

CHATBOT DEPLOYMENT WITH IBM CLOUD USING WATSON ASSISTANT

Team members:

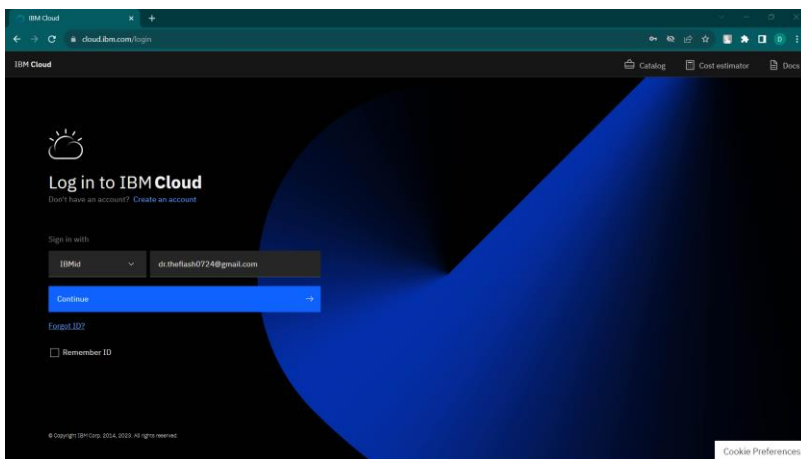
1. Harini.B - 311521106033
2. Akshaya.E -311521106006
3. Mageswari.M - 311521106057
4. Jayashree Priya.L - 311521106043

Project Title: Chatbot Deployment with IBM Cloud using Watson Assistant – **Travel-Based Chatbot**

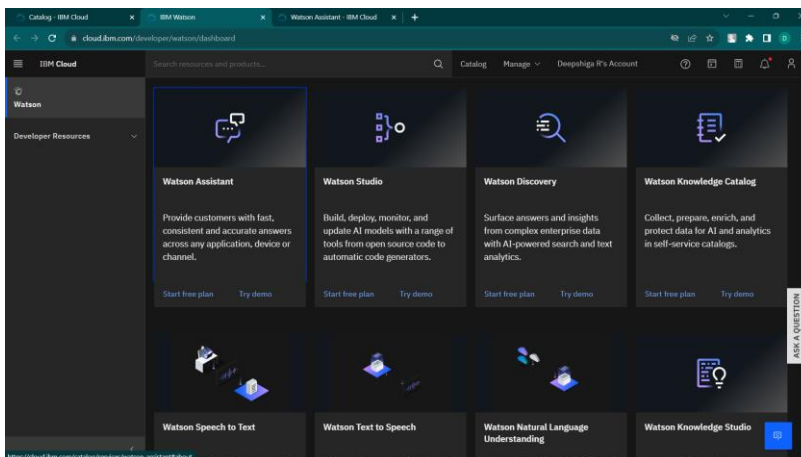
Phase 3: Development Part I

Development: Part 1

Task 1: Create the Assistant service



1. The first task is to create an instance of Watson Assistant on IBM Cloud.
2. Make sure that you are logged in to your IBM Cloud account. Click Catalog and then click Services > Watson > Assistant.



IBM Cloud Watson Services

3. For the service name, type **TOURISM**. Click Create. Assistant Service will be displayed.

Start Creating Your Chatbot

Click **Get Started** in the **Build a Chatbot** window.

Build a chatbot

Create a chatbot to interact with your customers.



[Get Started](#)

4. Click the Launch tool to open the Watson Assistant workspace.

Task 2: Create a workspace

1. In the Workspaces section, click Create. A workspace will be created.

^ Services (1)				
 assistant KVQ7Z	Default	Frankfurt	Watson Assistant	● Active

Click on your instance and then **Launch Watson Assistant**.

Start by launching the tool

[Launch Watson Assistant](#)

[Getting started tutorial](#)

[API reference](#)

Click **My first assistant**, then the **Dialog** window in the **Skill** section.

My first assistant <small>Built for you to explore and learn.</small>	Skills (1) <small>My first skill</small>	Integrations (0)
---	--	-------------------------

2. Type a name for the workspace., the workspace name is **TOURISM AND TRAVELS**.

Task 3: Create Intents

Add intents. An intent is a group of examples of things that a user might say to communicate a specific goal or idea. To identify intents, start with something that a user might want and then list the ways that the user might describe it. For each intent, think of the various ways that a user might express his or her desire—those are the examples. Examples can be developed by using a crowdsourcing approach.

For example, in a discussion with the user, we might gather this set of standard questions that support received from users:

- **Which region are you from?**
- **Phone number**
- **Gmail id**
- **Name if needed.**

Each of those questions is documented as a frequently asked question in the support team's document repository. Some solutions persist in a relational database in the form of application > problem > solution.

Based on the questions, you can extract these intents:

- **Explain about this chatbot to a 5-year-old.**
- **Common locations and Common places.**
- **Why is this chatbot?**

Add those intents to the workspace: From the Build page, click Intents and click Create New.

Intents

For the intent name, type applicationAccess after the number sign (#).

Questions

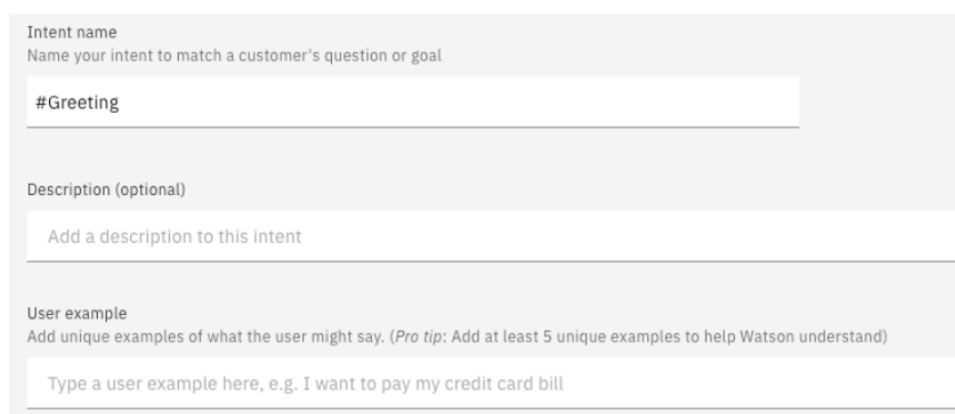
For each intent, add examples to train the conversation for intent recognition.

To get the IT support demonstration intents, click the Import link on the Intents page to import the Intents.csv file from the GitHub repository.

Task 4: Test the intent

Next, test your conversations.

1. As soon as you create an intent, you can test it by clicking Ask Watson icon in the top, right-hand side of the conversation editor.



