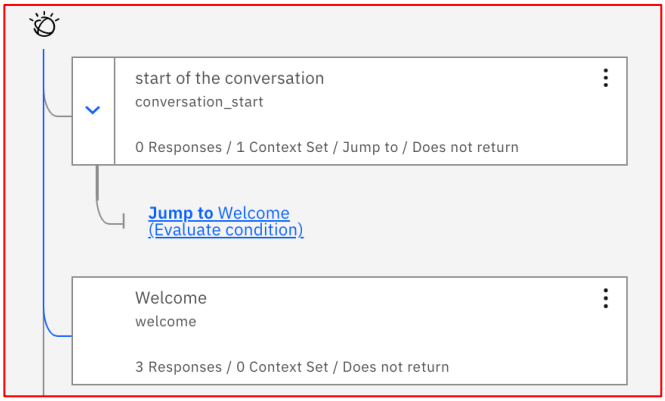
1. Select the **start of the conversation** node
2. In the **contextual** menu select **Jump to**



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



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

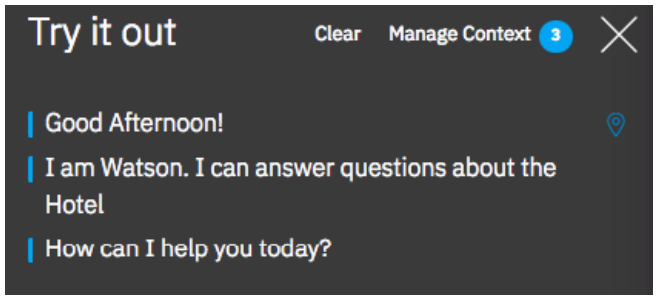


This design results in a dialog that works like this:

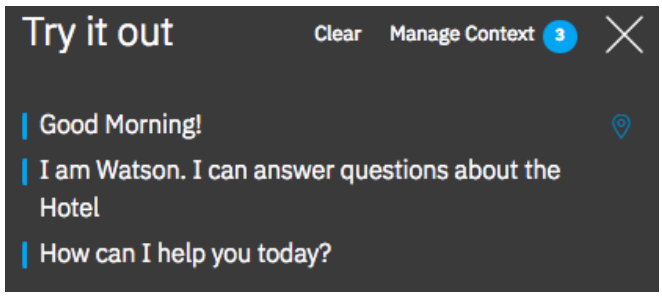
- Whatever the integration type, the *conversation_start* node is processed, which means any context variables that you define in it are initialized.
- In integrations where the assistant starts the dialog flow, the *welcome* node is triggered and its text response is displayed.
- In integrations where 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



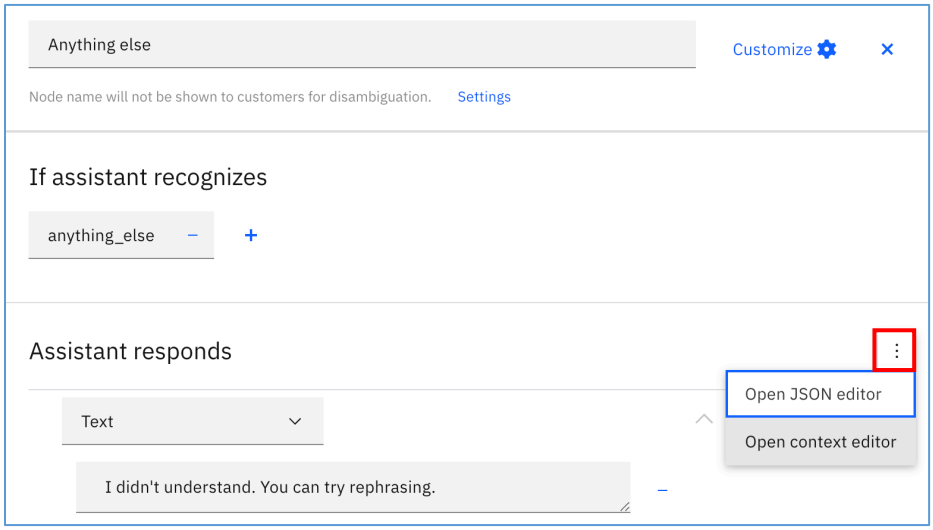
Human takeover

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.

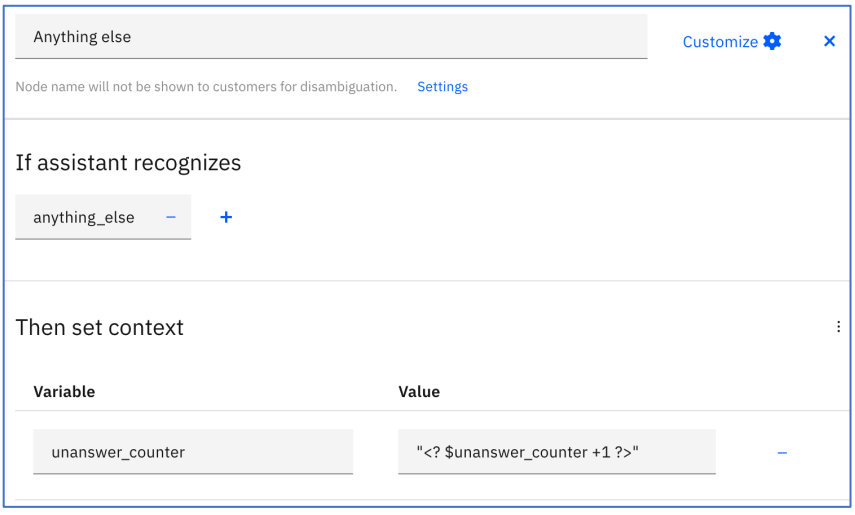
1. Select **Anything else** node to open its editor
2. Open the **context editor**



3. Fill it in like this

   Variable : *unanswer_counter*

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

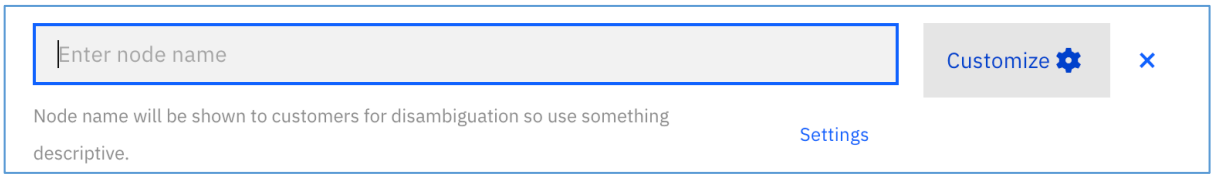


4. Close the **Anything else** node editor.

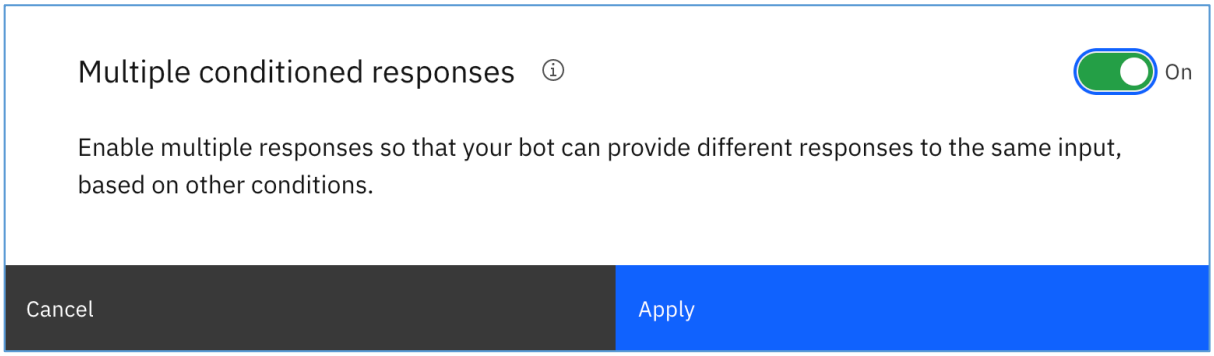
13. Talk to concierge node

1. Add a node above the **Anything else** node

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



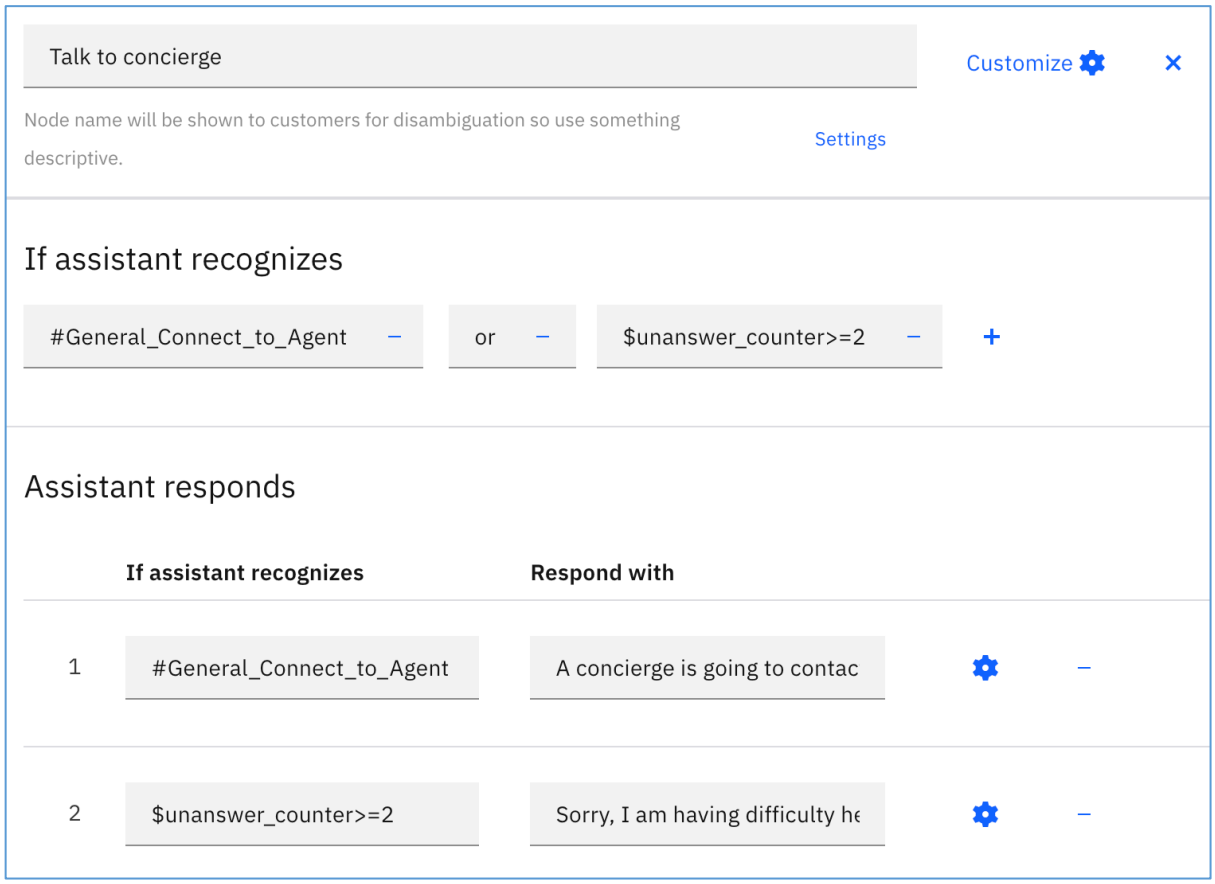
3. Activate **multiple responses** and click **Apply**:



4. Here are the parameters of your new node

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

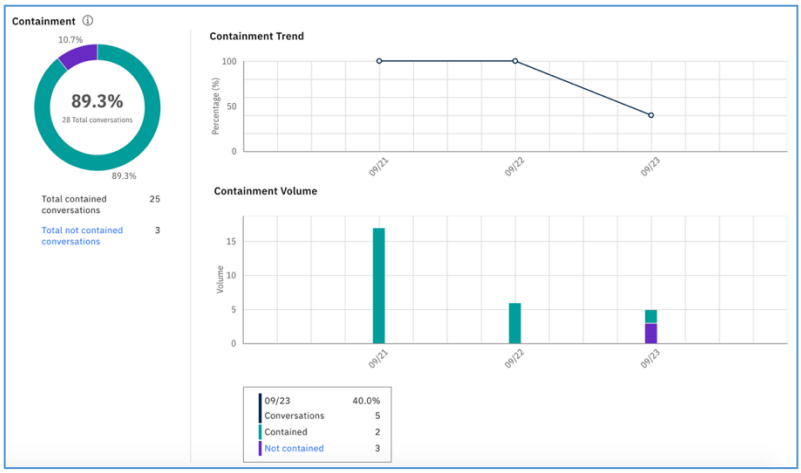
Your node should look like this:



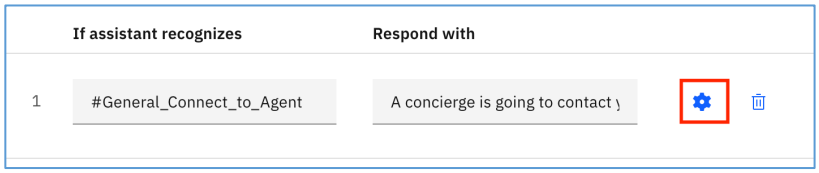
In this case, the response returned is a text. If you did an integration with a service desk solution, you could use the response type : "*Connect to human agent*". It enables to hand over to an operator of the service desk.

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**.

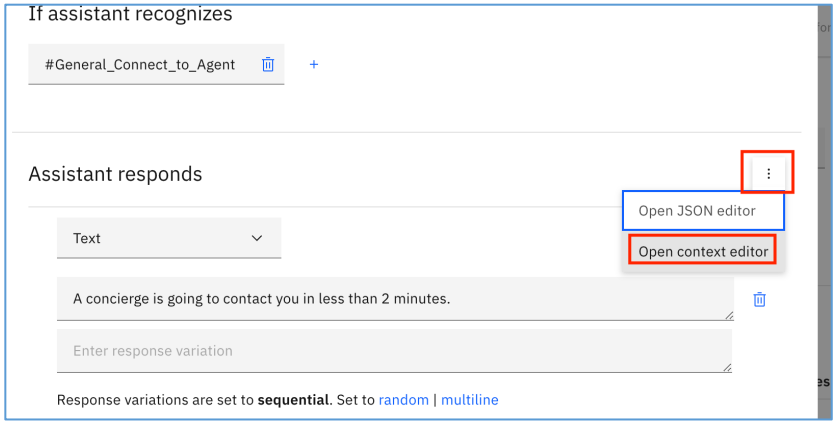
However, not all human interventions are transacted by using this response type. as you use an alternate method to deliver additional support, you must take additional steps to register the fact that the customer's need was not fulfilled by the assistant. so, you need to flag these types of redirects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



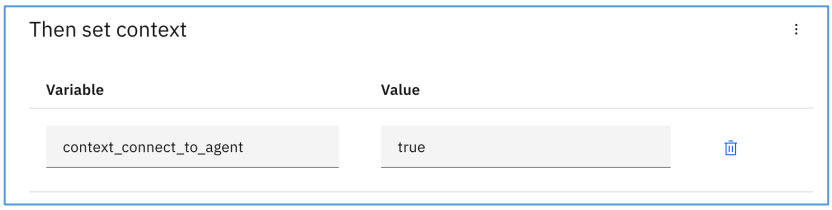
5. Click on the setting icon of the first conditioned response



6. Open context editor



7. Add the context variable *context_connect_to_agent* with the value *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%.

