

Project Report

Name : Sachin Gupta(sachin.gupta.0785@gmail.com)

Title : Intelligent Customer Help Desk With Smart
Document Understanding

Category: Artificial Intelligence

Internship at smartinternz.com@2020

CONTENTS

1 INTRODUCTION

1.1 Overview

1.2 Purpose

2 LITERATURE SURVEY

2.1 Existing problem

2.2 Proposed solution

3 THEORITICAL ANALYSIS

3.1 Block diagram

3.2 Hardware / Software designing

4 EXPERIMENTAL INVESTIGATIONS

5 FLOWCHART

6 RESULT

7 ADVANTAGES & DISADVANTAGES

8 APPLICATIONS

9 CONCLUSION

10 FUTURE SCOPE

11 BIBILOGRAPHY

APPENDIX

A. Source code

B. Reference

INTRODUCTION

1.1 Overview: We will be able to write an application that leverages multiple Watson AI Services (Discovery , Assistant, Cloud function and Node Red). By the end of the project, we'll learn best practices of combining Watson services, and how they can build interactive information retrieval systems with Discovery + Assistant.

- ❖ **Project Requirements:** Python, IBM Cloud, IBM Watson
- ❖ **Functional Requirements:** IBM cloud
- ❖ **Technical Requirements:** AI,ML,WATSON AI,PYTHON
- ❖ **Software Requirements:** Watson assistant, Watson discovery.
- ❖ **Project Deliverables:** Smartinternz Internship
- ❖ **Project Team:** Sachin Gupta
- ❖ **Project Duration:**1 month

1.2 Purpose:

The typical customer care chatbot can answer simple questions, such as store locations and hours, directions, and maybe even making appointments. When a question falls outside of the scope of the pre-determined question set, the option is typically to tell the customer the question isn't valid or offer to speak to a real person.

In this project, there will be another option. If the customer question is about the operation of a device, the application shall pass the question onto Watson Discovery Service, which has been pre-loaded with the device's owners manual. So now, instead of "Would you like to speak to a customer representative?" we can return relevant sections of the owners manual to help solve our customers' problems.

To take it a step further, the project shall use the Smart Document Understanding feature of Watson Discovery to train it on what text in the owners manual is important and what is not. This will improve the answers returned from the queries.

1.2.1 Scope of Work

- ✓ Create a customer care dialog skill in Watson Assistant
- ✓ Use Smart Document Understanding to build an enhanced Watson Discovery collection
- ✓ Create an IBM Cloud Functions web action that allows Watson Assistant to post queries to Watson Discovery
- ✓ Build a web application with integration to all these services & deploy the same on IBM Cloud Platform

LITERATURE SURVEY

2.1 Existing problem:

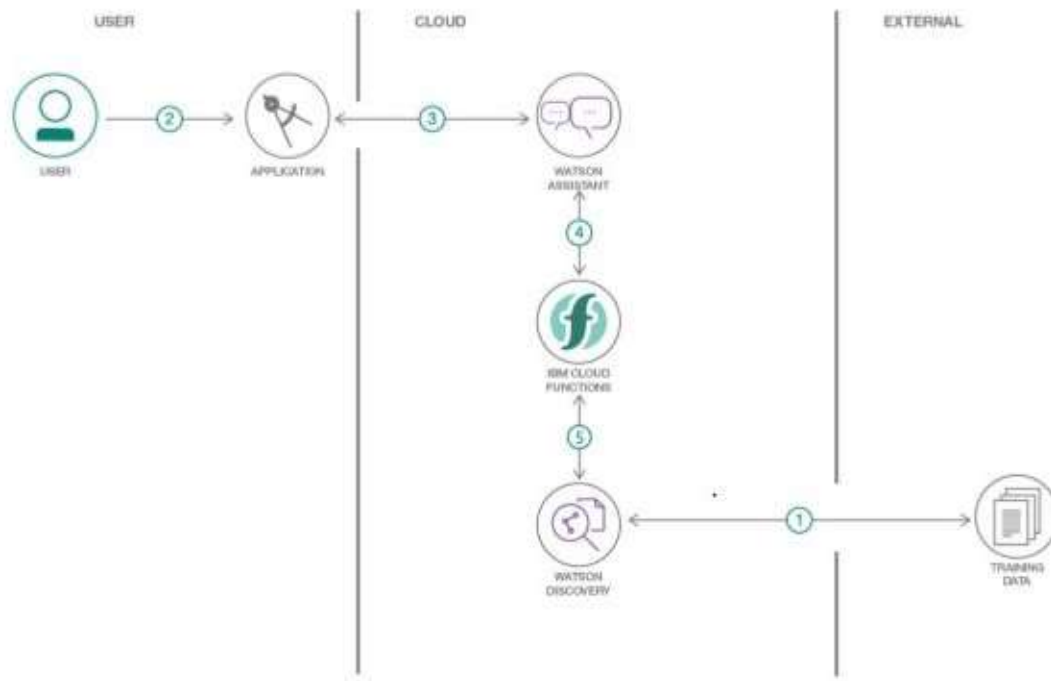
Generally Chatbots means getting input from users and getting only response questions and for some questions the output from bot will be like “try again”, “I don’t understand”, “will you repeat again”, and so on... and directs customer to customer agent but a good customer Chatbot should minimize involvement of customer agent to chat with customer to clarify his/her doubts. So to achieve this we should include an virtual agent in chatbot so that it will take care of real involvement of customer agent and customer can clarifies his doubts with fast chatbots.