The screenshot shows the 'Create New Intent' form in the IBM Watson Assistant interface. It has a light gray background with white input fields. The form is divided into three sections: 'Intent name', 'Description (optional)', and 'User example'. Each section has a label and a placeholder text. The 'Intent name' section has a text input field with '#Greeting' entered. The 'Description (optional)' section has a text input field with 'Add a description to this intent'. The 'User example' section has a text input field with 'Type a user example here, e.g. I want to pay my credit card bill'.

Intent name
Name your intent to match a customer's question or goal

#Greeting

Description (optional)
Add a description to this intent

User example
Add unique examples of what the user might say. (Pro tip: Add at least 5 unique examples to help Watson understand)

Type a user example here, e.g. I want to pay my credit card bill

Ask Watson

2. Enter one of the examples. You should get the #greetings intent identified by Watson. Enter other greetings to test the #greetings intent.
3. Test a few of the other intents that you imported from the .csv file.

Topic: Start building the chatbot using IBM Cloud Watson Assistant

Tourism

Introduction:

A Travel-based chatbot using IBM Cloud and Watson Assistant is a computer program that can simulate conversation with humans, specifically about Tourism. It can be used to provide information about travel, such as places, information, and hotel recommendations. It can also be used to take food orders and make reservations.

To build a travel-based chatbot using IBM Cloud and Watson Assistant, you will need to:

- Create an IBM Cloud account and subscribe to the Watson Assistant service.
- Create a new Watson Assistant assistant.
- Define the intents and entities that your chatbot will need to understand. Intents are the purposes or goals that users express in their input, such as finding a recipe or ordering food. Entities are specific objects or data types that are relevant to your chatbot, such as food names, ingredients, and quantities.
- Create a dialog flow for your chatbot. The dialog flow defines how the chatbot will respond to different user inputs.
- Train your chatbot. Watson Assistant will use your training data to learn how to understand and respond to user inputs.
- Deploy your chatbot. Once your chatbot is trained, you can deploy it to a variety of channels, such as your website, social media, or messaging apps.

Here are some examples of how a tourism-based chatbot could be used:

- ✓ A chatbot could help tourists to plan their trip by providing them with information about different destinations, attractions, and events. It could also help them to book travel arrangements, purchase tickets, and make reservations.
- ✓ A chatbot could help tourists to get around their destination by providing them with information about public transportation, maps, and directions. It could also help them to find restaurants, shops, and other businesses.

- ✓ A chatbot could help tourists to learn about the culture and history of their destination by providing them with information about local attractions, museums, and historical sites. It could also recommend activities and events that are relevant to their interests.
- ✓ Overall, a tourism-based chatbot using IBM Cloud and Watson Assistant can be a valuable tool for tourism businesses of all sizes. It can help to improve customer service, increase engagement, and generate leads and sales.

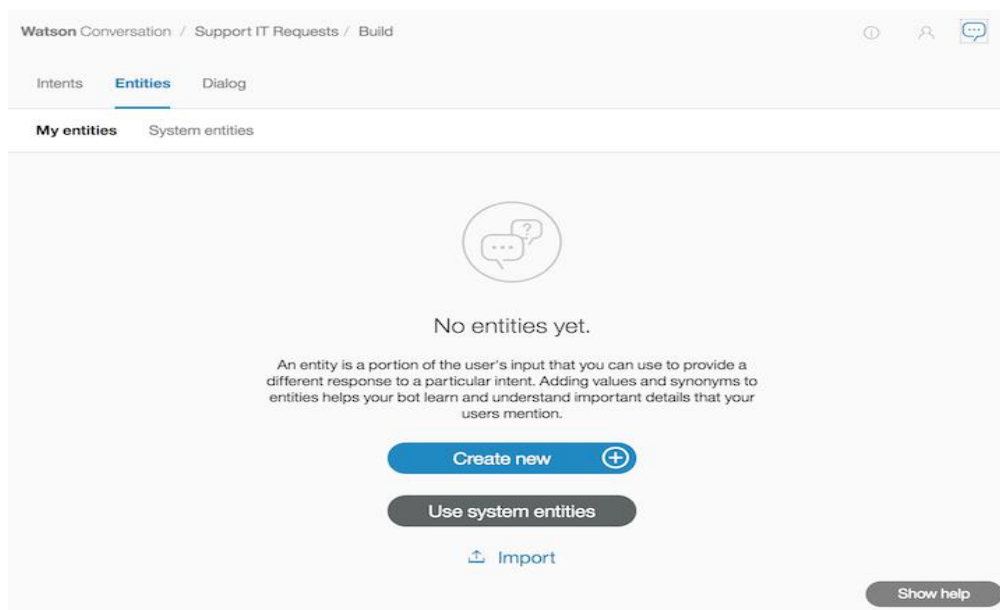
Development: Part 2

***Images are subjected to the demo we've seen and learned, since, the cloud is not activated, we have given our idea using the demo.

Task 5: Add Entities

An entity is a portion of the user's input that you can use to provide a different response to a particular intent.

1. Click Entities. On the Entities page, click Create new.



- Adding values and synonyms to entities helps your chatbot learn important details that your users might mention.
 - Each entity definition includes a set of specific entity values that can be used to trigger different responses. Each value can have multiple synonyms that define different ways that the same value can be specified in user input.
2. Create entities to represent to the application what the user wants to access.

Intents **Entities** Dialog

Entity **@application** Done

Fuzzy Matching BETA on
 Turning this on will increase the ability for Watson to recognize misspelled entity values.

Value

Inventory plus	iv	ivp	Add synonyms...	
AbC	Oder management			
Expense Report	expense report	travel expense		

- Fuzzy logic is a feature that allows Watson Assistant to accept misspelled words. You can enable this feature at the entity level.
 - As you did for intents, you can reuse entities' definitions through the export and import capabilities. Import the wcs-workspace/ITSupport-Entities.csv file.
3. If you click the Ask Watson icon immediately after you import the entities, the Watson is training message is displayed. Watson Assistant classifies the entities. You can unit-test the entities by entering I want to access application AbC. The following figure shows both the intent and entity (@application:AbC) extracted by Watson Assistant:

Try it out Clear Manage Context 1

I want to access application AbC

#ApplicationAccess

@application:AbC

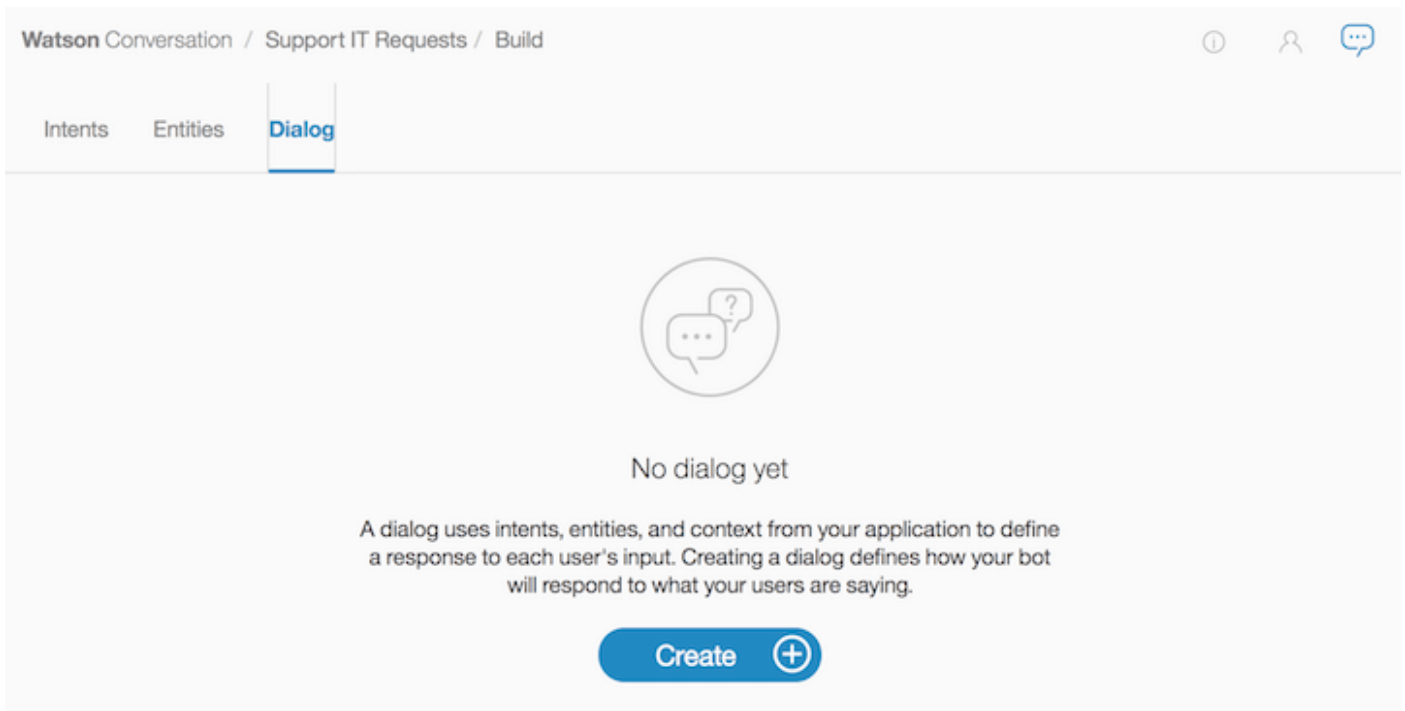
Enter something to test your bot

Use the up key for most recent

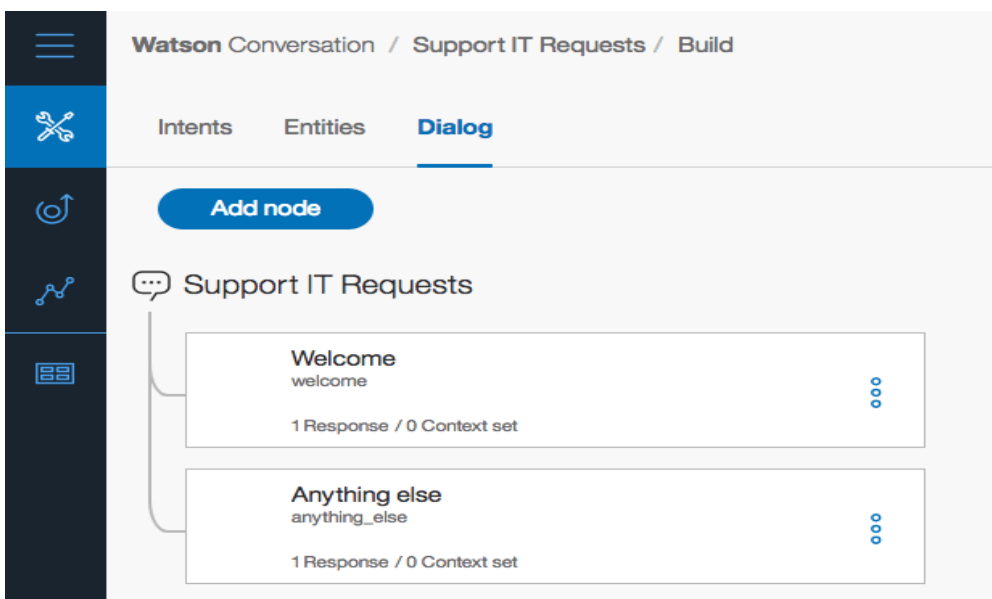
You are now ready to create the dialog flow.

Task 6: Build the dialog flow

After you specify your intents and entities, you can construct the dialog flow.



A dialog is made up of nodes that define steps in the conversation.

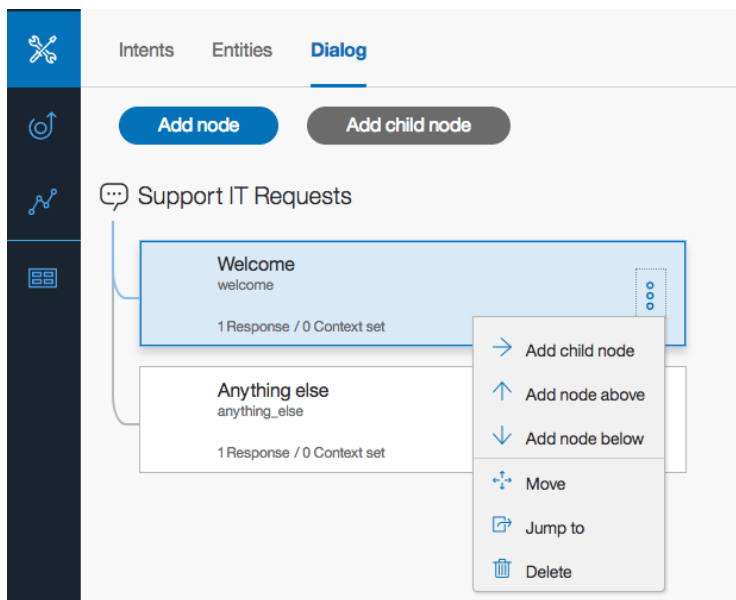


In the previous image, two dialog nodes are shown. The first node is the standard welcome message. The other node is a catch-all node named "Anything else." Dialog nodes are chained in a tree structure to create an interactive conversation with the user. The evaluation starts at the top, so the welcome node is assessed before the "Anything else" node.