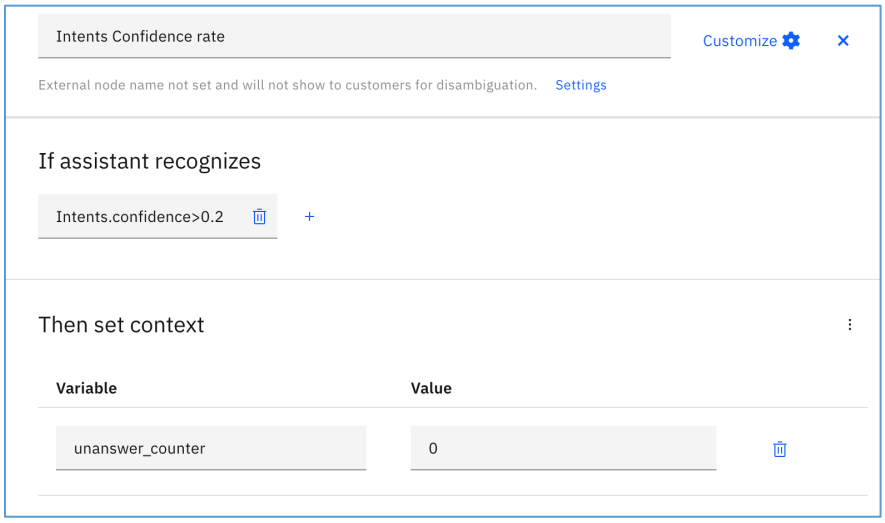
1. Add a node below the **Welcome** node
2. Fill the node as below

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

10. Open the nodes **context editor** (If you can't remember how, scroll up few pages).
11. Fill it in like this

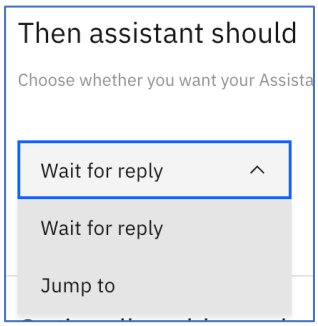
    Variable : *unanswer_counter*

    Value : *0*

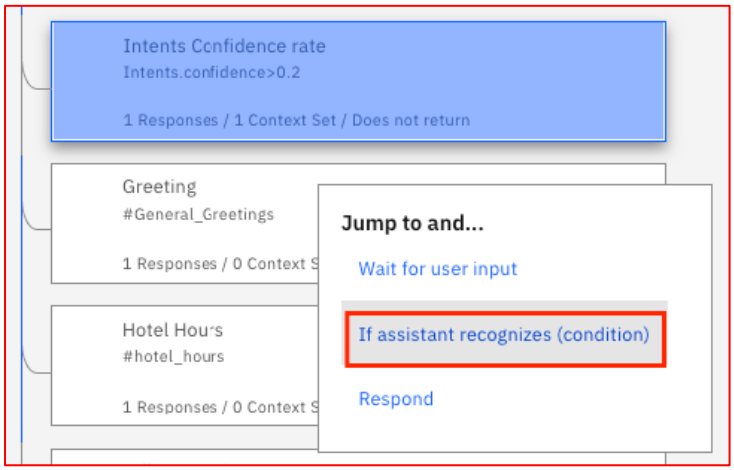


As we don't expect any interaction with the user in this node, the node must jump to the next existing node – the **Greeting** node.

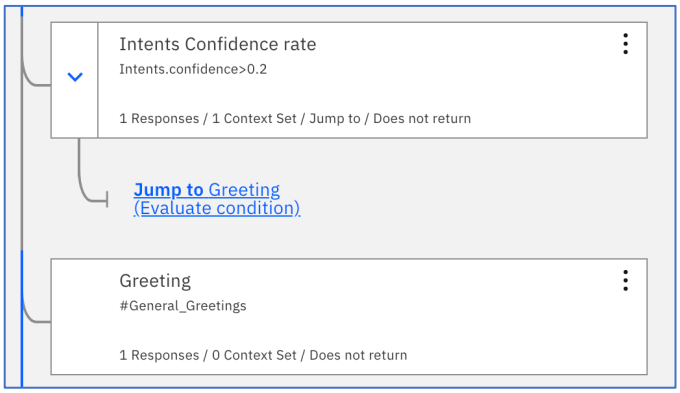
12. Click **Wait for reply** drop box (bottom of the editor page)



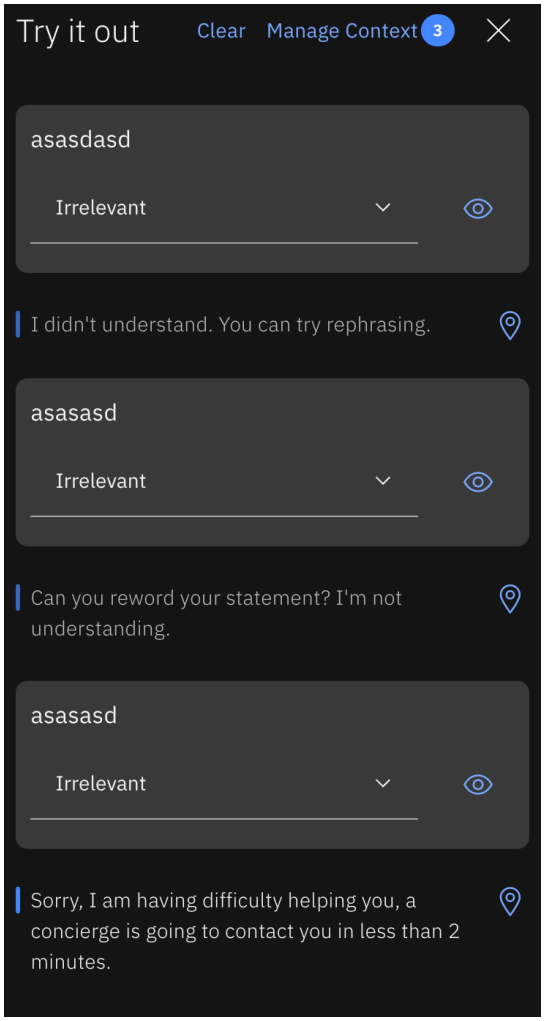
13. Select **Jump to** option
14. Select **Greeting** node, then select **If assistant recognizes (condition)**



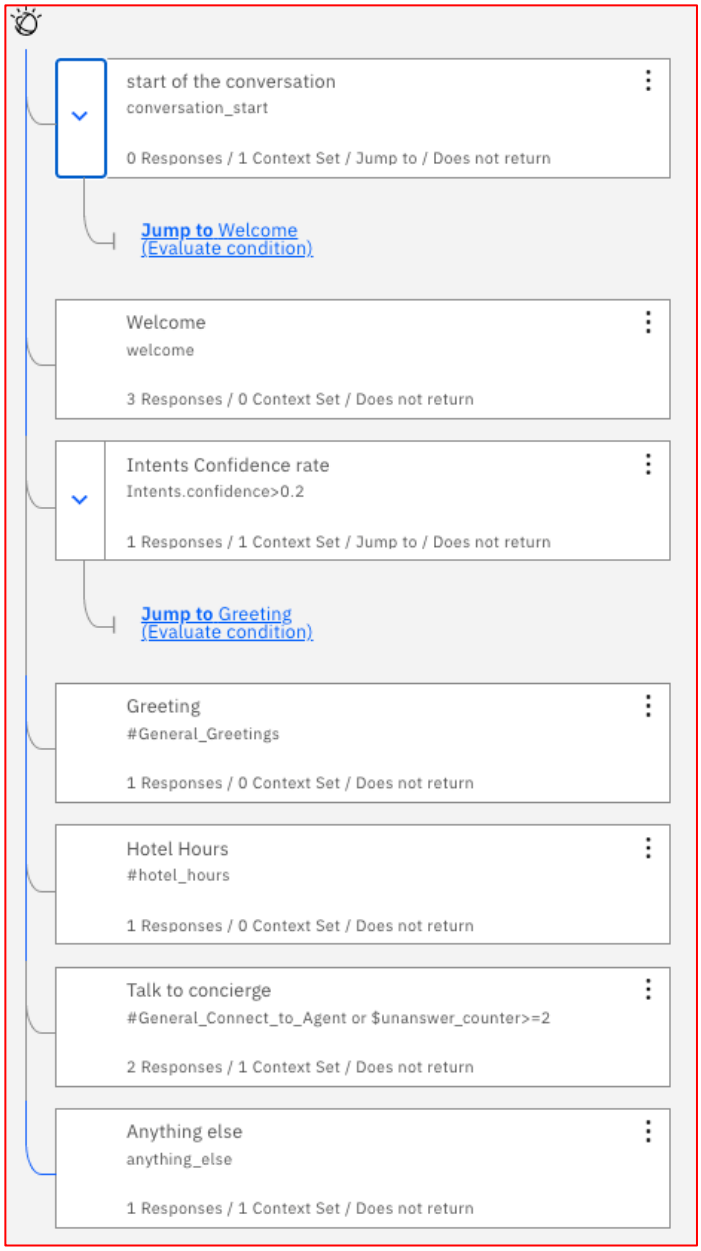
Your partial node tree should now look like this:



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.

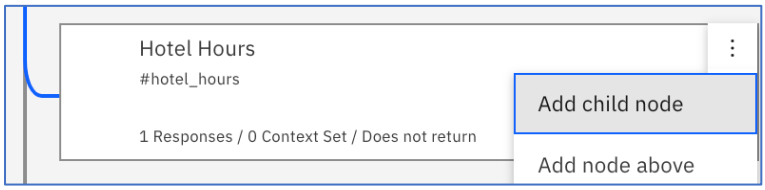


Hotel amenities management

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.

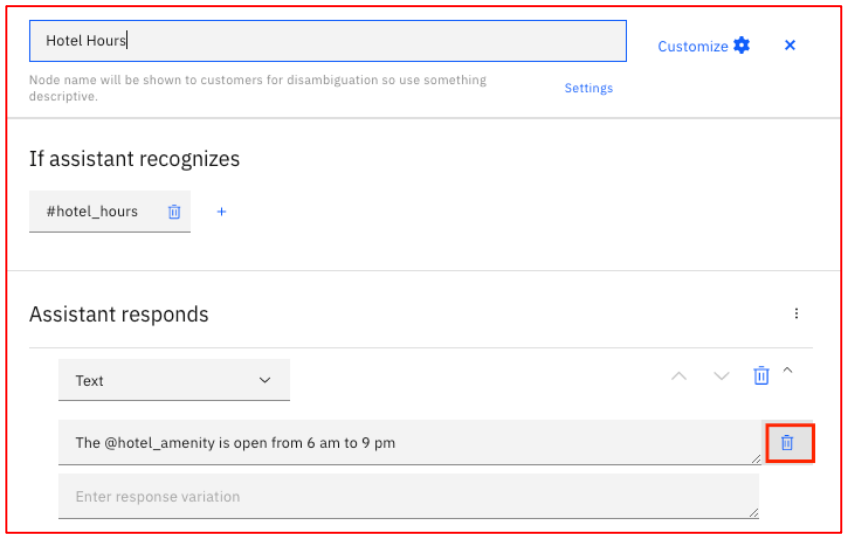
1. Click the **Hotel Hours** node to highlight it
2. Click on the contextual menu and click **Add child node**



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

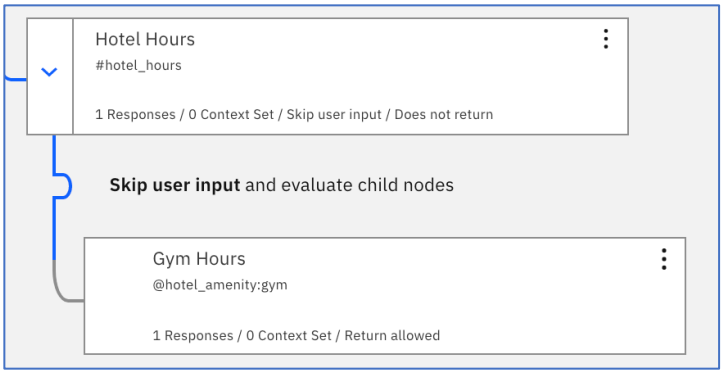
Field	Value
Node name	*Gym Hours*
If assistant recognizes	*@hotel_amenity:gym*
Assistant responds	*The @hotel_amenity.value is open from 5am to 11pm.*

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



6. 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

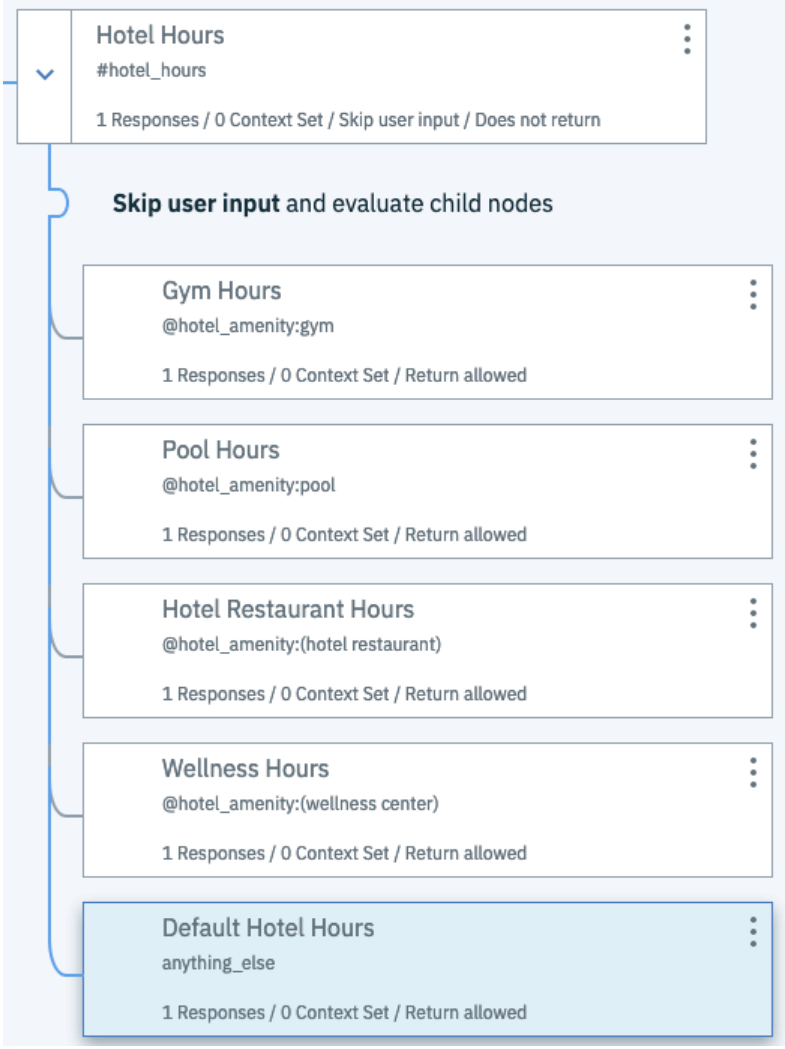
Field	Value
Node name	*Pool Hours*
If assistant recognizes	*@hotel_amenity:pool*
Assistant responds	*The @hotel_amenity.value is open from 5am to 8pm.*

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

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

Field	Value
Node name	*Default Hotel Hours*
If assistant recognizes	*anything_else*
Assistant responds	*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.*

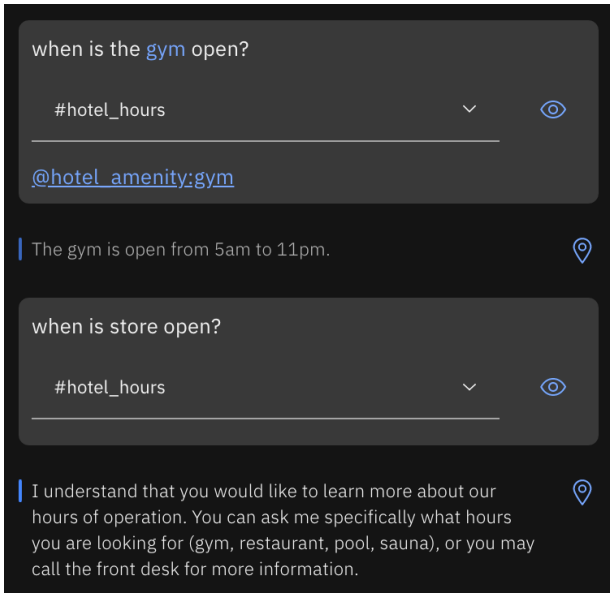
Your chat flow should look similar to this:



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

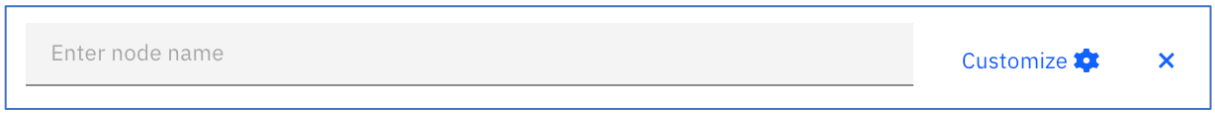


16. Single node with multiple conditions / responses

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

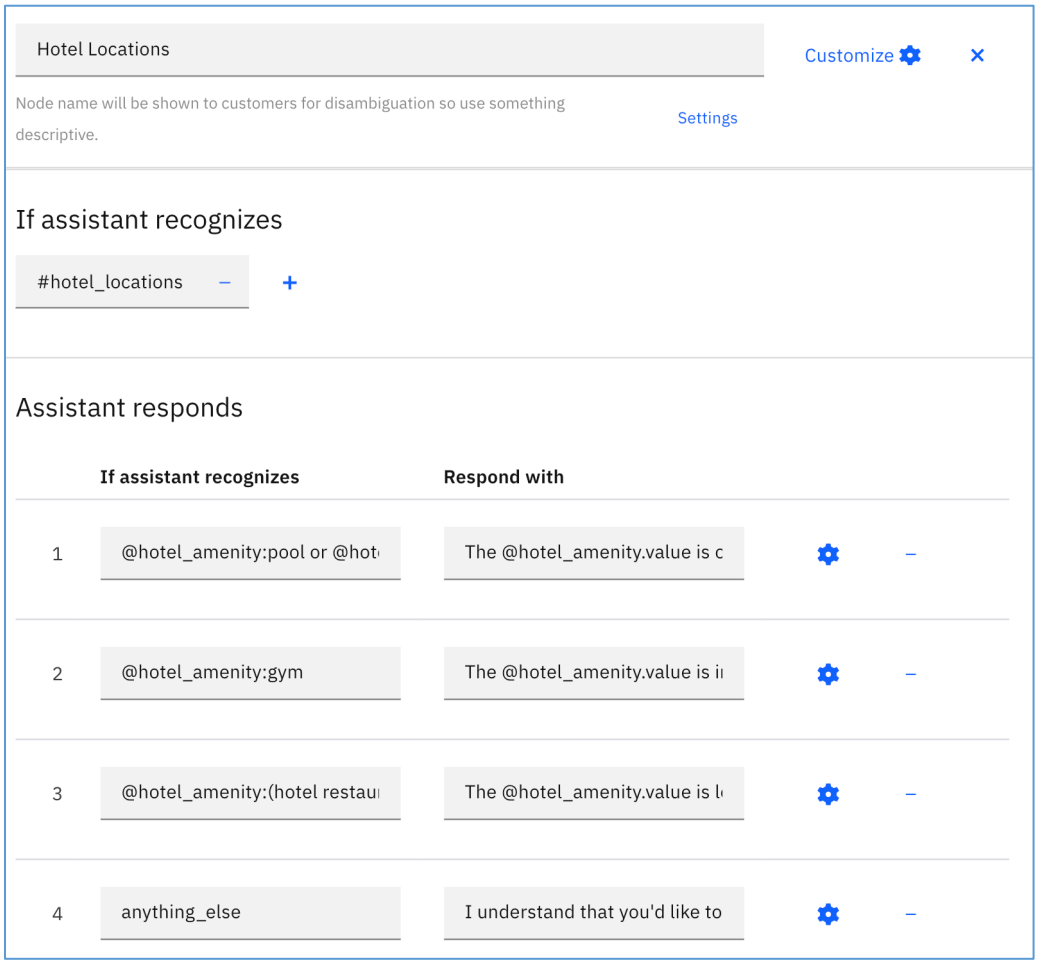
As you already created a such node for **Welcome**, instead of giving you specific step-by-step instructions for this section, you will be given a list of conditions with the paired responses. There is a screen shot of the final dialog node after the instructions.

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):



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

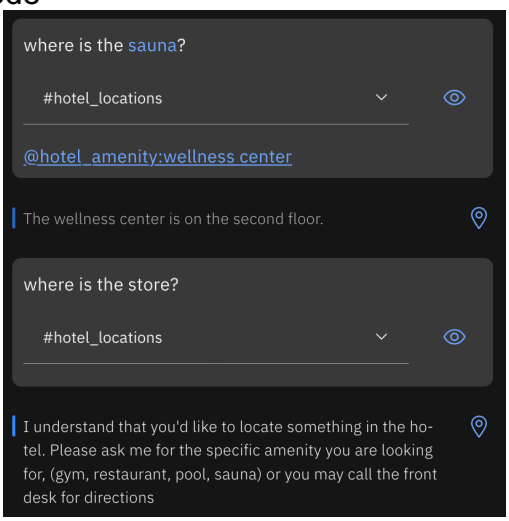
Field	Value
Node name	*Hotel Locations*
If assistant recognizes	*#hotel_locations*
Response 1	
Condition	*@hotel_amenity:pool or @hotel_amenity:(wellness center)*
Responses	*The @hotel_amenity.value is on the second floor.*
Response 2	
Condition	*@hotel_amenity:gym*
Responses	*The @hotel_amenity.value is in the front lobby.*
Response 3	
Condition	*@hotel_amenity:(hotel restaurant)*
Responses	*The @hotel_amenity.value is located on the ground floor of the hotel.*
Response 4	
Condition	*anything_else*
Responses	*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.*



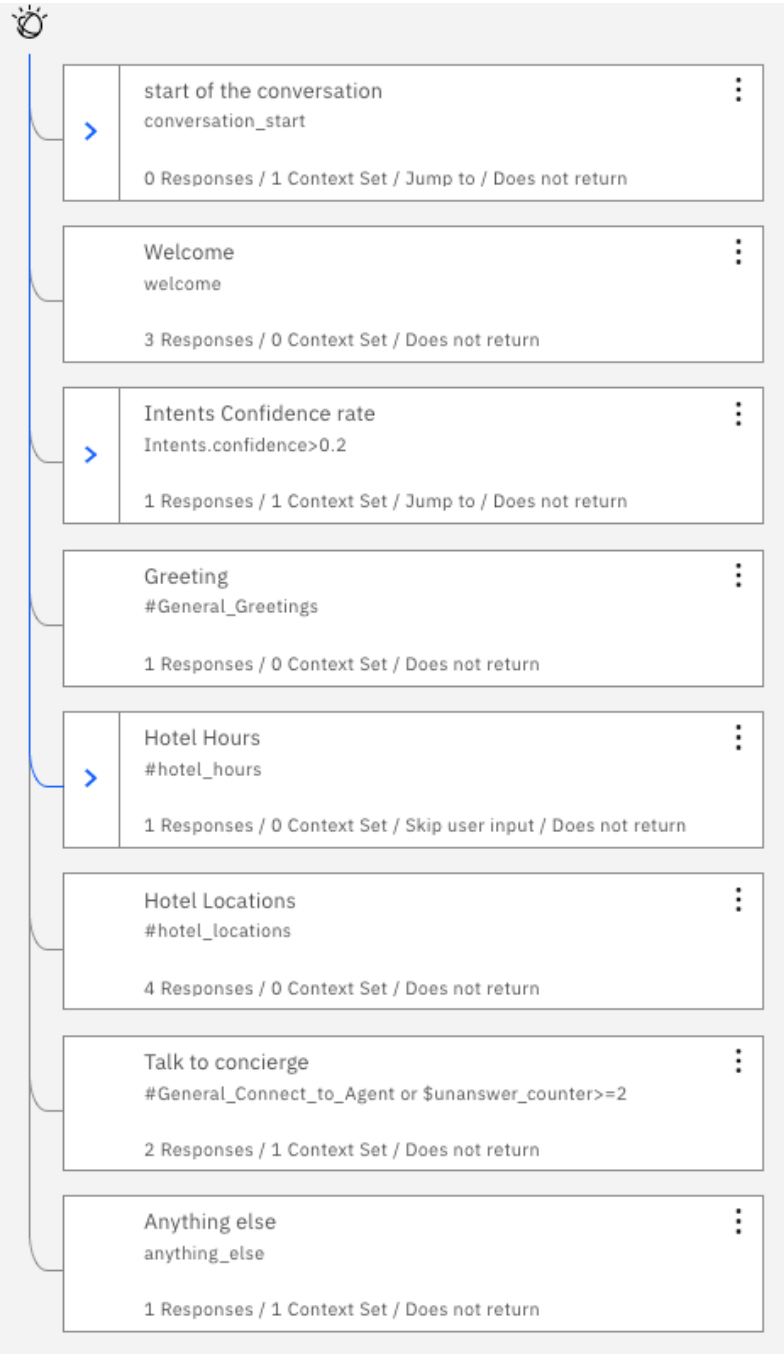
5. Close the node editor and open the **Try it out** panel
6. Type *where is the sauna?*

   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:

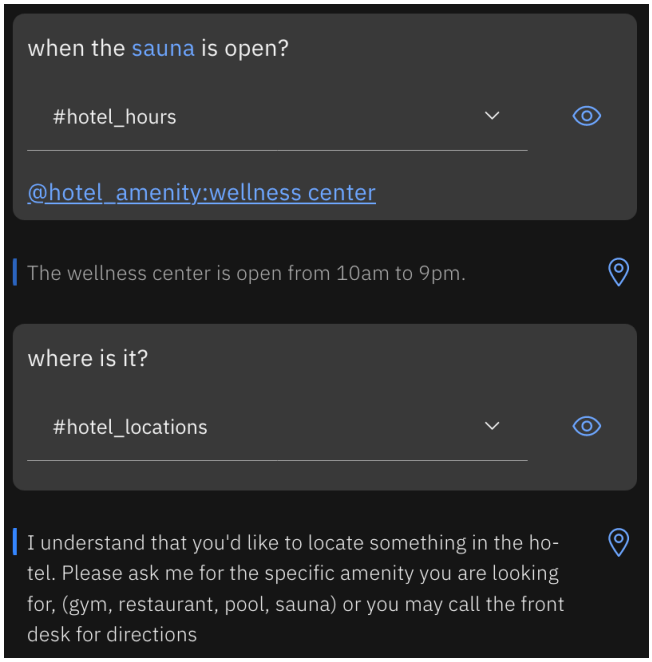


17. How to keep the hotel amenity context? (Hotel_hours node)

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.



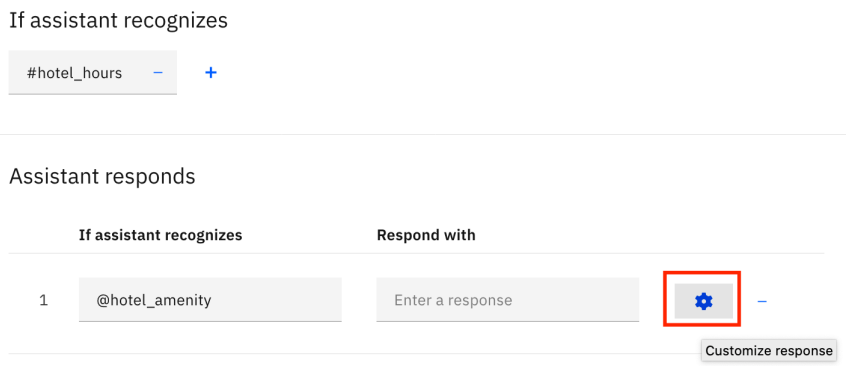
To store data, you must use context variables. Context variable is also the mechanism for passing data between WA and the application that submits the user's input to Watson.

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

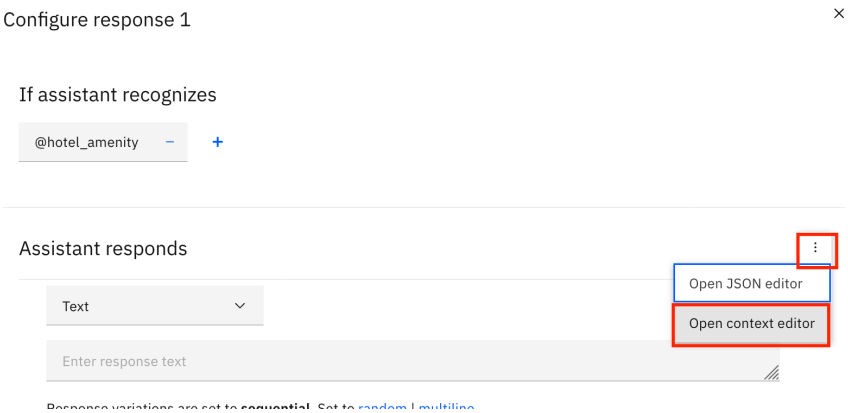
There are two methods to access it:

- 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



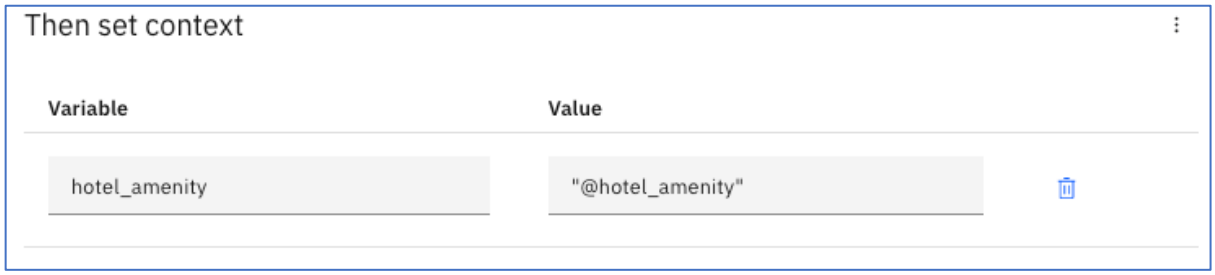
6. After a new window opens, open the **context editor** (Click on the three dots on the right of condition line then select **Open context editor**)



7. Fill in the context like this

   Variable : *hotel_amenity*

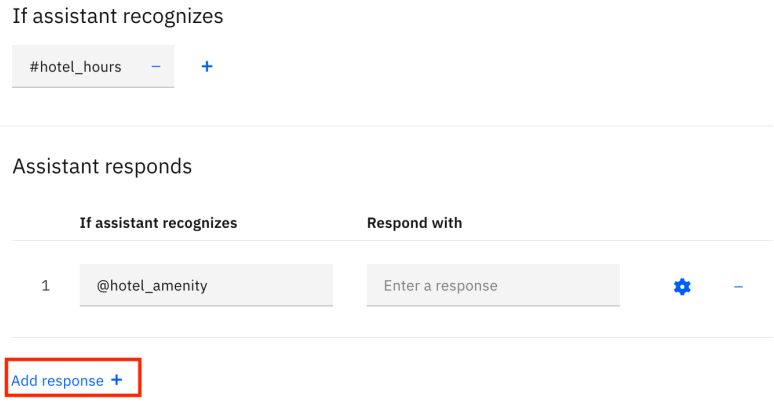
   Value : *"@hotel_amenity"*



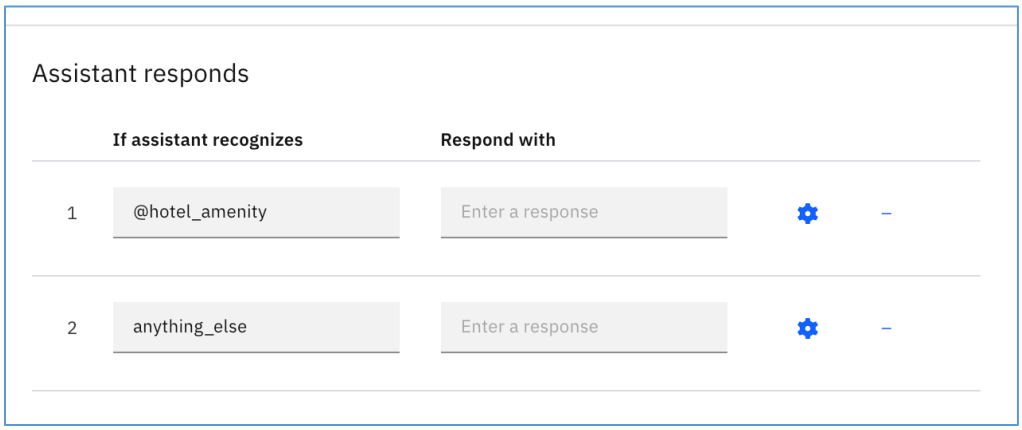
That's the way to store the entity value of *@hotel_amenity* in the context variable *$hotel_amenity when the @hotel_amenity is captured by Watson*

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:

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



10. Fill in the second row like this:
    condition : *anything_else*
    response :

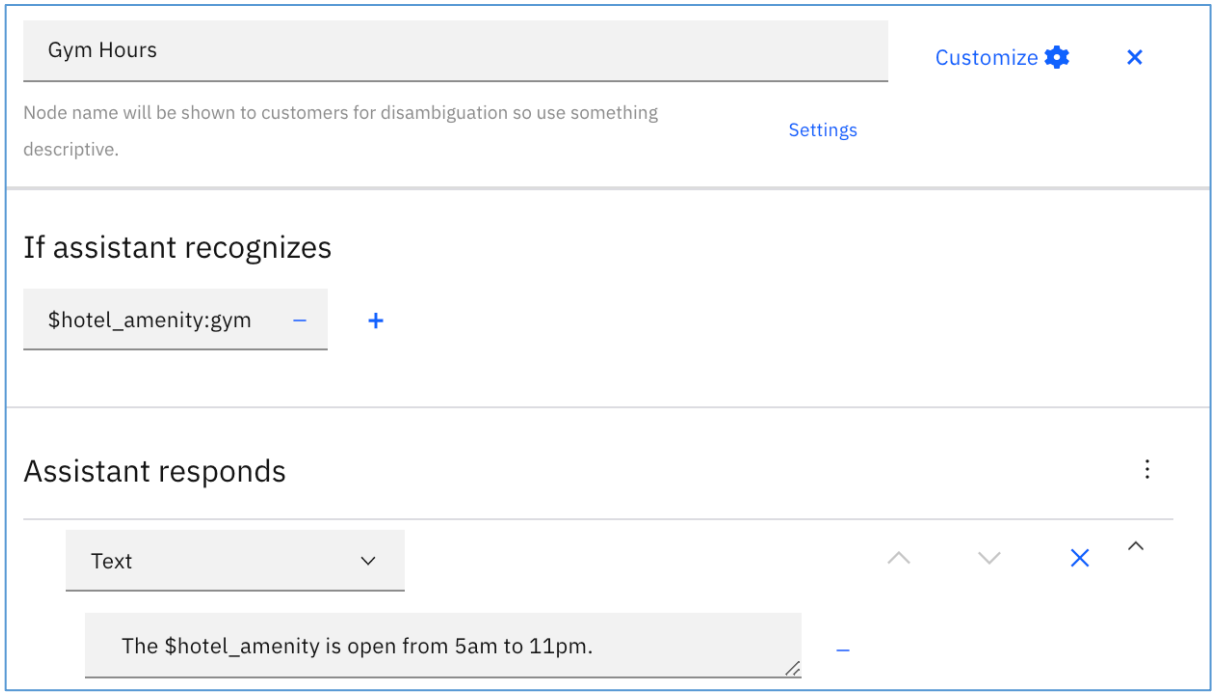


11. In each child of **Hotel Hours** node, you must replace

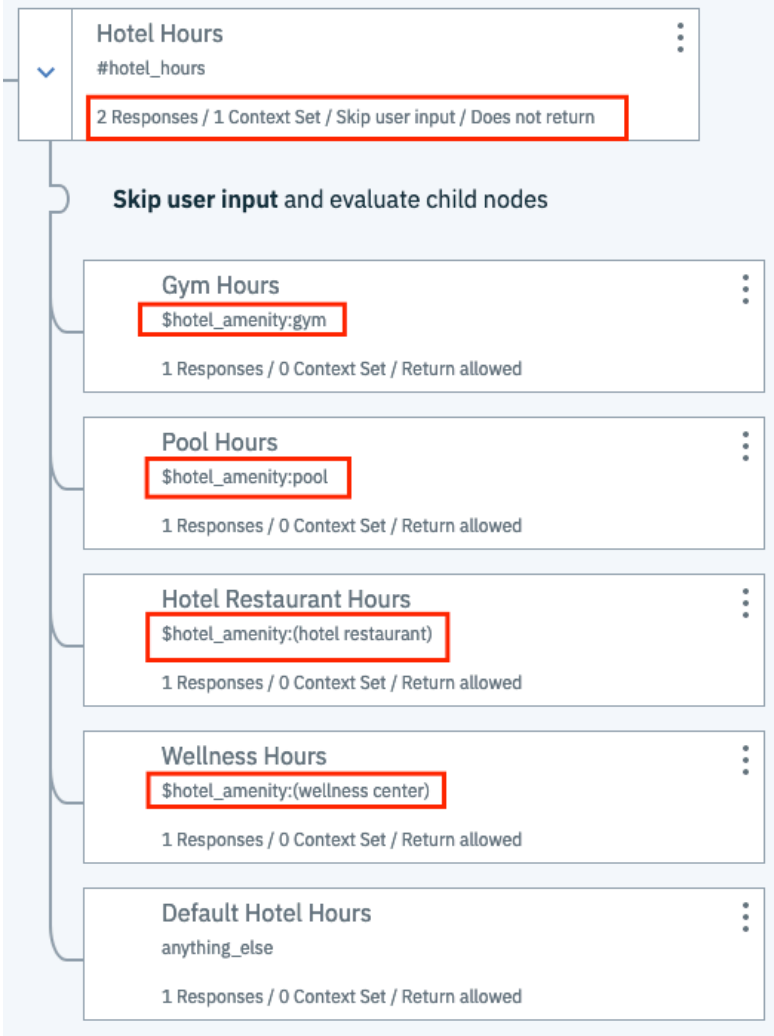
@hotel_amenity	with	*$hotel_amenity*	as condition
@hotel_amenity.value	with	*$hotel_amenity*	in the response

Note: pay attention to all @ and $ signs!

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



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



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

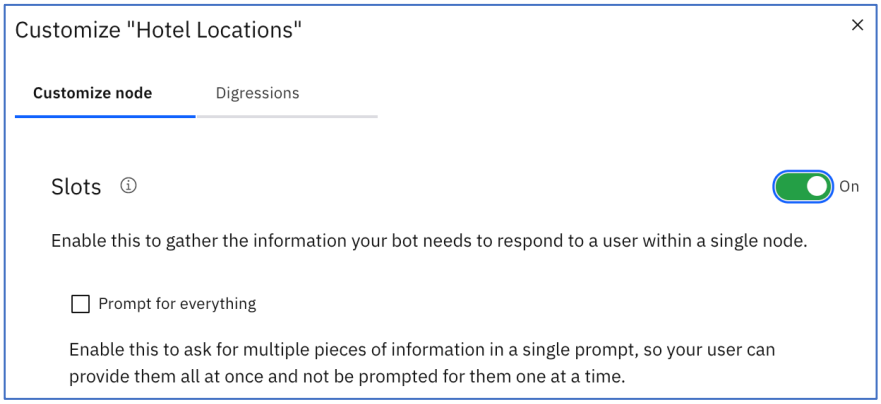
If you test these updates now, the behaviour of your bot didn't change yet

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

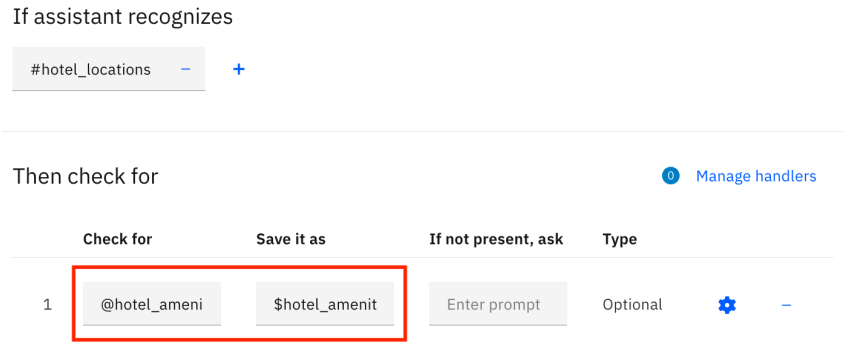
The next step is to update the node Hotel Locations to set the context variable *$hotel_amenity* and manage it correctly.

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

1. Select **Hotel Location**s 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*

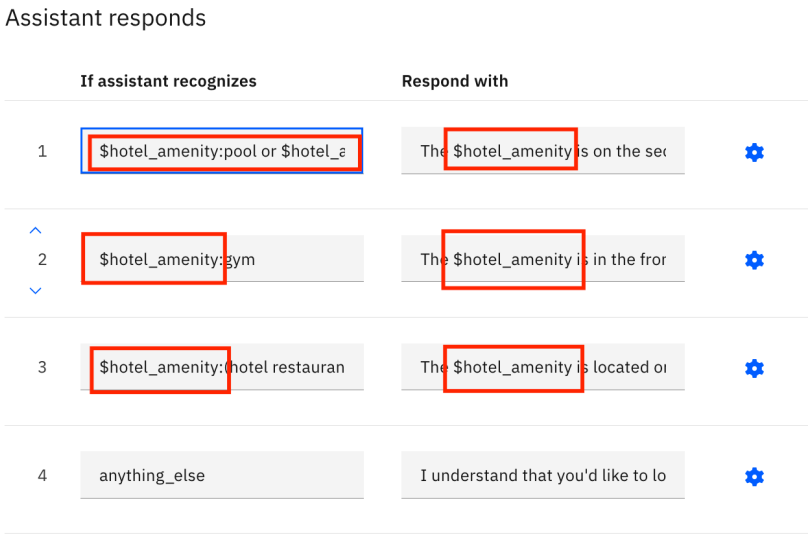


6. In each response of this node, you must replace

   @hotel_amenity with *$hotel_amenity* as condition

   @hotel_amenity.value with *$hotel_amenity* in the response

   Note: pay attention to all @ and $ signs!

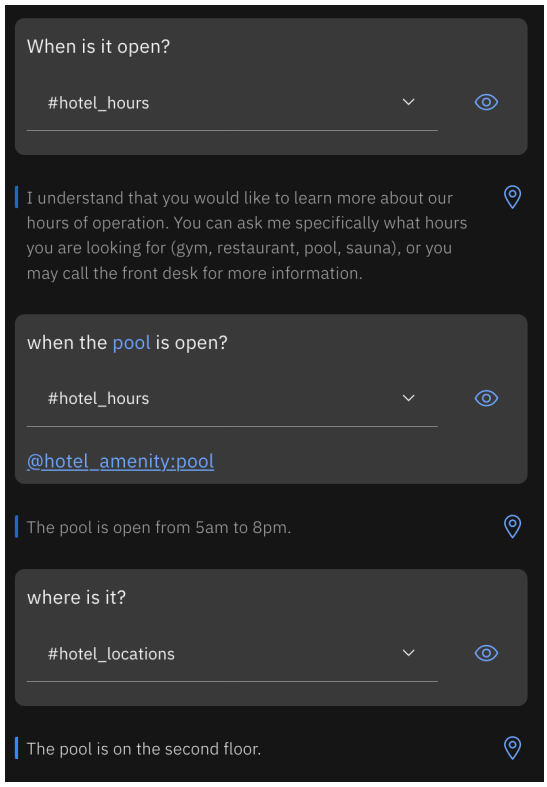


7. Let's test it, open the **Try it out** panel, enter successively:

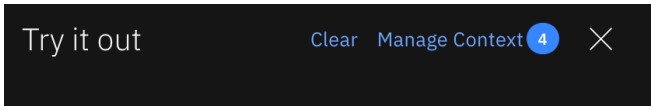
   When is it open?

   when the pool is open?

   where is it?

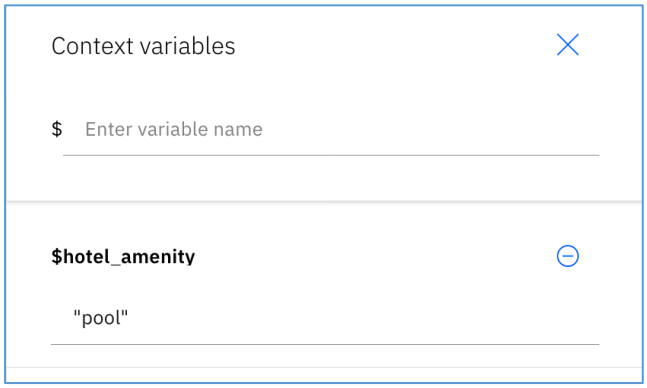


At the top of the Try it out panel, you can see that Watson manages few context variables.



8. Open the **Manage Context** panel

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



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

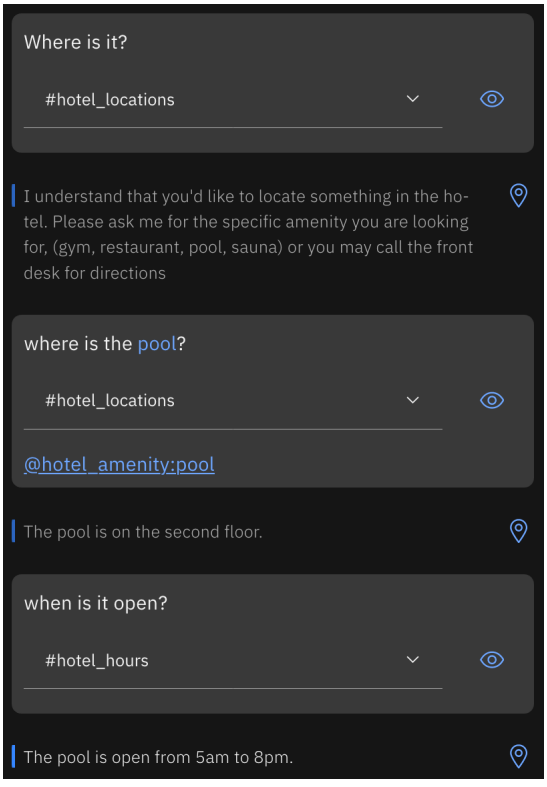
10. On the **Try it out** panel, enter successively:

    Where is it?

    where is the pool?

    when is it open?

    You will get the following behaviours:



Find a restaurant! (optional)

Let's find a restaurant for two potential intents:

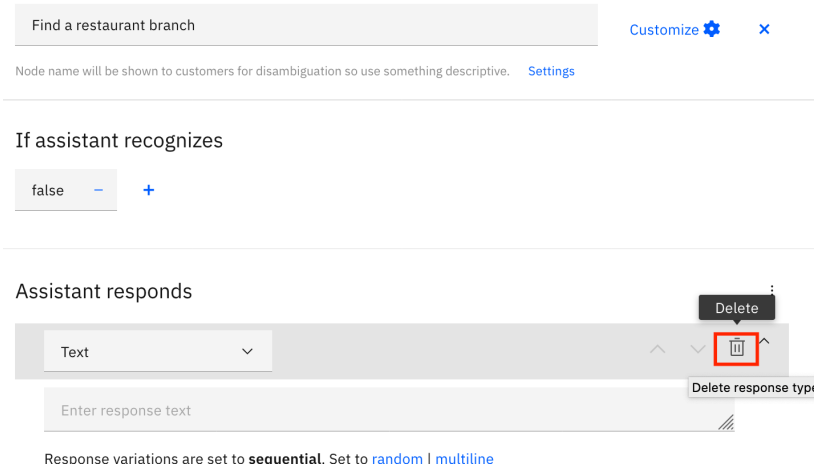
- you want to eat something
- you are looking for a specific cuisine (such as French, Italian etc.).

Watson needs 2 data: the type of restaurant and when you want to go to 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.