2.2 Proposed solution:

For the above problem to get solved we have to put an virtual agent in chatbot so it can understand the queries that are posted by customers. The virtual agent should trained from some insight records based company background so it can answer queries based on the product or related to company. In this project I used Watson Discovery to achieve the above solution. And later including Assistant and Discovery on Node-RED.

THEORITICAL ANALYSIS

3.1 Block/Flow Diagram



1. The document is annotated using Watson Discovery SDU
2. The user interacts with the backend server via the app UI. The frontend app UI is a chatbot that engages the user in a conversation.
3. Dialog between the user and backend server is coordinated using a Watson Assistant dialog skill.
4. If the user asks a product operation question, a search query is passed to a predefined IBM Cloud Functions action.
5. The Cloud Functions action will query the Watson Discovery service and return the results.

3.2 Hardware / Software designing:

1. Create IBM Cloud services
2. Configure Watson Discovery
3. Create IBM Cloud Functions action
4. Configure Watson Assistant
5. Create flow and configure node
6. Deploy and run Node Red app.

EXPERIMENTAL INVESTIGATIONS

1. Create IBM Cloud services

Create the following services:

- Watson Discovery
- Watson Assistant
- Node Red

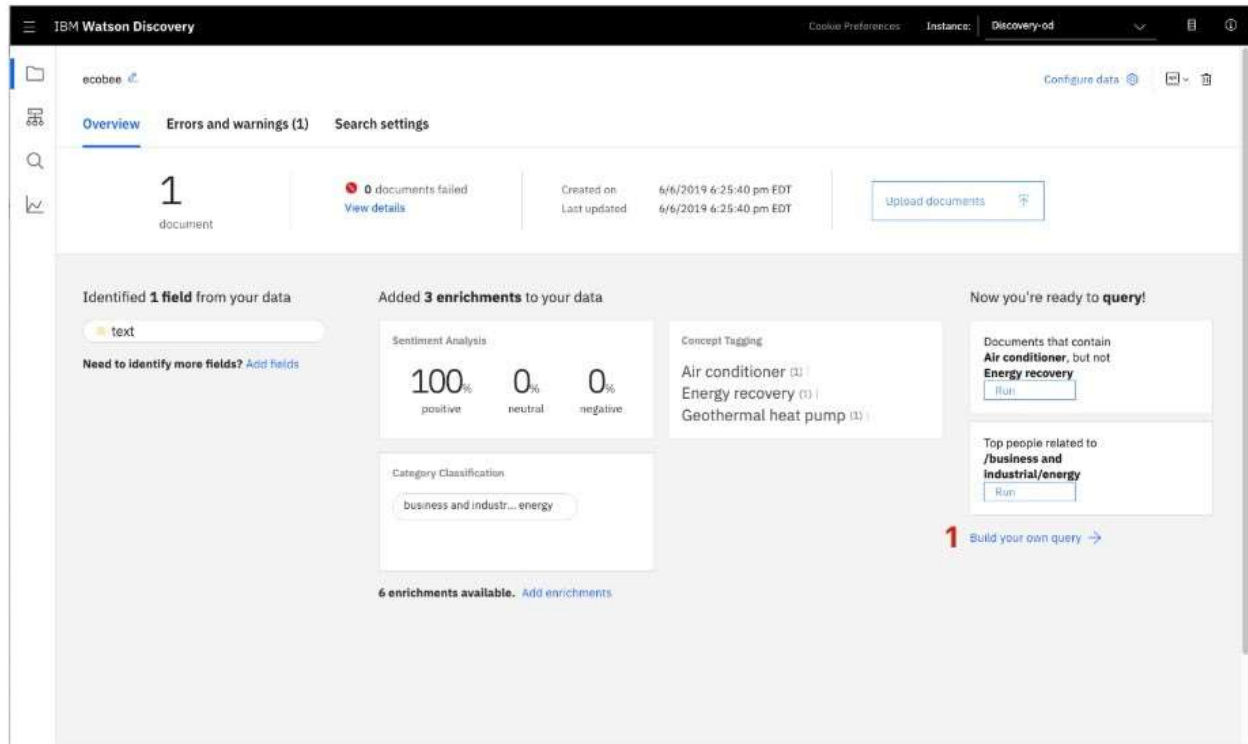
2. Configure Watson Discovery

Import the document

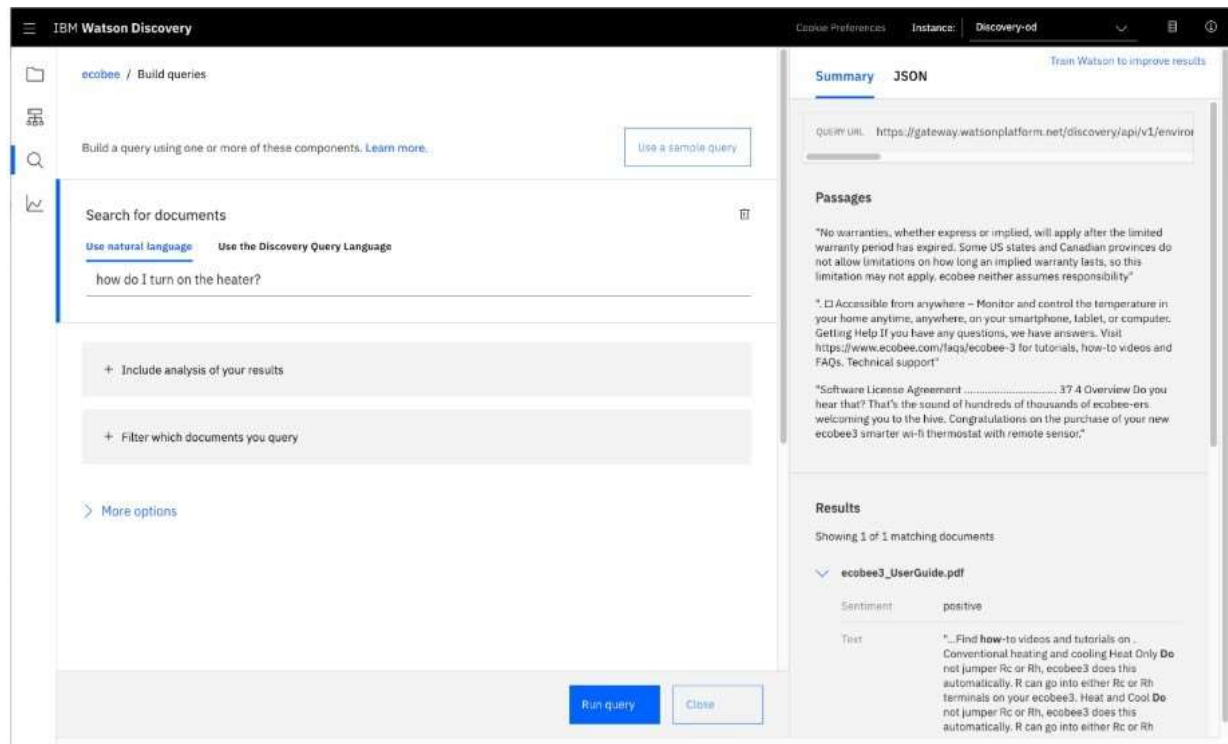
Launch the Watson Discovery tool and create a new data collection by selecting the Upload your own data option. Give the data collection a unique name. When prompted, select and upload the ecobee3_UserGuide.pdf file located in the data directory of your local repo.