If you click the welcome node, the standard Watson response is "Hello. How can I help you?" To validate how the flow works, you can click the **Ask Watson** icon.

Define the greetings node

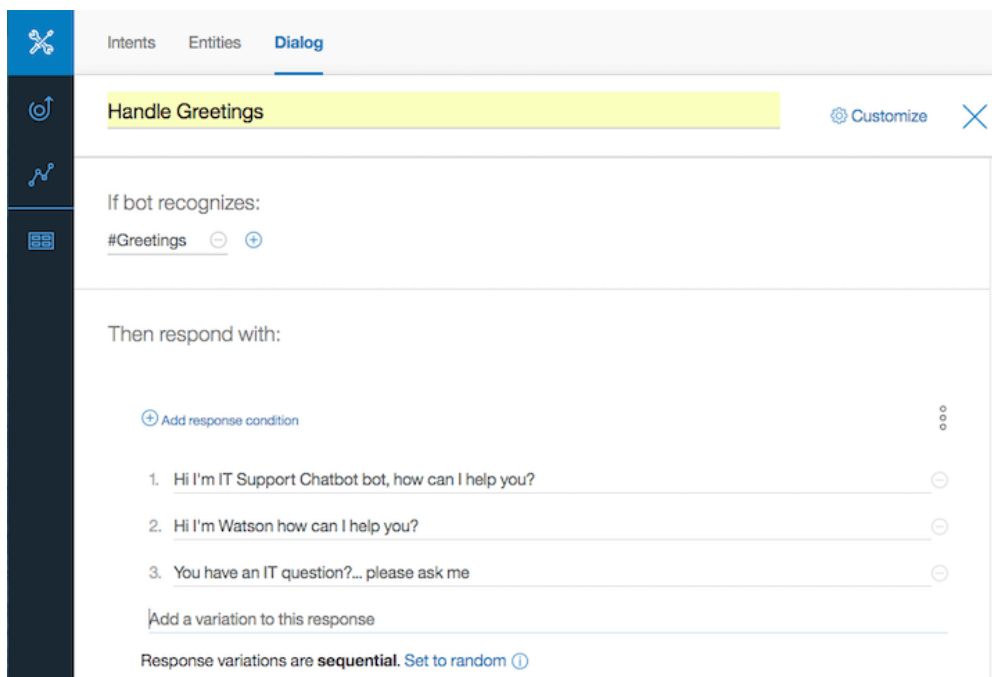
1. The first node addresses greetings in a response to a query such as "hello." Click the welcome node and click **Add node below**:



A new node is added between the welcome and "Anything else" nodes.

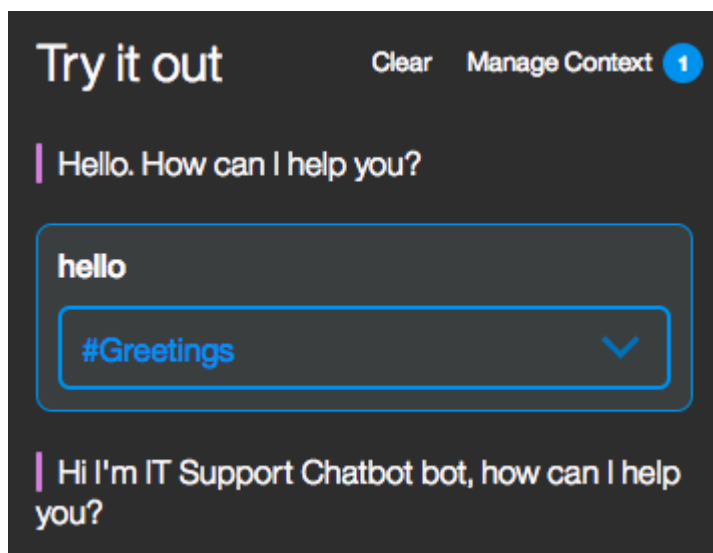
At each node level, you can expand the conversation by adding nodes. If you add nodes at the same level, the flows are parallel. Adding a child node creates a dependent track of conversation, and the conversation branches out into a tree structure.

2. Name the new node Handle Greetings. In the **If bot recognizes** field, change the value to #Greetings. The number sign (#) represents a prefix for intent. The condition is triggered when the Watson natural language classifier classifies the query as a greeting intent.
3. Add these responses:



The previous image also illustrates how to use the multiple responses pattern to avoid being repetitive. The bot can present different answers to the same query. You can allow the system to randomly select an answer from the list of potential responses.

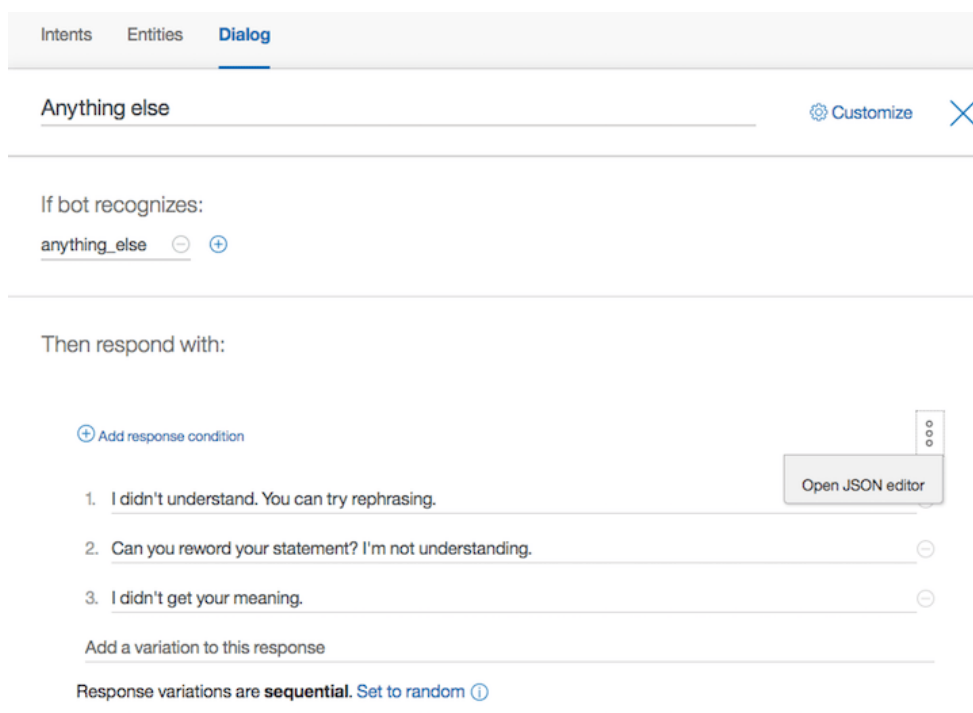
4. Unit-test your dialog by clicking the **Ask Watson** icon:



At the beginning of each conversation, the evaluation starts at the top level of dialog nodes.