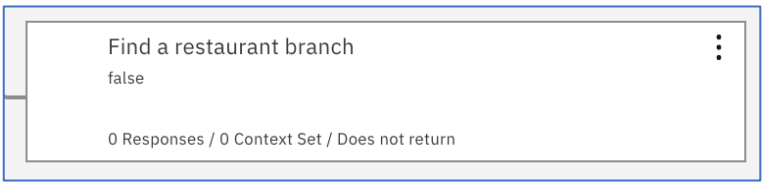
19. Find a restaurant branch

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	*Find a restaurant branch*
If assistant recognizes	*false*
Assistant responds	(Delete the response field)



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

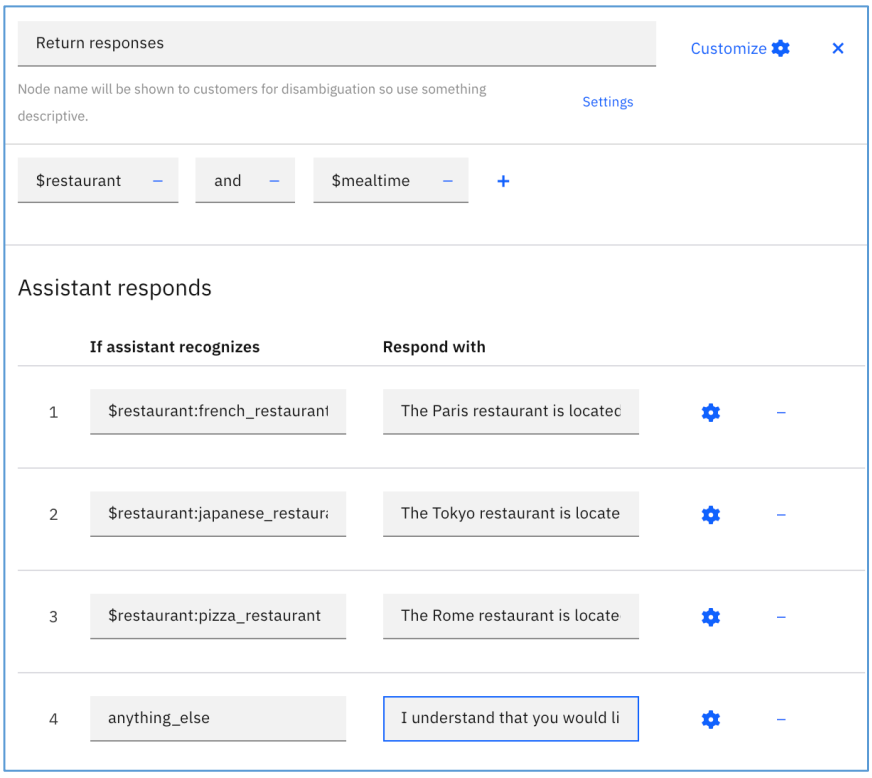
condition is false.

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.

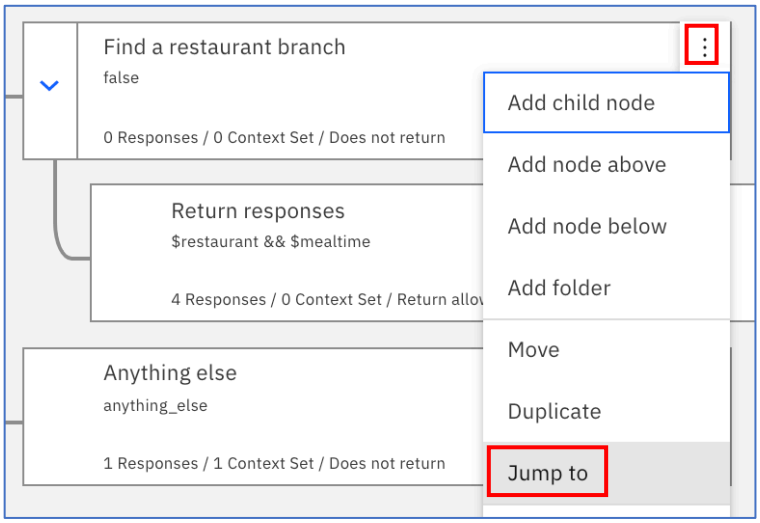
1. Add a child to **Find a restaurant branch** node and fill in the node as below (don't forget to enable **Multiple conditioned responses** option):

   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.*



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.



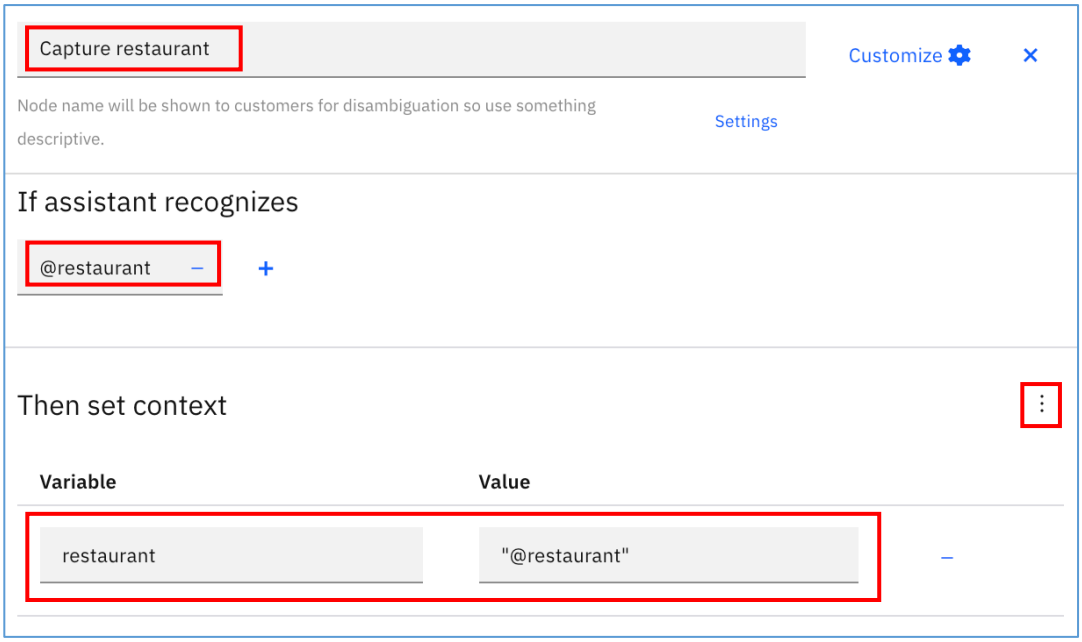
5. Select the **Return responses** node.
6. As we want to evaluate the condition, select **If assistant recognizes (condition)** Now you should have:

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*

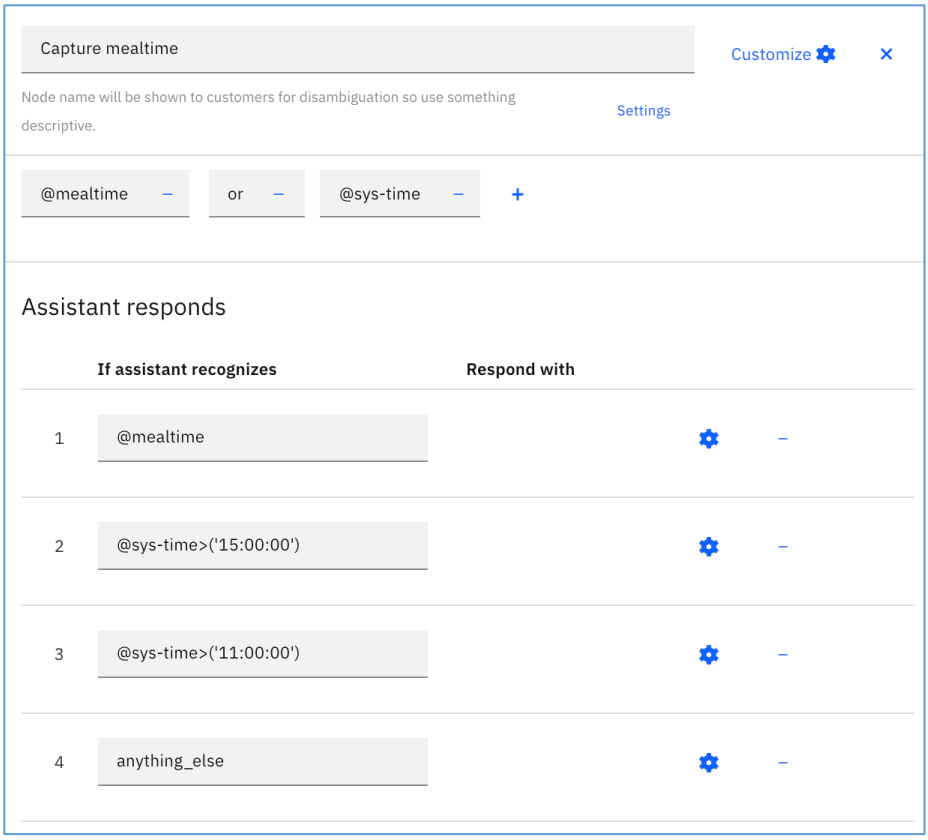
   resp3 context Value: *lunch*

   resp4 condition: *anything_else*

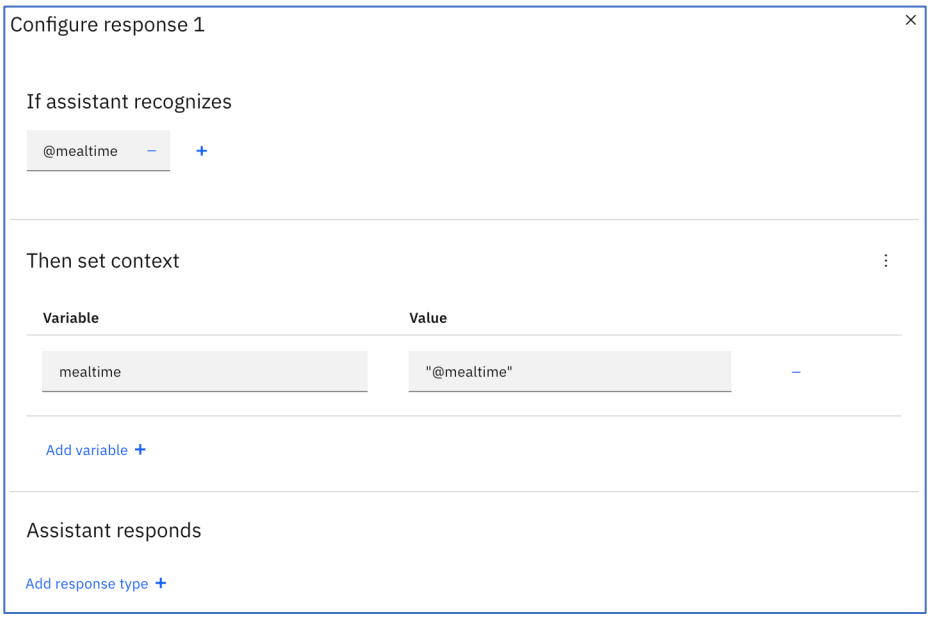
   resp4 context Variable: *mealtime*

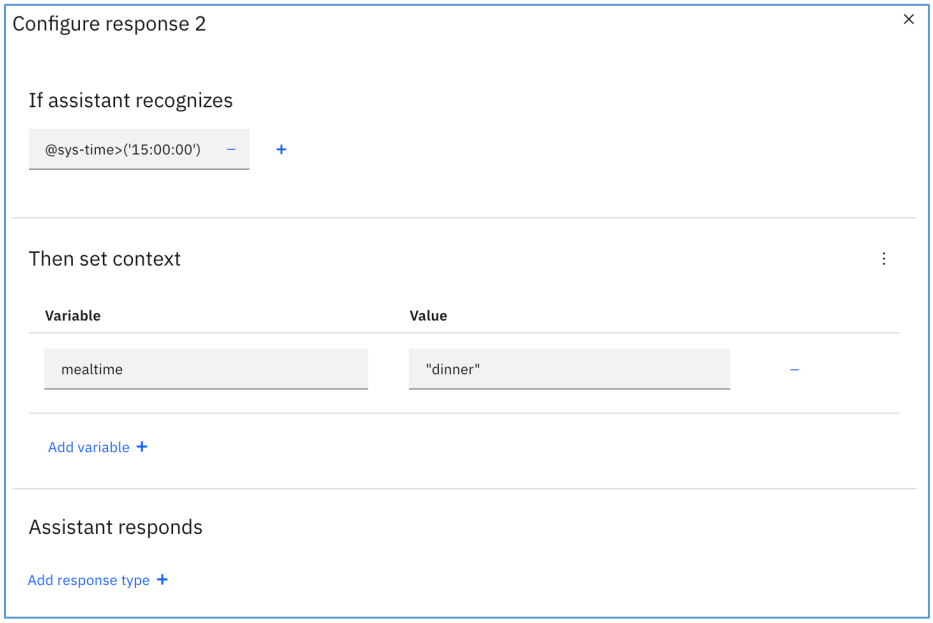
   resp4 context Value: *breakfast*

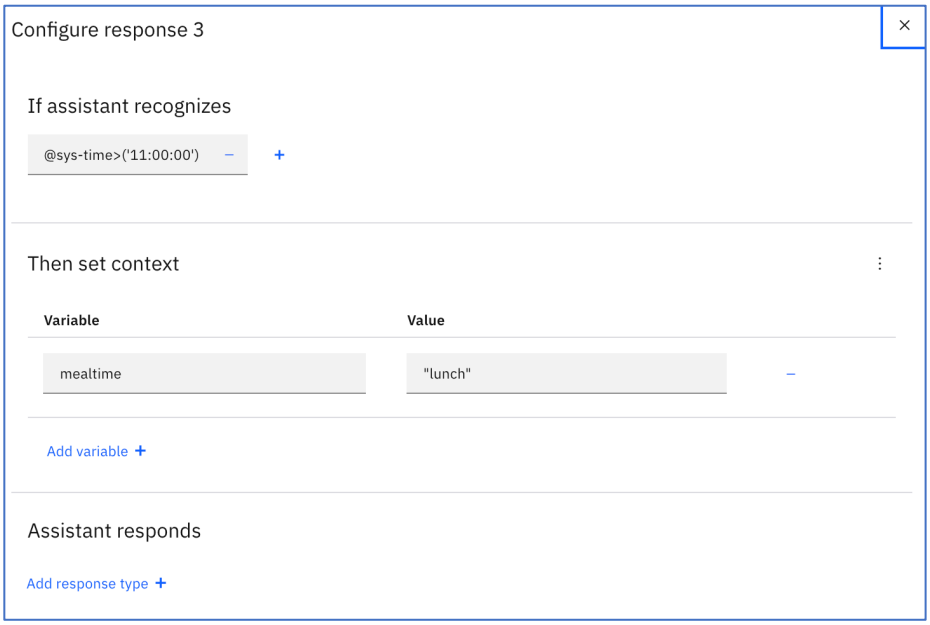
Now you should have:

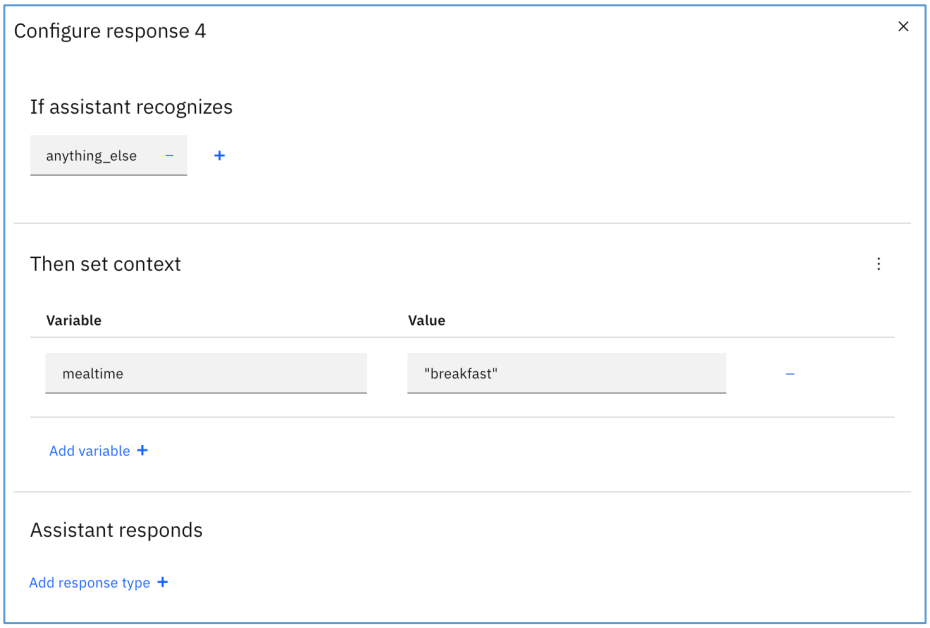


Note: When you click on each row, **configure** (the wheel icon), then **Open context editor**, you should see something like the below 4 screenshots. *Delete the response option for each row.*






Configure response 3
If assistant recognizes
@sys-time>('11:00:00')
Then set context
Variable
Value
mealtime
"lunch"
Add variable
Assistant responds
Add response type