The Ecobee is a popular residential thermostat that has a wifi interface and multiple configuration options.

Before applying SDU to our document, lets do some simple queries on the data so that we can compare it to results found after applying SDU.



Click the Build your own query [1] button.

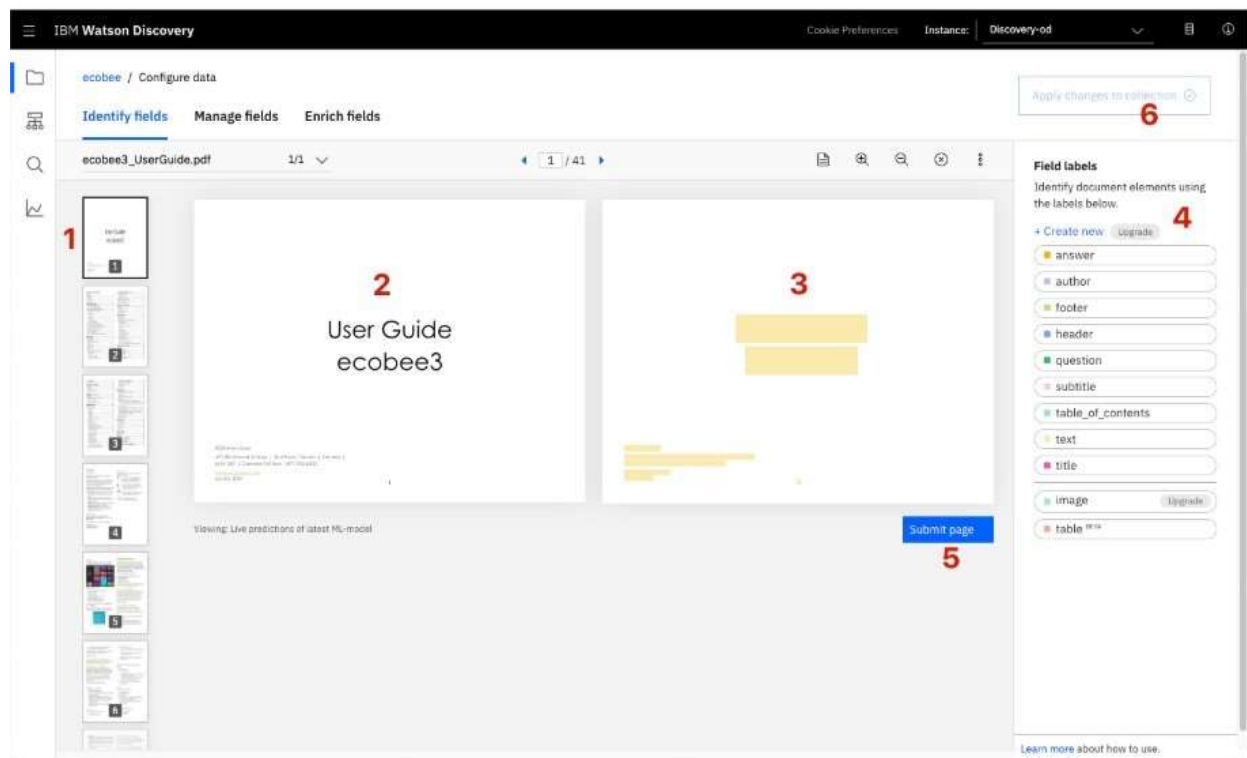


Enter queries related to the operation of the thermostat and view the results. As you will see, the results are not very useful, and in some cases, not even related to the question.

Annotate with SDU

Now let's apply SDU to our document to see if we can generate some better query responses. From the Discovery collection panel, click the Configure data button (located in the top right corner) to start the SDU process.