Manage the "Anything else" use case

The bottom node is used when none of the defined intents are matched. It can provide a default message, as shown in this image:



From the menu on the right side of the response area, you can open the JSON editor and assess the data that is returned as part of the conversation interaction. The JSON document includes an output JSON object with text with different values. To the output, add an attribute, name it Missing case, and set it to true. When you persist the conversation flow into a document oriented database , you can search the queries that were not addressed by the dialog nodes so that you can add more cases later, if needed.

If bot recognizes:

anything_else - +

Then respond with:

```
1 {
2   "output": {
3     "text": {
4       "values": [
5         "I didn't understand. You can try rephrasing.",
6         "Can you reword your statement? I'm not understanding.",
7         "I didn't get your meaning."
8       ],
9       "selection_policy": "sequential",
10      "Missing case": true
11    }
12  }
13 }
```

Define the application access dialog flow

Now you can create dialog branch to handle the ApplicationAccess intent.

1. Click the "Handle Greetings" node and then click **Add node**. Name the new node Handle application access. In the **If bot recognizes** field, type #ApplicationAccess.

Handle application access Customize ✕

If bot recognizes:

#ApplicationAccess - +

Then respond with:

1 @application:AbC - +

1. To go to application AbC use the following URL : <https://w3caseinc.com/abc56> -

Add a variation to this response

When users enter queries about application access, they will most likely specify one of the supported applications; for example, I want to access application abc. Therefore, in the Application Access node, you need to add multiple conditions by looking at the different possible entities. As illustrated in the previous image, the first condition tests on the presence of the @application:AbC entity. When it matches, the response returns a static URL. Later, you will learn how to make the URL clickable from Watson Assistant in the chatbot application.



2. To add more conditions in the same node, click **Add another response**.

+ Add another response

3. In the condition, click **@application:ExpenseReport** and provide the solution to access the app. Repeat these steps for all the entities that you defined for the application.

Handle application access



 Customize

@application:AbC  


⋮

1. To go to application AbC use the following URL : <https://w3caseinc.com/abc56> 


Add a variation to this response

@application:(Expense Report)  


⋮

1. To access the expense report app, go to Start Menu > Applications> CaseInc Tools > Expense Report 

Add a variation to this response

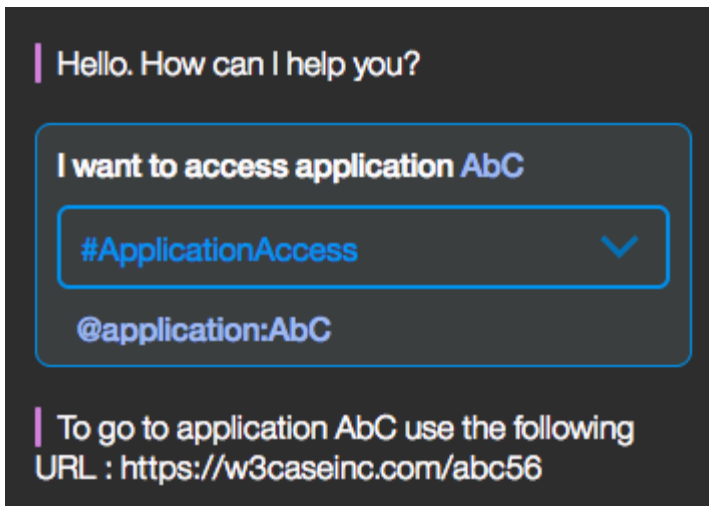
3  @application:(Inventory plus)  

⋮

1. To access inventory plus go to URL: <http://caseincapp.mybluemix.net> and select the 'Inventory Plus' mer 

Add a variation to this response

4. Test the sentence I want to access application AbC. You should get the ApplicationAccess intent and the application AbC entity. Verify that the node match gives the expected response:



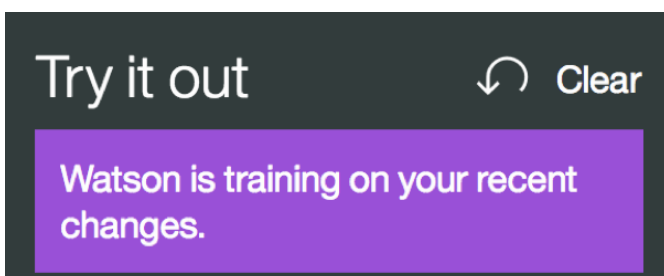
5. If the user does not specify any application, you want to ask which one to access from a list of predefined apps. This is another response pattern where predefined content can be selected by the user. Add a node as shown in this image:



6. If you enter a sentence like Access application, the expected response is proposed.
7. If you enter a choice like Inventory Plus, you might not get the expected result because the classifier might have extracted the wrong intent. From the unit test, you can enforce Watson Assistant to do a better job by selecting the right answer: select the **applicationAccess** intent and Watson will retrain it. Be sure to test again to ensure that the conversation works as expected.

8. Try a few statements that aren't the same as your training data.

Adding any new entity or intent makes Watson Assistant retrain its classifier.



The evaluation round works in two stages. In the first stage, the dialog tries to find an answer in the child nodes of the contextual node. That is, the dialog tries to match all the conditions of the child nodes of the contextual node. If the final child node has a condition of "true,"

meaning that it is to be used if none of its siblings are matched, that node's response is processed.

If no match is found, the dialog continues to a second stage where it tries to find an answer to an input by matching the top level of dialog nodes.

The top level of dialog nodes should contain an `anything_else` node as the last node, which is reached when no match occurs in the conditions of the top-level nodes. Typically, if an `anything_else` node is defined, the dialog returns an answer to every user input.

At this stage, you can build a static flow. Next, you get user input and do something with it.

Task 7: Complete advanced dialog work

Each time the dialog returns a response and waits for user input, the dialog stores the ID of the node where the conversation must resume. This node is called the *contextual node*. Its ID is added to the `context.system.dialog_stack` property, which contains a JSON array of dialog node IDs that are on the dialog stack.

The context object

State information for your conversation is maintained by using the context object.

The context object is a JSON object that is passed between your application and the Watson Assistant service.

This example shows context that is returned from an interaction that reaches a top node:

json

```
{
  "context": {
    "conversation_id": "Conversation-ut",
    "system": {
      "dialog_stack": [
        {
          "dialog_node": "root"
        }
      ],
      "dialog_turn_counter": 1,
      "dialog_request_counter": 1,
```

```

    "_node_output_map": {
      "AbC access": [0]
    },
    "dialog_in_progress": false
  }
}
}


```

Add variables to context

Suppose that you want to return an actionable URL, meaning that the response includes a URL variable that the user can click to go to a new page. To try this scenario, you present the URL of a business process that is deployed on IBM BPM on Cloud.