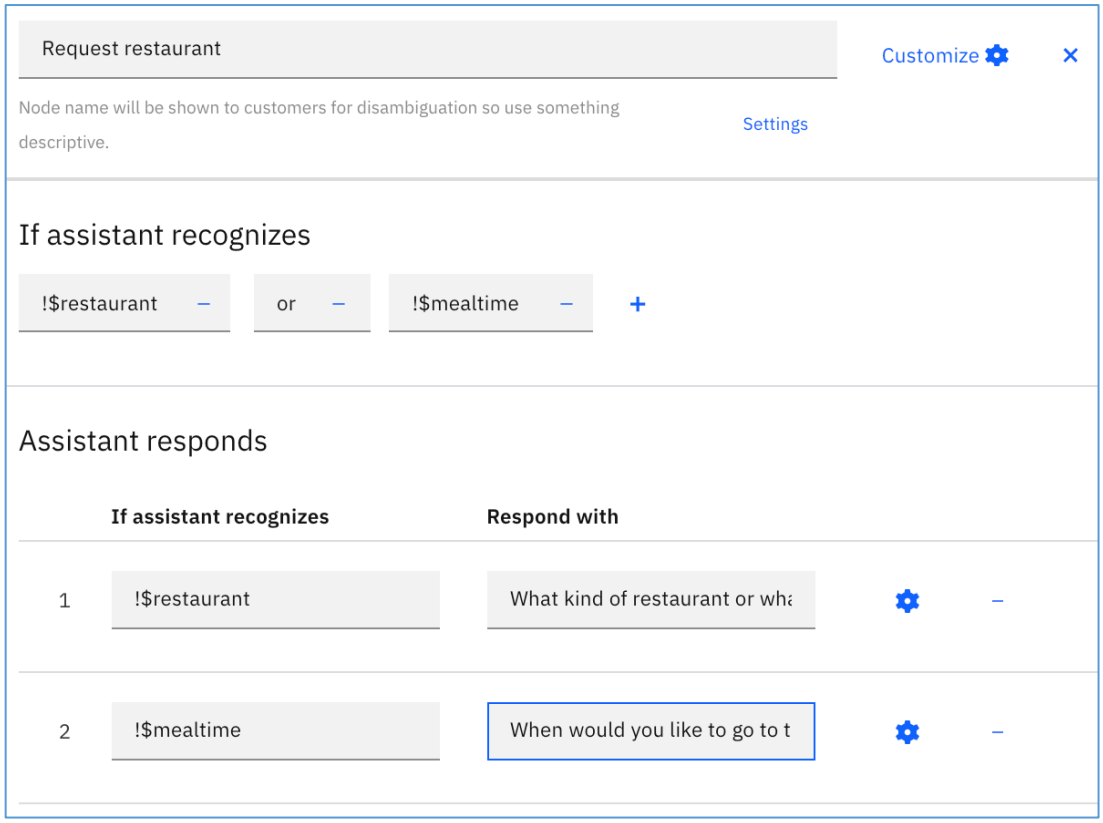

Configure response 4
If assistant recognizes
anything_else
Then set context
Variable
Value
mealtime
"breakfast"
Add variable
Assistant responds
Add response type

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)

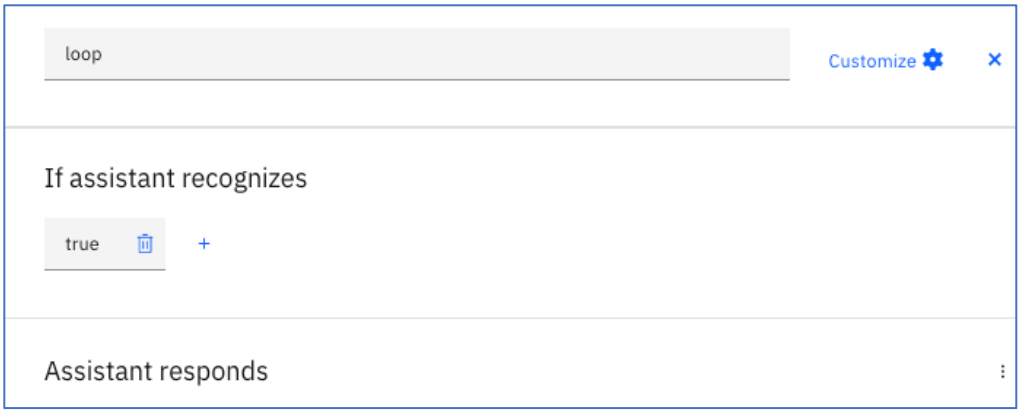
   name: *Request restaurant*
   condition: *!$restaurant or !$mealtime*
   resp1 condition: *!$restaurant*
   resp1 response: *What kind of restaurant or what food would you like?*
   Resp2 condition: *!$mealtime*
   Resp2 response: *When would you like to go to the restaurant?*



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).

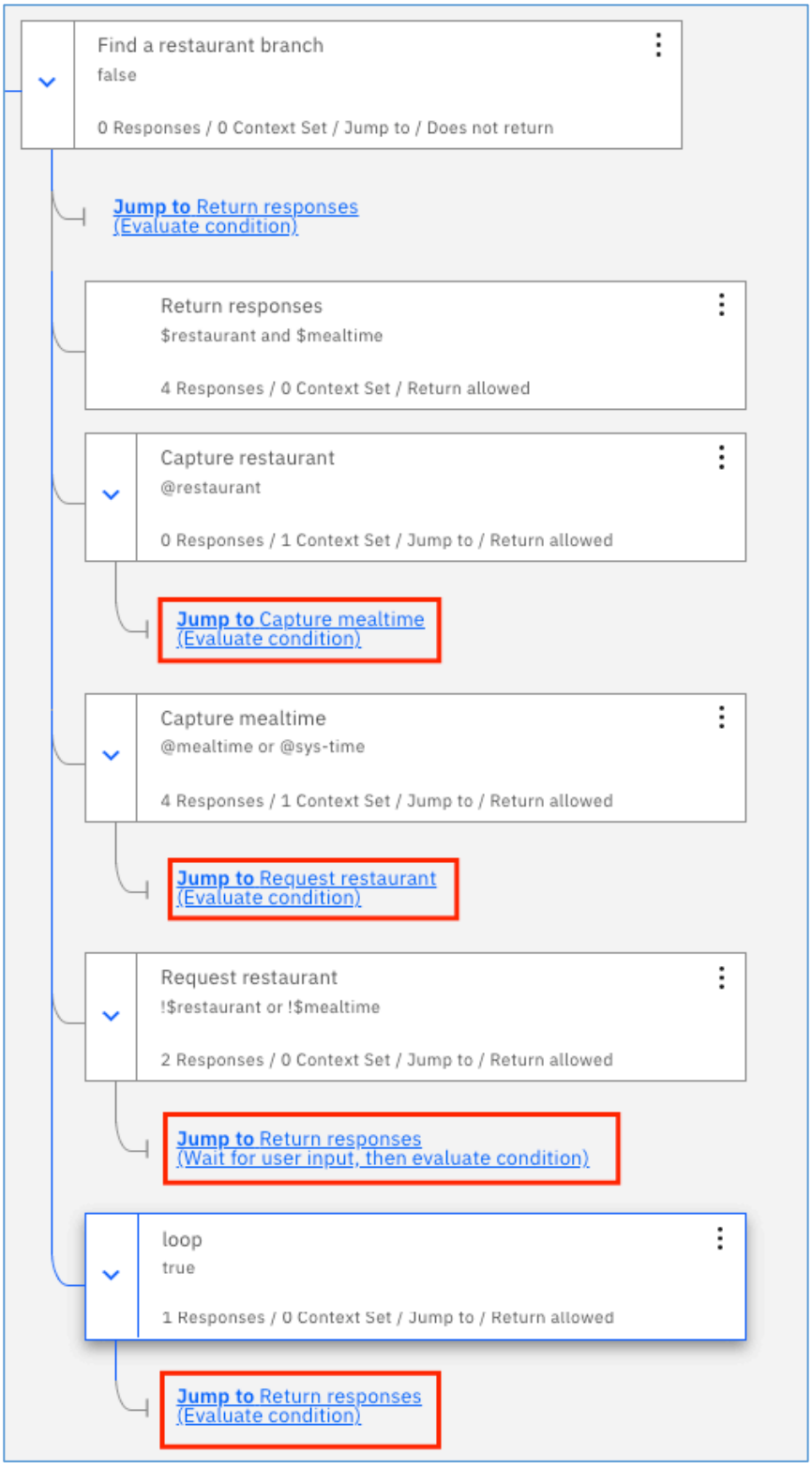
2. To do this we will add **Jump to** and **wait for user input** then evaluate condition

   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:

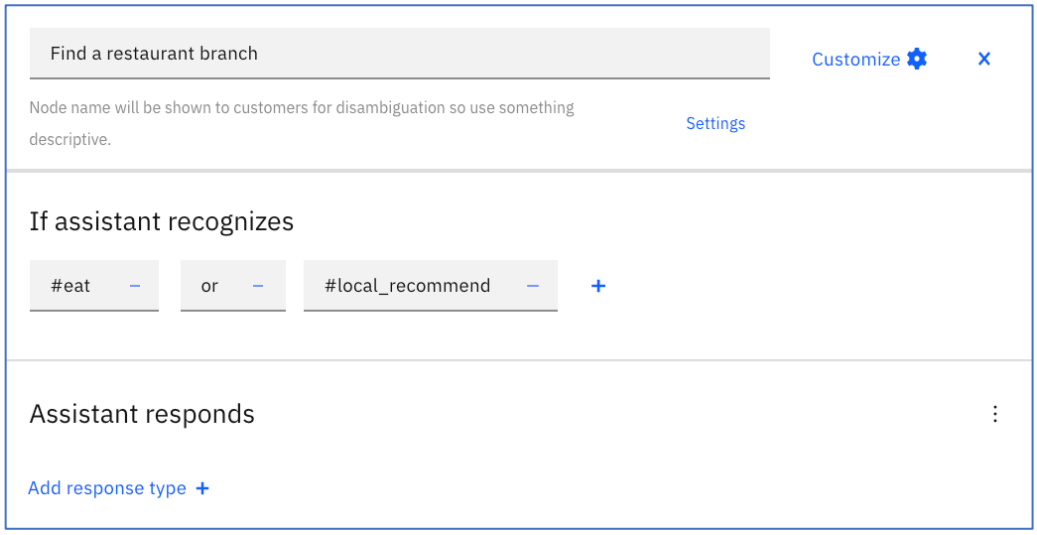


25. Test your branch

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.



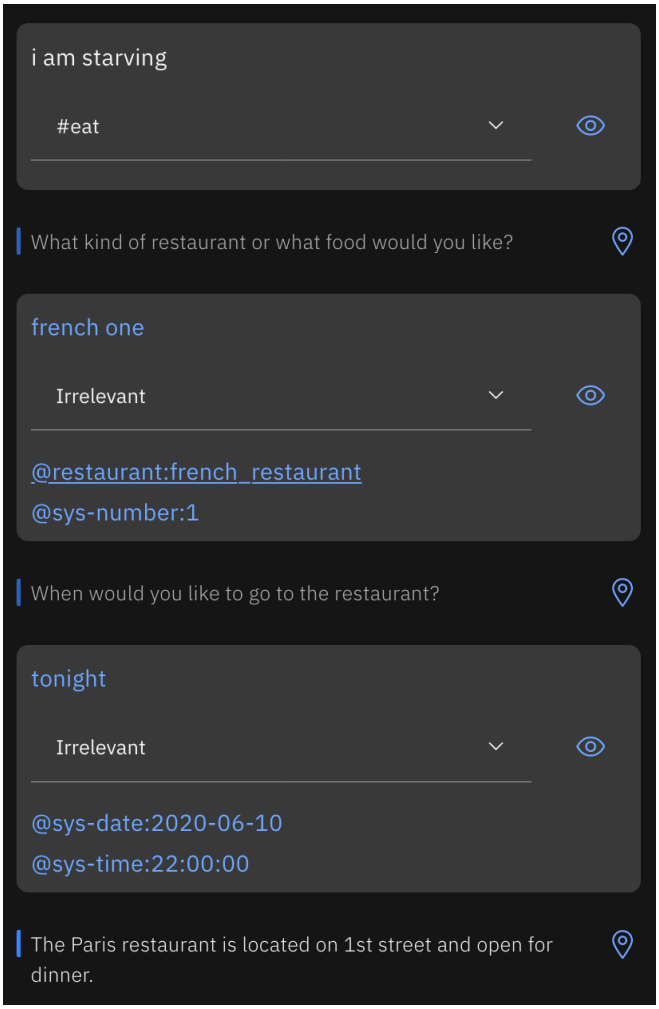
Now we should have:

4. Open **Try it out** panel
5. Type in consecutively

   I am starving

   French one

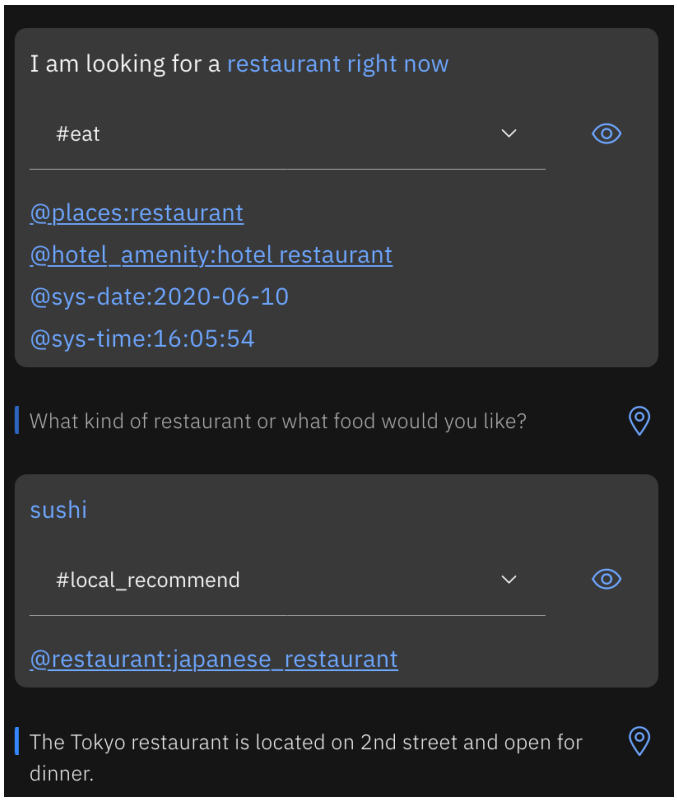
   tonight



   Watson works as expected and requests the missing information.

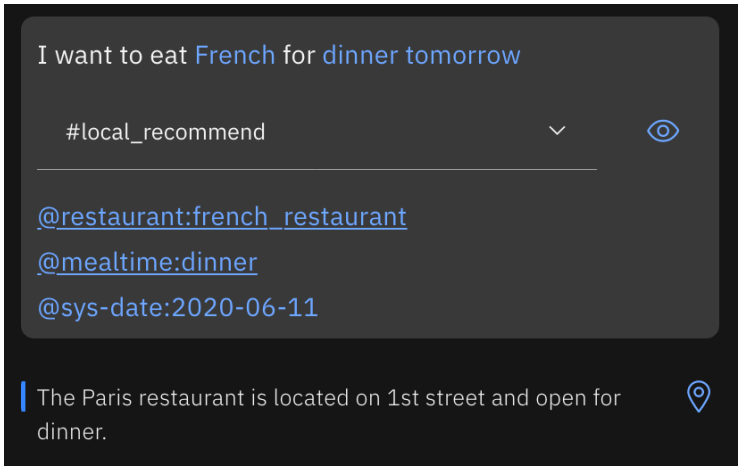
6. Click **Clear.**

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



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*



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)

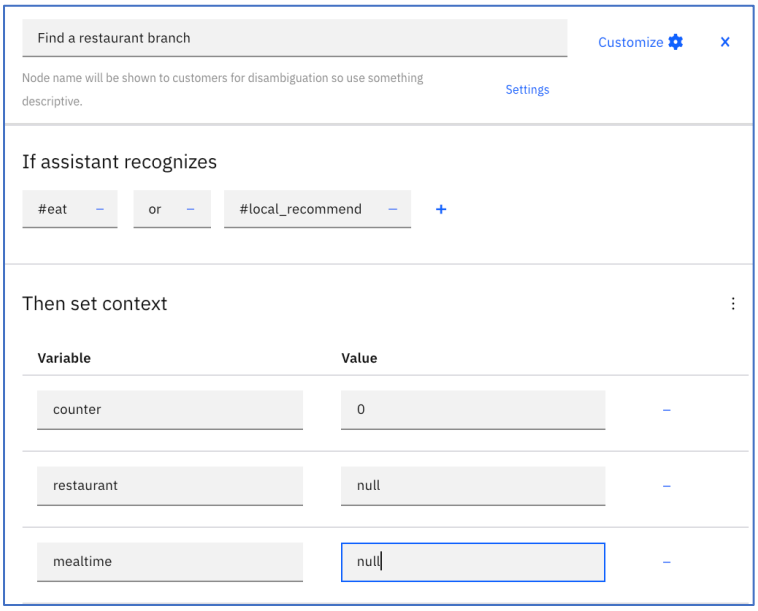
In this section, we will do some enhancements to the Find a restaurant branch.