Here is the layout of the Identify fields tab of the SDU annotation panel:



The goal is to annotate all of the pages in the document so Discovery can learn what text is important, and what text can be ignored.

[1] is the list of pages in the manual. As each is processed, a green check mark will appear on the page.

[2] is the current page being annotated.

[3] is where you select text and assign it a label.

[4] is the list of labels you can assign to the page text.

Click [5] to submit the page to Discovery.

Click [6] when you have completed the annotation process.

As you go through the annotations one page at a time, Discovery is learning and should start automatically updating the upcoming pages. Once you get to a page that is already correctly annotated, you can stop, or simply click Submit [5] to acknowledge it is correct. The more pages you annotate, the better the model will be trained.

For this specific owner's manual, at a minimum, it is suggested to mark the following:

The main title page as title

The table of contents (shown in the first few pages) as table_of_contents

All headers and sub-headers (typed in light green text) as a subtitle

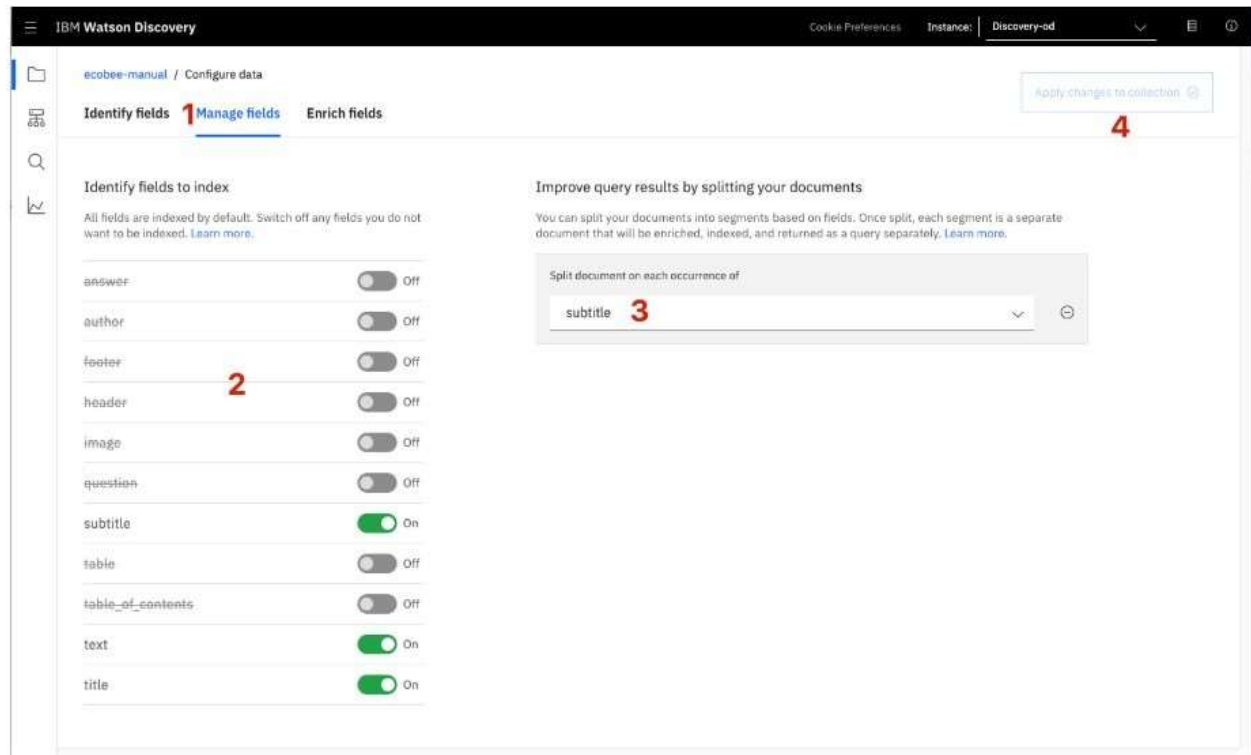
All page numbers as footers

All warranty and licensing information (located in the last few pages) as a footer

All other text should be marked as text.

Once you click the Apply changes to collection button [6], you will be asked to reload the document. Choose the same owner's manual .pdf document as before.

Next, click on the Manage fields [1] tab.