1. If you didn't import the intents definition from the lab CSV file, add an intent to support the user's query about accessing the "Supplier on boarding business process."

#supplieronboarding

 Add a new user example...

☐ access to supplier due diligence

☐ I want to board new supplier

☐ on boarding new supplier

☐ supplier due diligence

☐ what is the process to on board new supplier?

☐ what is the url of the business process for on boarding supplier?

2. Add a node to the dialog flow under the "Application access" node. Name the node Handling supplier on boarding and for its intent, type #SupplierOnBoarding.
3. In the response, access the Advanced editor to edit the JSON response object:

Supplier On Boarding
Customize
X

If bot recognizes:
#SupplierOnBoarding

Then respond with:

1
Enter an intent, entity or context variable...

1. Do you want me to trigger the process for you?

Add a variation to this response

The response object includes an output object with the text to present and a context object. The context object has a new url variable to give IBM BPM on Cloud access and an action variable to control the behavior of the broker code. When the user enters a question like "I want to access the supplier on boarding business process," the returned object looks like this example:

json

```
{
  "intents": [
    {
      "intent": "supplieronboarding",
      "confidence": 0.940047025680542
    }
  ],
  "entities": [
    {
      "entity": "supplier",
      "value": "supplier"
    }
  ],
  "input": {
    "text": " I want to access the supplier on boarding business process"
  },
}
```

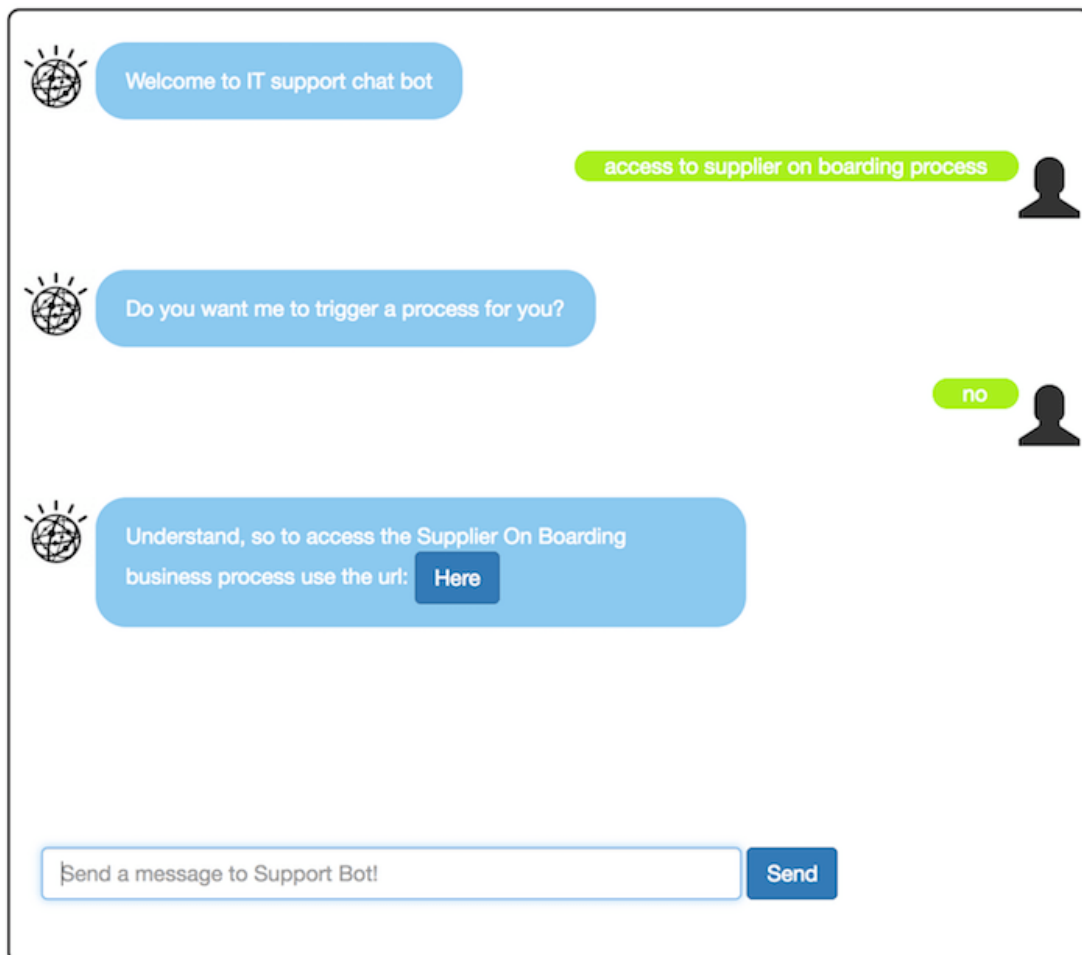
```
"output": {
  "log_messages": [],
  "text": [
    "To access the Supplier On Boarding business process use the url: "
  ],
  "nodes_visited": [
    "Supplier On Boarding",
    "Provide URL for Supplier on boarding"
  ],
},
"context": {
  "conversation_id": "Conversation-ut",
  "system": {
    "dialog_stack": [
      {
        "dialog_node": "root"
      }
    ],
    "dialog_turn_counter": 1,
    "dialog_request_counter": 1
  },
  "url":
  "https://vhost001.bpm.ibmcloud.com/bpm/dev/ProcessPortal/dashboards/SYSRP/RESPONSIVE_WORK",
  "action": "click"
}
```


The code that calls the Watson Assistant API can take the URL value and create a hyperlink in HTML so that the display has an actionable link:

javascript

```
if (rep.context.url != undefined) {  
  if (rep.context.action === "click") {  
    rep.text=rep.output.text[0] + "<a class=\"btn btn-primary\"  
    href=\""+rep.context.url+"\">Here</a>"  
  }  
}
```

The actionable link is shown in the last interaction, with the clickable button added to the message:



Much can be done on the context object. For example, the application code can add elements in the context object before it calls the Watson Assistant service. Then, at the condition level in a node, tests can be done on those elements.

To use a context variable in your condition, use one of these formats:

- `$variable_name:value`
- `$variable_name == 'value'`.

The value of the condition might have been set by the application or in the response portion of a previous dialog node. In the next test, the `canAccessSOD` context variable is a Boolean that is set by accessing an internal authorization service that, for example, returns true if a user ID can access an application.

Use slots

Add a dialog flow to address when a user wants to bring his or her own device. Only certain brands and devices are supported, so you must determine the brand and type of device. The goal is illustrated by this dialog:

Welcome to IT support chat bot

I want to bring my phone

#BYOD

@deviceType:phone

What is the brand of your device?