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*

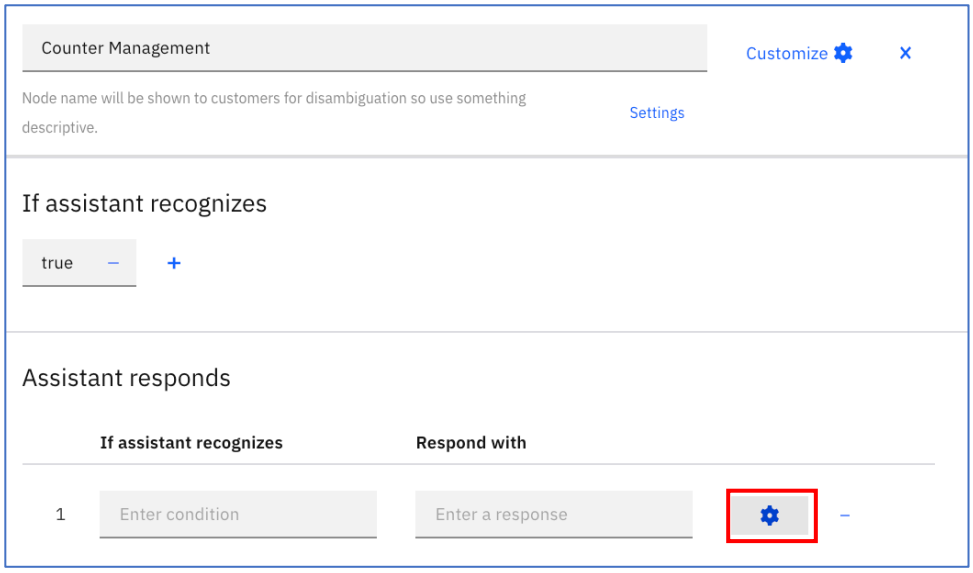


27. Avoid infinite loop

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).



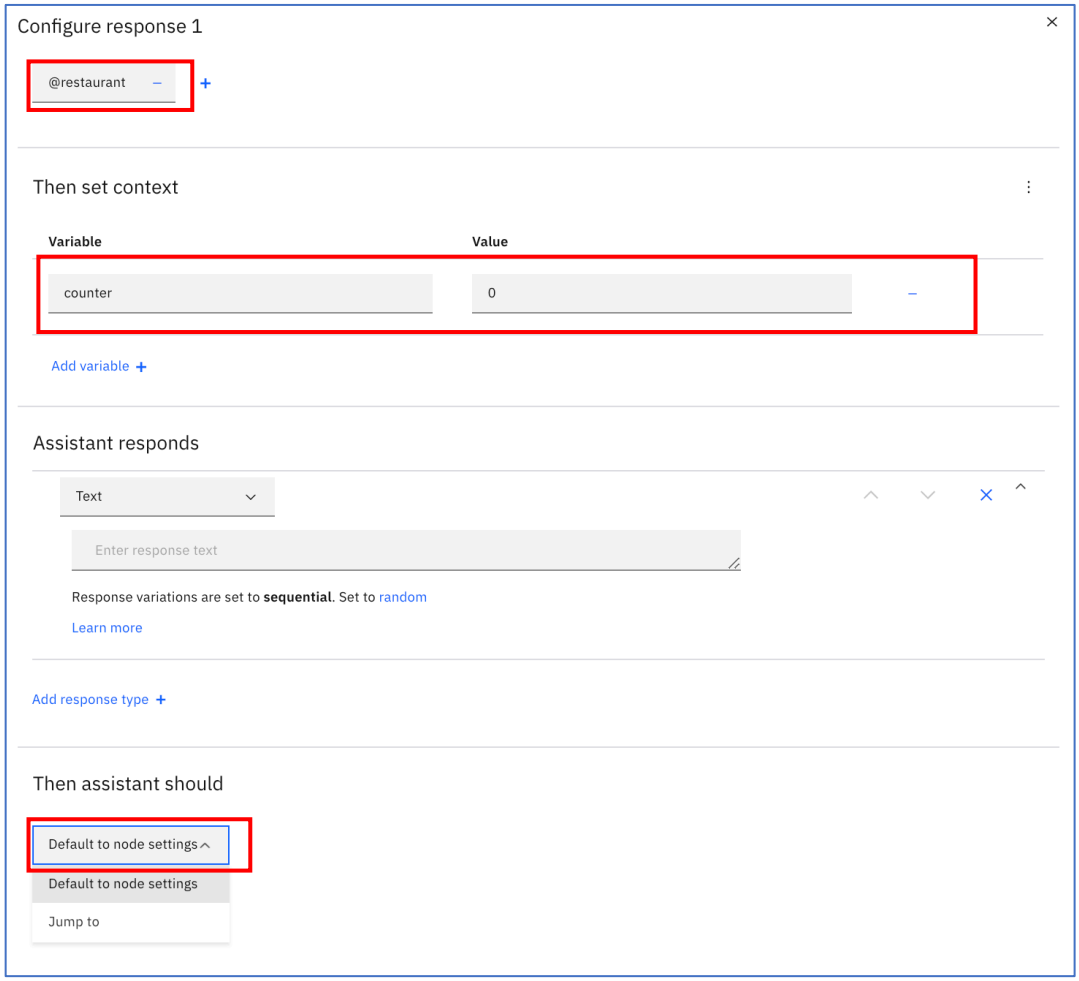
5. Click on the 3 dots menu and open the context editor.
6. Fill the node as below to reinitialize the counter when we get an expected input.

   resp1 condition: *@restaurant*

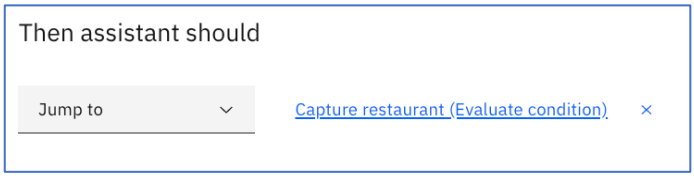
   resp1 context Variable: *counter*

   resp1 context Value: *0*

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



8. Select **Capture restaurant** node.
9. Select the **condition** option. You should get:



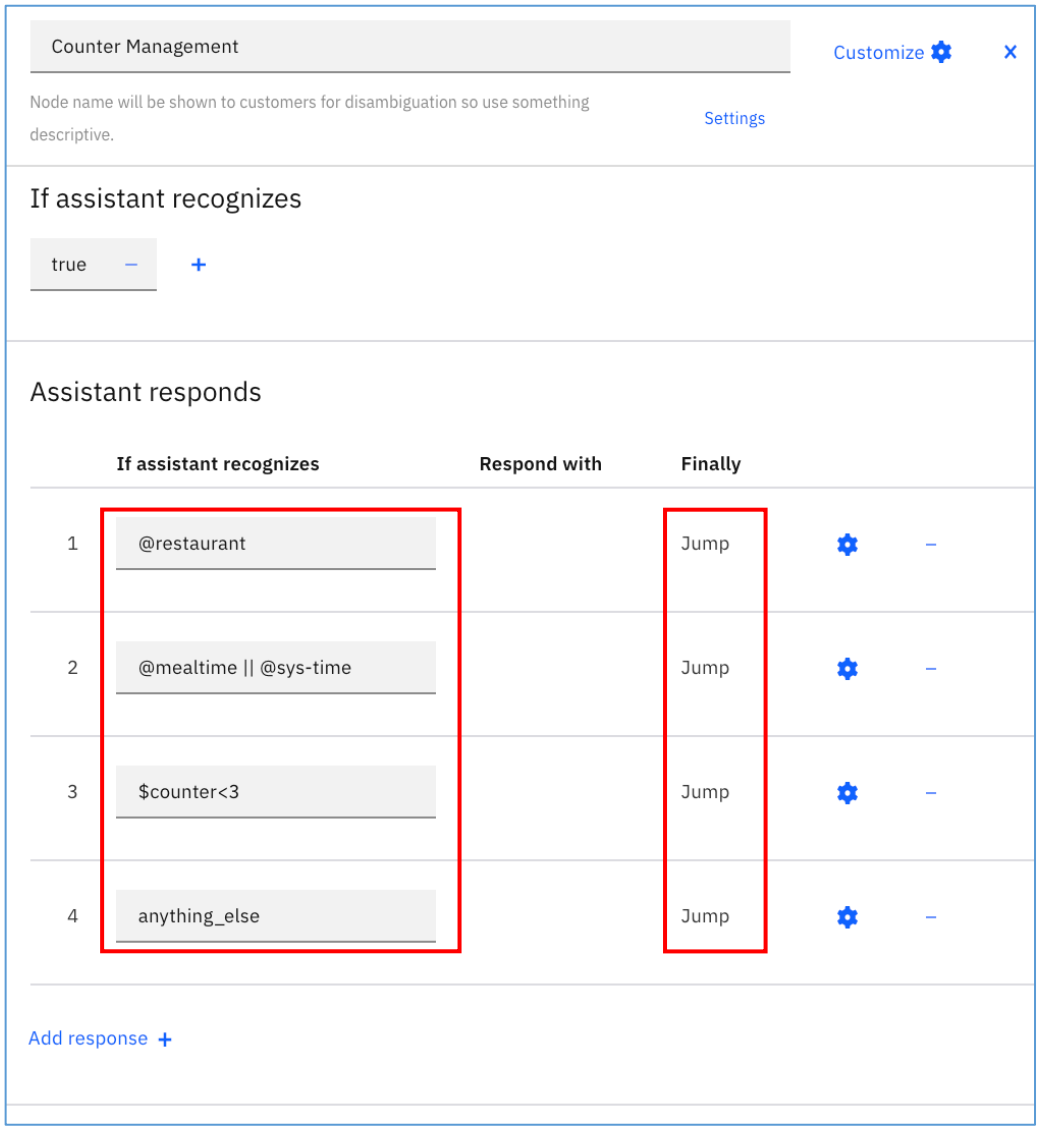
10. Click **Save**.
11. Repeat the steps 4 to 9 for the 3 other responses with the following parameters:
    - to reinitialize the counter when we get an expected input.

      resp2 condition: *@mealtime || @sys-time*
      resp2 context Variable: *counter*
      resp2 context Value: *0*
      resp2 Jump to condition node: *Capture mealtime*

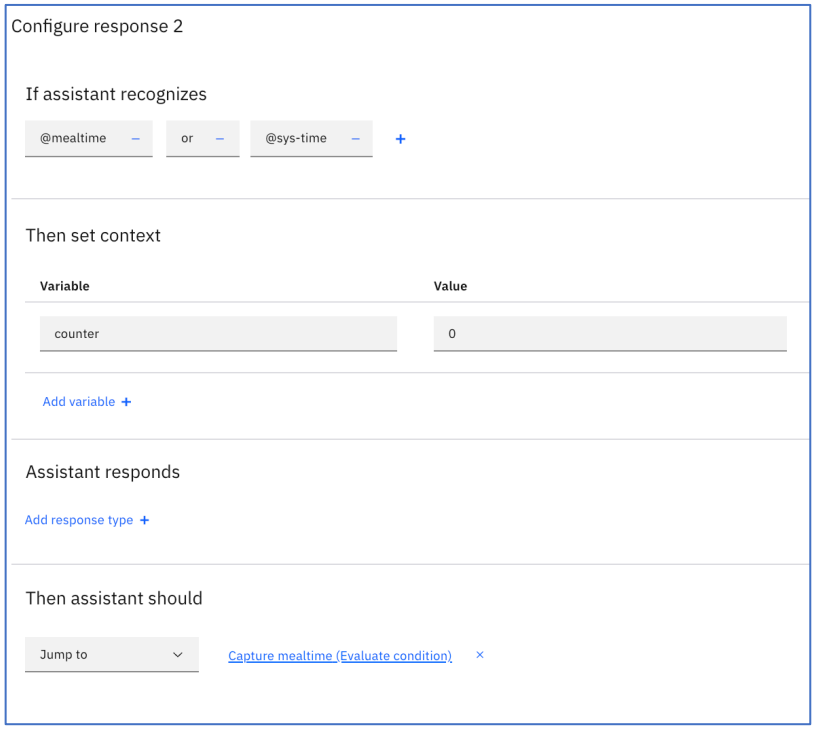
- to increment the counter when we don't get any expected input.
  resp3 condition: *$counter<3*
  resp3 context Variable: *counter*
  resp3 context Value: *"<? context.counter +1 ?>"*
  resp3 Jump to condition node: *Request restaurant*

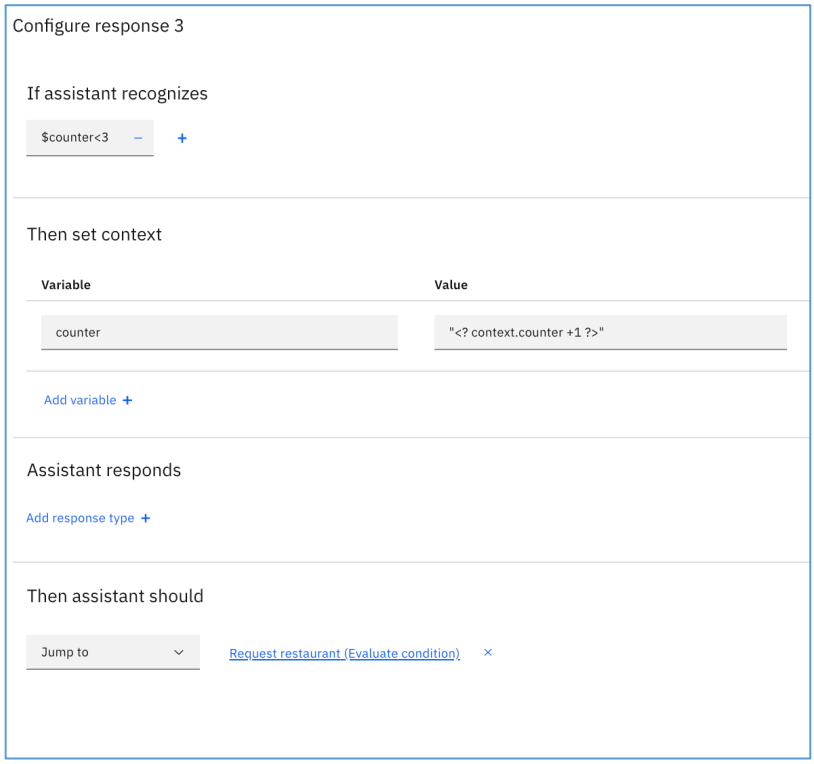
- to close the loop after too many iterations.
  resp4 condition: *anything_else*
  resp4 context Variable: *mealtime*
  resp4 context Value: *"nodef"*
  resp4 context Variable: *restaurant*
  resp4 context Value: *"nodef"*
  resp4 Jump to condition node: *Return responses*