[2] Here is where you tell Discovery which fields to ignore. Using the on/off buttons, turn off all labels except subtitles and text.

[3] is telling Discovery to split the document apart, based on subtitle.

Click [4] to submit your changes.

Once again, you will be asked to reload the document.

Now, as a result of splitting the document apart, your collection will look very different:

The screenshot shows the IBM Watson Discovery Overview page for the 'ecobee-manual' dataset. The page displays 130 documents, 0 failed documents, and creation/last updated dates of 3/28/2019 4:27:53 pm EDT. It lists identified fields (footer, subtitle, table_of_contents, text, title) and added enrichments (Entity Extraction, Sentiment Analysis, Concept Tagging, Category Classification). The Entity Extraction results show 0.3°C (4), 0.5°F (4), 10 °F (4), and 900 seconds (4) / 20 min (3). Sentiment Analysis shows 37% positive, 26% neutral, and 36% negative. Concept Tagging results include Heat (1.7), Internet (1.4), HVAC (1.3), Netscape (1.3), and Temperature (1.3). Category Classification results include 'technology and com...' and 'operating systems'. The page also includes a 'Build your own query' link.

Return to the query panel (click Build your own query) and see how much better the results are.

The screenshot shows the IBM Watson Discovery Query panel for the 'ecobee-manual' dataset. The search query is 'how do I turn on the heater?'. The panel includes options to 'Include analysis of your results' and 'Filter which documents you query'. The search results are displayed in a list, showing 10 of 38 matching documents. The first result is 'ecobee3_UserGuide.pdf'. The panel also includes a 'Run query' button and a 'Close' button.

Store credentials for future use

In upcoming steps, you will need to provide the credentials to access your Discovery collection. The values can be found in the following locations.

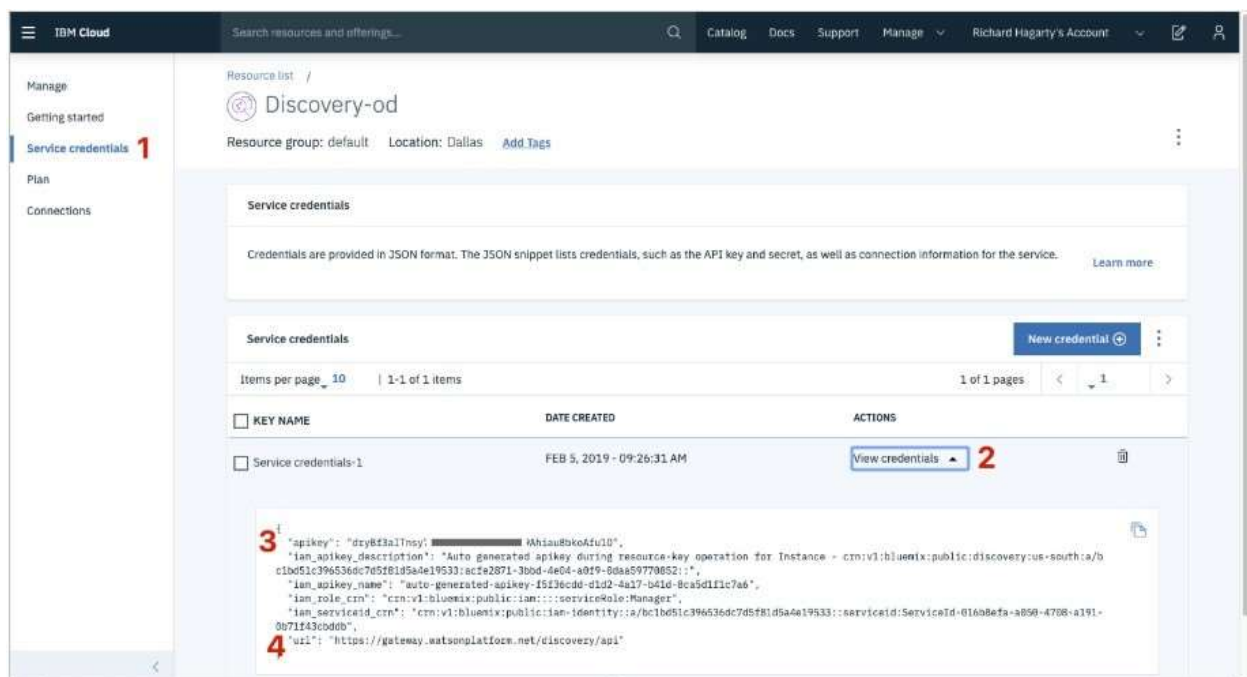
The Collection ID and Environment ID values can be found by clicking the dropdown button [1] located at the top right side of your collection panel:



Parameters ⓘ Add Parameter

Parameter Name	Parameter Value
url	"https://api.eu-gb.discovery.watson.cloud.ibm.com/instances/5c8e95e2-f032-4f13"
collection_id	"3a191d05-bc88-48ee-8276-548f59061a56"
environment_id	"c520a71c-e99b-4e41-a637-f632fbbe9680"
iam_apikey	"bMAJb2Nm_0-MpD_MlBAaCc1qeOT4YiFtpS44ox04bJhJ"

For credentials, return to the main panel of your Discovery service, and click the Service credentials [1] tab:



IBM Cloud Search resources and offerings... Catalog Docs Support Manage Richard Hagarty's Account

Manage
Getting started
Service credentials 1
Plan
Connections

Resource list: /
Discovery-od
Resource group: default Location: Dallas Add Tags

Service credentials

Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service. [Learn more](#)

Service credentials New credential

Items per page 10 | 1-1 of 1 items 1 of 1 pages

KEY NAME	DATE CREATED	ACTIONS
<input type="checkbox"/> Service credentials-1	FEB 5, 2019 - 09:26:31 AM	View credentials 2

3

```
{
  "apikey": "dry8f3allnoy: [REDACTED]khsau8bkoafu10",
  "iam_apikey_description": "Auto generated apikey during resource-key operation for Instance - crn:v1:bluemix:public:discovery:us-south:a/bc1bd51c396536dc7d5f81d5a4e19533:acf2871-3b0d-4e04-a019-0da69779052::",
  "iam_apikey_name": "auto-generated-apikey-f5f3ecdd-d1d2-4a17-b41d-8ca5d1f1c7a6",
  "iam_role_crn": "crn:v1:bluemix:public:iam:::serviceRole:Manager",
  "iam_serviceid_crn": "crn:v1:bluemix:public:iam-identity::a/bc1bd51c396536dc7d5f81d5a4e19533::serviceid:ServiceId-016a8efa-a890-4708-a191-0b71f43c0d0b",
  "url": "https://gateway.watsonplatform.net/discovery/api"
}
```