Apple

Irrelevant

@deviceBrand:Apple

You want to bring a phone from Apple, let me check our policies...

From the first query, "I want to bring my phone," Watson Assistant can get the `#BYOD` intent and the `@deviceType:phone` entity. The dialog flow asks for the brand of the device. If the device type is not extracted, it asks a question about the type of device that the user wants to bring.

One way to support this combination is to add a hierarchy of nodes and code the conditions on entity. In releases from August 2017 and later, you can use slots to get information so that you can accurately respond to the user.

1. If you didn't import the intent and entities, create an entity for the "bring your own device" question.

#BYOD

+

 Add a new user example...

☐ I want to bring my own device

☐ byod

☐ use my phone for business

☐ use my ipad for business

☐ use my own smartphone for work

☐ use my own phone

☐ bring my tablet

☐ bring my laptop

2. Create the @deviceBrand and @deviceType entities.

@deviceBrand

+

 Add a new value

☐ Dell *dell*

☐ Apple *apple*

☐ Samsung *notes* *galaxy* *samsung*

☐ Lenovo *lenovo*

☐ Motorola *nexus* *motorola* *moto*

☐ IBM *ibm*

@deviceType

+

 Add a new value

☐ laptop *computer*

☐ tablet *kindle* *Galaxy* *ipad*

☐ phone *moto g* *nexus* *android* *iphone* *telephone*

3. Add a flow by adding a top-level node that uses the #BYOD intent as the recognize condition. Click the **Customize** menu to enable slots for this node.

Customize "Bring your device"

Slots ⓘ

on

Enable this to gather the information your bot needs to respond to a user within a single node.

☒ **Prompt for everything**

Enable this to ask for multiple pieces of information in a single prompt, so your user can provide them all at once and not be prompted for them one at a time.

Cancel

Apply

Verify that the condition changes to "Then check for."

Bring device

Customize

×

If bot recognizes:

#BYOD − +

Then check for:

0 Manage handlers

	Check for	Save it as	If not present, ask	Type		
1	@deviceBrand	\$deviceBrand	What is the brand of your	Required	⚙	🗑
2	@deviceType	\$deviceType	What is the type of your d	Required	⚙	🗑

+ Add slot

4. Add two slots:

- Check for the @deviceBrand entity and save the result in the \$deviceBrand context variable. If the brand is not present, ask "What is the brand of your device?"

- Check for the @deviceType entity and save the result in the \$deviceType context variable. If the type is not present, ask "What is the type of your device (tablet, smartphone, computer)?"

Slots make it possible for the service to answer follow-up questions without reestablishing the user's goal.

5. For the response, use the advanced dialog to enter output text to display the content of the device type and brand:

Then respond with:

```
1 {
2   "context": {
3     "action": "AssessByodPolicy"
4   },
5   "output": {
6     "text": {
7       "values": [
8         "You want to bring a $deviceType from $deviceBrand, let me check our policies..."
9       ],
10      "selection_policy": "sequential"
11    }
12  }
13 }
```

In the And then action section, be sure to set wait for user input.

Use an interaction to get parameters to call a web service

The last example is more complex, but it represents a common pattern: in the context of a dialog flow, the set of interactions aim to gather input parameters so that after the interaction is done, the broker code can call a web service and pass the parameters as part of the payload.

In this example, the IBM BPM supplier onboarding process is triggered through a SOAP call. Two input parameters exist: the company name and the product name. The client code is in the conversation broker code as a feature in the server/routes/features/supplier-bpm-client.js file.




Because the focus is on the dialog, you must modify the previous flow to handle the "Supplier process" intent. The first node asks whether the user wants the chatbot to trigger the process.




If bot recognizes:

#SupplierOnBoarding  

Then respond with:

 1 Enter an intent, entity or context variable... 

1. Do you want me to trigger the process for you? 

Add a variation to this response

When the response is no, you use the previous behavior by providing the URL to the IBM BPM Process portal so that the user can manually start the process.