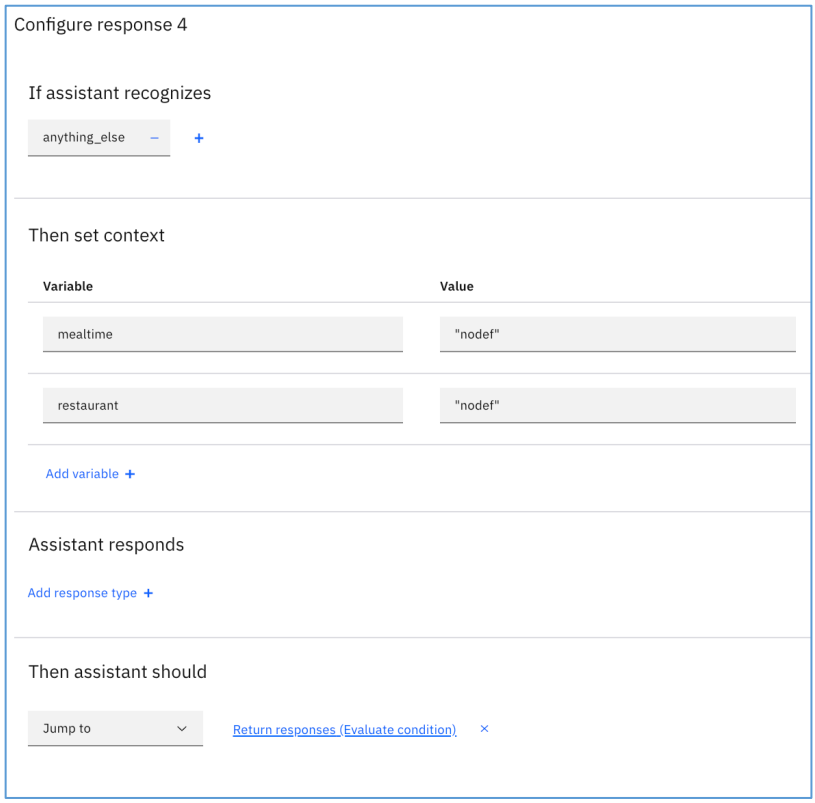
Now, you should have in your node:



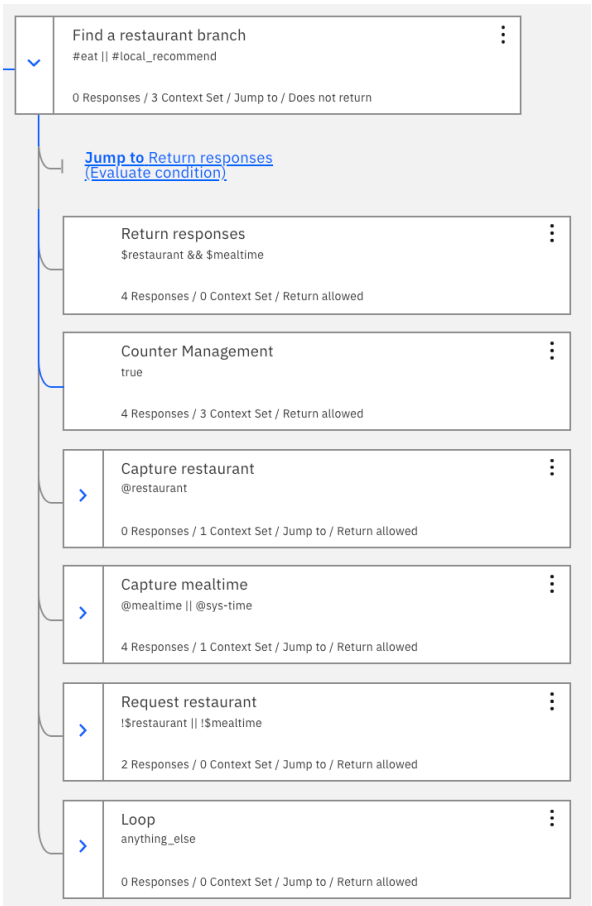
With for the 3 last responses:





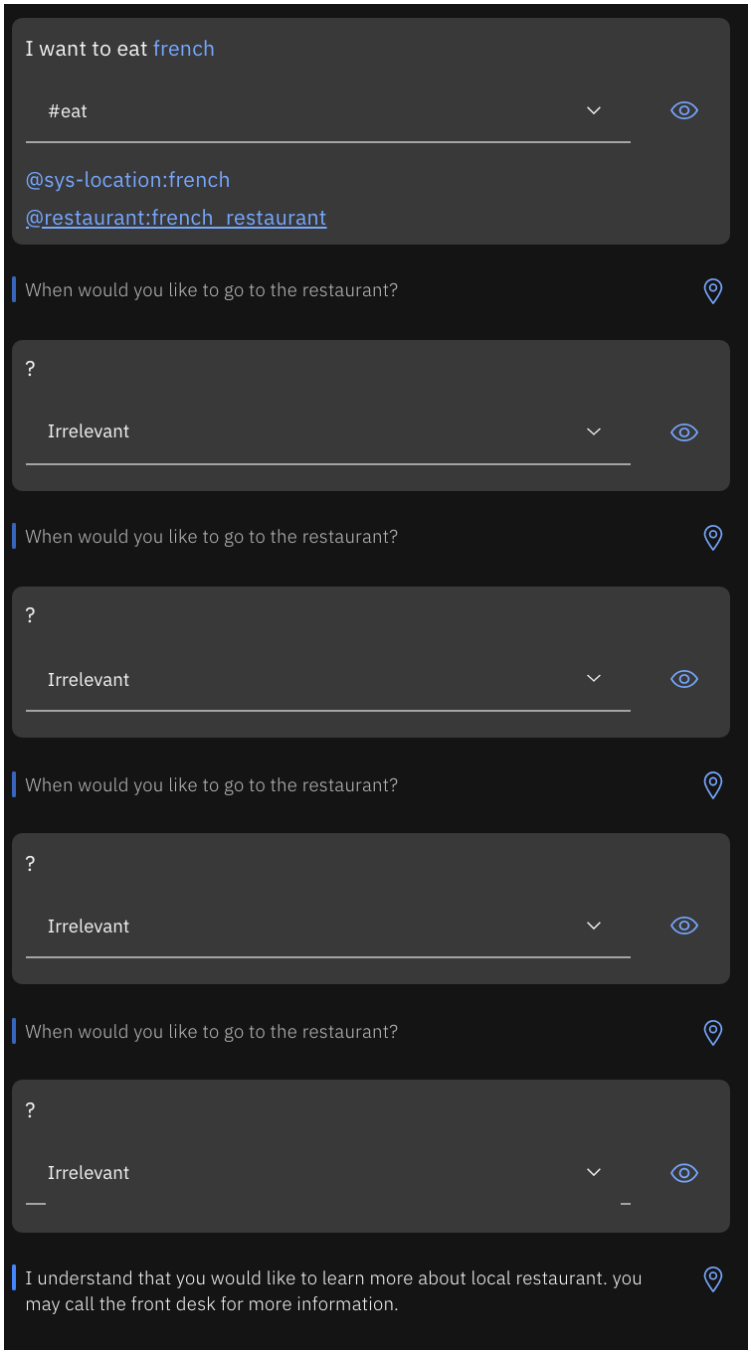


Now you should have the following branch:



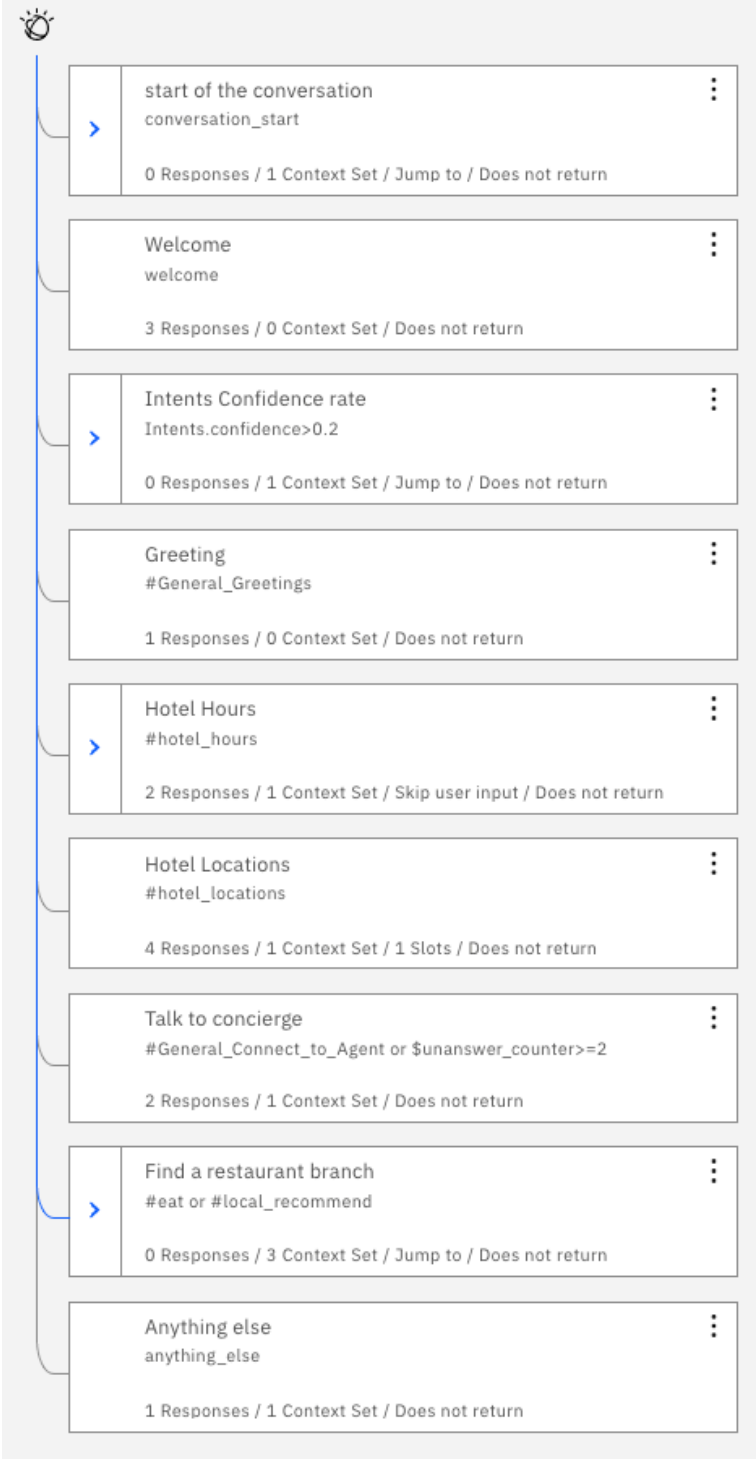
28. Test the enhancements

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.

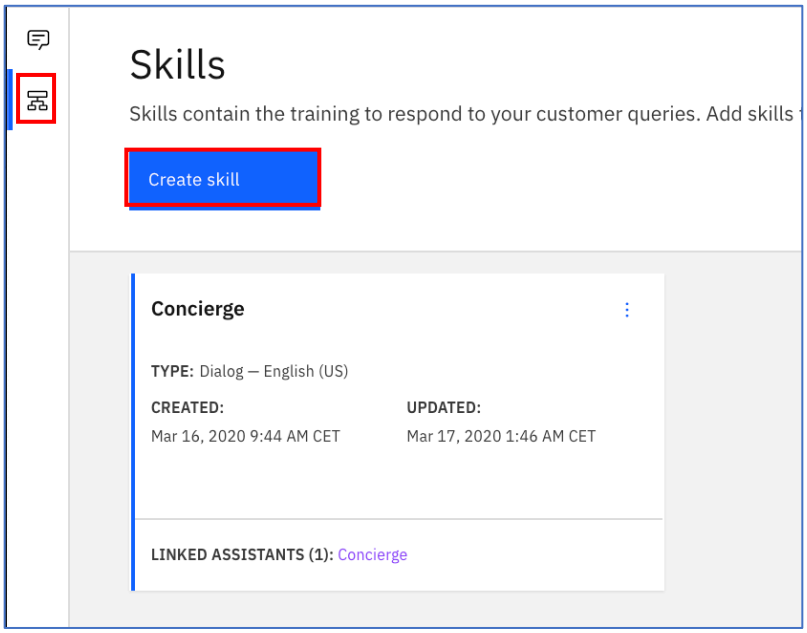
The final dialog:



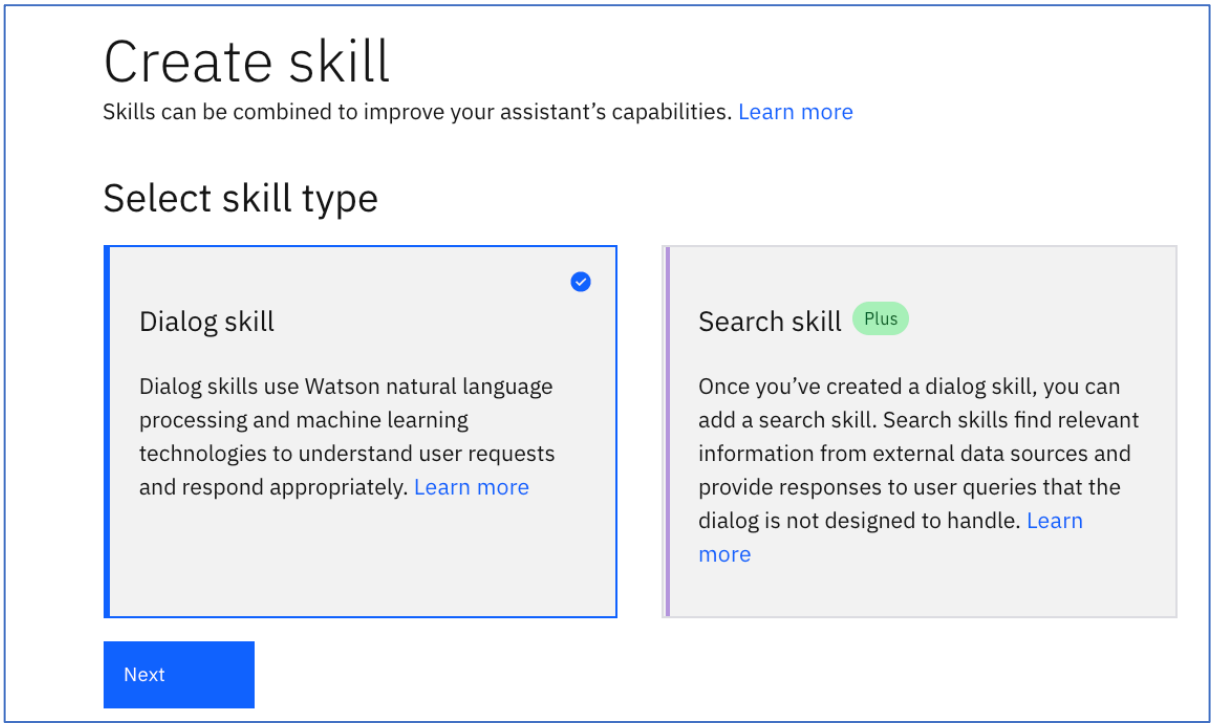
Note: if you want to compare your dialog with the one we created, feel free to import into your Assistant our Skill as it was after the end of this exercise. Our Skill is in the file *skill-My-Concierge-Skill-Lab2part1.json*.

You can import a Skill into your Assistant this way:

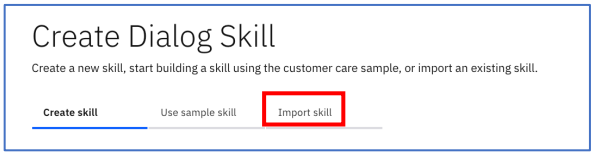
1. On Skills tab, click Create skill.



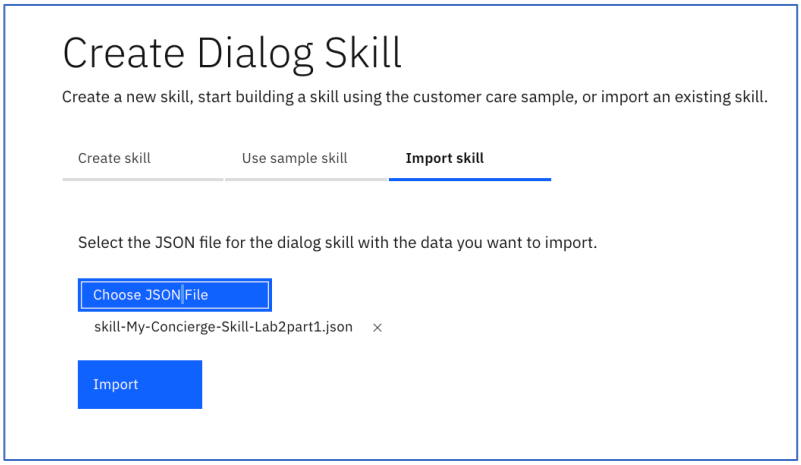
2. Select **Dialog skill** and click **Next**.



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.