1 Response / 3 Context set / 2 Slots


Supplier On Boarding
#SupplierOnBoarding
1 Response / 0 Context set



User wants access only
@yes-no:no
1 Response / 2 Context set

Watson trigger the process
@yes-no:yes
1 Response / 2 Context set

Reset Password
#ResetPassword
1 Response / 0 Context set

Otherwise
True
1 Response / 0 Context set

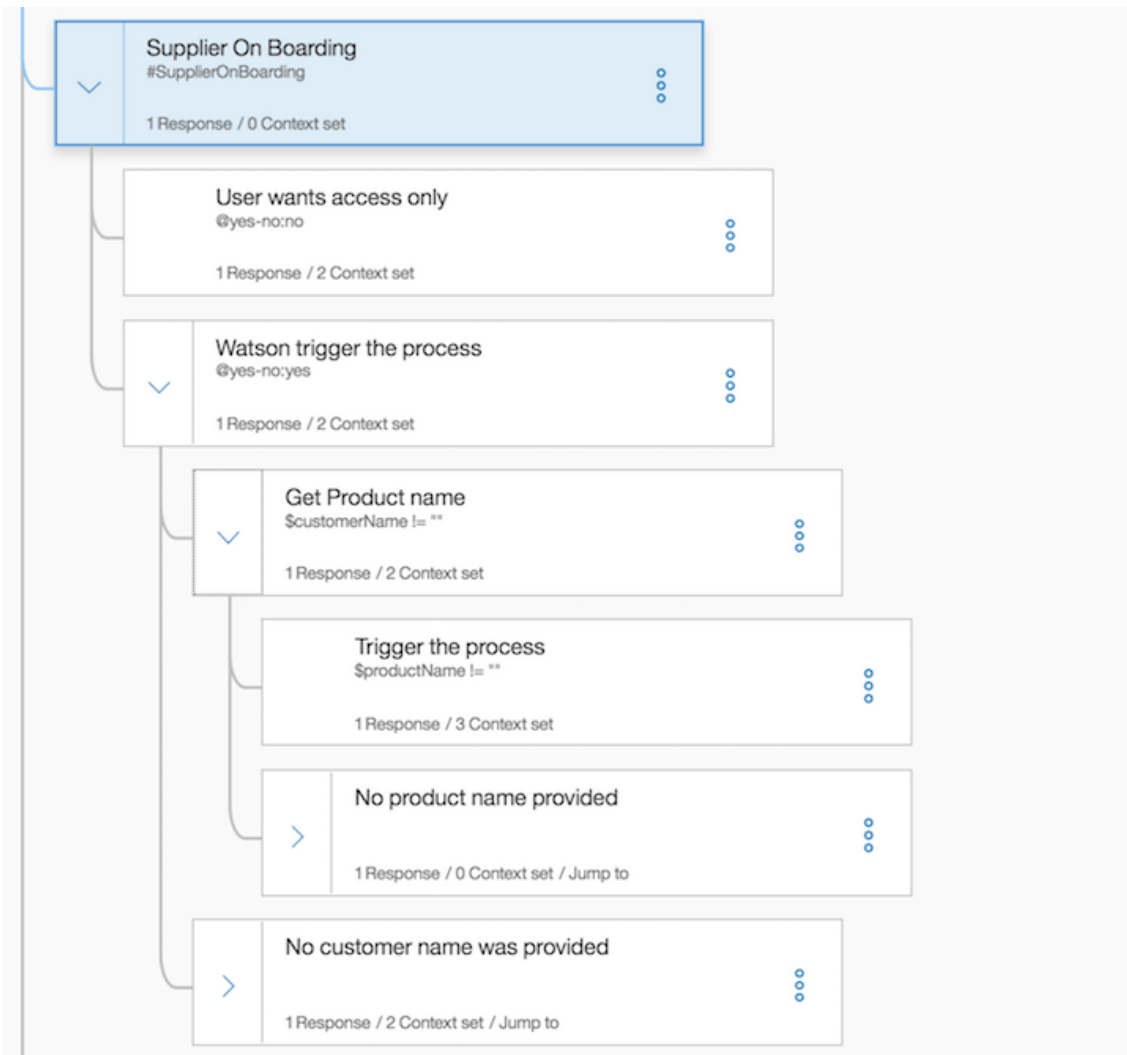
User wants access only 

If bot recognizes:
@yes-no:no  

Then respond with:

```
1 {
2   "context": {
3     "url": "https://vhost001.bpm.ibmcloud.com/bpm/dev/teamworks/executecf?
4     modelID=1.15ecldb4-9051-41d4-8cbe-6053016ed917&snapshotID=2064.f1d3070c-f42a-49c1-
5     b19a-d0646f1f95cba",
6     "action": "click"
7   },
8   "output": {
9     "text": [
10      "Understand, so to access the Supplier On Boarding business process use the
11      url: "
12    ]
13  }
```

When the response is yes, the two next interactions are used to get the company name and product name. You can try to reproduce this structure:



Outside of the output.text file, which provides a question about the company, the context uses two variables to drive the broker code. The action is set to getVar and the varname is set to the name of the variable to add to the context via code. On a yes response, the bot asks about the supplier company name.

Watson trigger the process

@yes-no:yes ⊖ ⊕

Then respond with:

```

1 {
2   "context": {
3     "action": "getVar",
4     "varname": "customerName"
5   },
6   "output": {
7     "text": {
8       "values": [
9         "what is the supplier company name?"
10      ],
11      "selection_policy": "sequential"
12    }
13  }
14 }

```

The broker code dynamically adds the specified variable in the context to keep the data for the next interaction. The following code is in the function to manage the user's response before it calls the Watson Assistant service:

javascript

```
if (req.body.context.action === "getVar") {  
    req.body.context[req.body.context.varname] = req.body.text;  
}
```

The same approach is done for the product name:

The screenshot shows the IBM Watson Assistant interface for configuring a node named "Get Product name". At the top right, there is a "Customize" button with a gear icon. Below the node title, the condition "If bot recognizes:" is followed by the expression "\$customerName != "" with minus and plus icons. Under "Then respond with:", there is a text input field containing "Enter an intent, entity or context variable...". Below this is a code editor with a dark background showing a JSON configuration for the node's response:

```
1 {  
2   "context": {  
3     "action": "getVar",  
4     "varname": "productName"  
5   },  
6   "output": {  
7     "text": {  
8       "values": [  
9         "What is the product name?"  
10      ]  
11     },  
12     "selection_policy": "sequential"  
13   }  
14 }
```

At the bottom right of the code editor, there is a full-screen icon (four arrows pointing outwards).

During the dialog execution, if the user doesn't specify the product or company name, a node in the dialog flow identifies the missing parameters. The node then routes the flow back to the node, asking for the missing parameters. As you can see in the next image, the "Jump to..." constraint is added to the node definition so that the dialog flow returns to the "Get Product name" node.

No product name provided

 Customize



If bot recognizes:

Enter an intent, entity or context variable... 


Then respond with:

 Add response condition



1. I need the product name to trigger the process 

Add a variation to this response

 Add response

And then

Jump to... 

"Get Product name" (Response) 

When all the parameters are set, the interaction sets the action to trigger so that the broker call can perform the SOAP call to the service.



If bot recognizes:

`$productName != ""`

Then respond with:

```
1 {
2   "context": {
3     "action": "trigger",
4     "actionName": "supplierOnBoardingProcess",
5     "actionType": "bpm"
6   },
7   "output": {
8     "text": {
9       "values": [
10        "I am triggering the supplier on boarding process. The process task is
visible at : https://vhost001.bpm.ibmcloud.com/bpm/dev/ProcessPortal"
11      ],
12      "selection_policy": "sequential"
13    }
14  }
15 }
```

Task 8: Use the API

To use the API, you need the service credentials and the tool to perform an HTTP request. For detailed instructions, see [Use Watson Assistant API](#).

Conclusion

Tourism -based chatbots using IBM Watson Assistant can be a valuable tool for both businesses and consumers. They can help businesses to improve customer service, increase sales, and reduce costs. They can also help consumers to save time and money and make easier choices.

Some of the benefits of using IBM Watson Assistant to build a Tourism-based chatbot:

1. **Scalability:** IBM Watson Assistant can scale to meet the needs of businesses of all sizes. This means that your chatbot can handle a large number of users without any performance issues.
2. **Reliability:** IBM Watson Assistant is a highly reliable platform. This means that your chatbot will be available to users when they need it.
3. **Security:** IBM Watson Assistant offers a variety of security features to protect your chatbot and your users' data.

4. **AI capabilities:** IBM Watson Assistant provides a variety of AI capabilities that can be used to make your chatbot more intelligent and conversational. For example, IBM Watson Assistant can be used to generate natural language responses, understand user intent, and learn from user interactions.

Here are some ideas for how we use a food-based chatbot using IBM Watson Assistant:

Creating a tourism-based chatbot using IBM Watson Assistant can greatly enhance the travel experience for users. Here are some ideas on how to use it effectively:

Destination Recommendations:

Ask users about their preferences (e.g., beach, mountains, city) and budget to suggest suitable travel destinations.

Travel Itinerary Planning:

Help users create a personalized itinerary based on their interests, including attractions, activities, and dining options.

Weather and Packing Advice:

Provide weather forecasts for the selected destination and offer packing tips based on the weather conditions.

Local Attractions and Landmarks:

Offer information about popular tourist spots, historical sites, museums, and landmarks.

Accommodation Suggestions:

Recommend hotels, hostels, vacation rentals, or other accommodation options based on budget, preferences, and location.