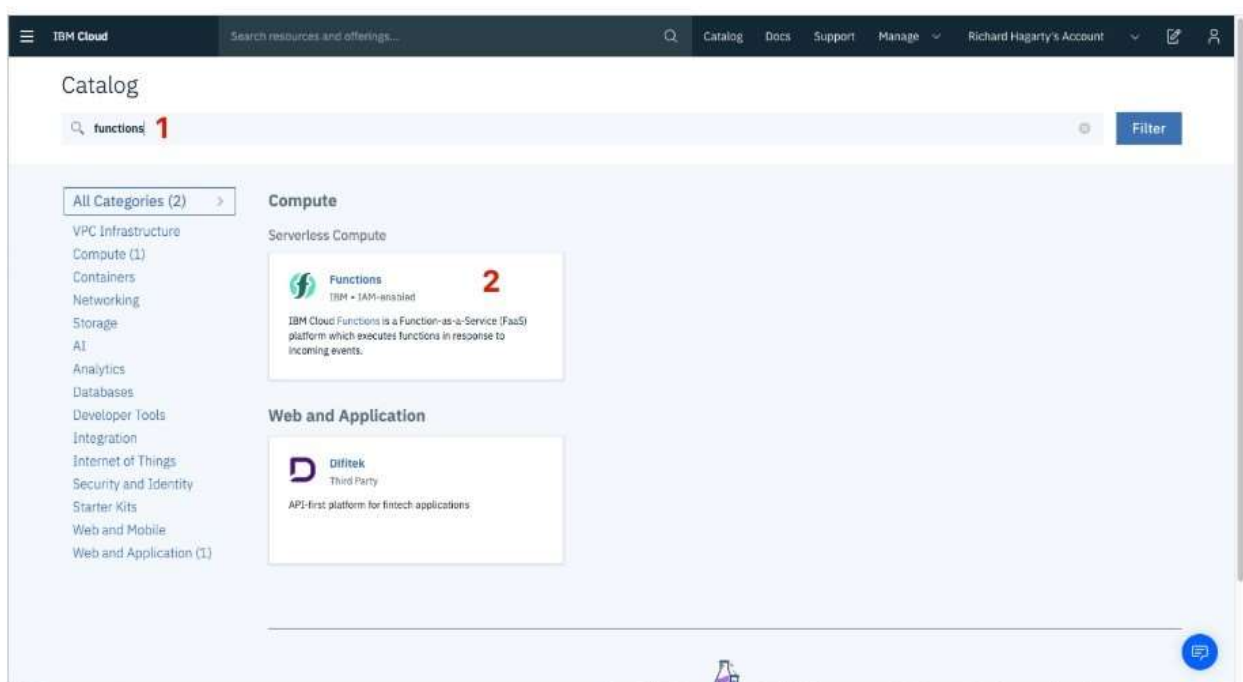
4

Click the View credentials [2] drop-down menu to view the IAM apikey [3] and URL endpoint [4] for your service.

3. Create IBM Cloud Functions action

Now let's create the web action that will make queries against our Discovery collection.

Start the IBM Cloud Functions service by selecting Create Resource from the IBM Cloud dashboard. Enter functions as the filter [1], then select the Functions card [2]:



From the Functions main panel, click on the Actions tab. Then click on Create.

From the Create panel, select the Create Action option.

On the Create Action panel, provide a unique Action Name [1], keep the default package [2], and select the Node.js 10 [3] runtime. Click the Create button [4] to create the action.

IBM Cloud

Search resources and offerings...

Functions

Getting Started

Actions

Triggers

APIs

Monitor

Logs

Namespace Settings

Create Action

Actions contain your function code and are invoked by events or REST API calls.

[Learn more about Actions](#)

[Learn more about Packages](#)

Action Name

disco-action-2

Enclosing Package

(Default Package)

Create Package

Runtime

Node.js 10

Looking for Java, .NET or Docker? [Docker](#) Actions can be created with the [CLI](#)

Cancel

Previous

Create

Once your action is created, click on the Code tab [1]:



In the code editor window [2], cut and paste in the code from the disco-action.js file found in the actions directory of your local repo. The code is pretty straight-forward - it simply connects to the Discovery service, makes a query against the collection, then returns the response.

If you press the Invoke button [3], it will fail due to credentials not being defined yet. We'll do this next.

Select the Parameters tab [1]:



Parameters ⓘ Add Parameter

Parameter Name	Parameter Value
url	"https://api.eu-gb.discovery.watson.cloud.ibm.com/instances/5c8d95e2-f032-4f13"
collection_id	"3a191d05-bc88-48ee-8276-5f8f59061a56"
environment_id	"c520a71c-e99b-4e41-a637-f632fbb9680"
iam_apikey	"bMAJb2Nm_0-MpD_MfBAaCc1qeOT4YiFtpS44oxO4bJhJ"

Add the following keys:

- ❖ url
- ❖ environment_id
- ❖ collection_id
- ❖ iam_apikey

For values, please use the values associated with the Discovery service you created in the previous step.

Now that the credentials are set, return to the Code panel and press the Invoke button again. Now you should see actual results returned from the Discovery service:

IBM Cloud Search resources and offerings...

Functions / Actions / disco-action Namespace: IBM Cloud Storage_DSX-journey-2 Location: Dallas

disco-action Web Action

Code Node.js 10 Change Input Invoke

```

1 // **
2 *
3 * @param {object} params
4 * @param {string} params.iap_apikey
5 * @param {string} params.url
6 * @param {string} params.username
7 * @param {string} params.password
8 * @param {string} params.environment_id
9 * @param {string} params.collection_id
10 * @param {string} params.configuration_id
11 * @param {string} params.input
12 *
13 * @return {object}
14 *
15 */
16
17 const assert = require('assert');
18 const DiscoveryV1 = require('watson-developer-cloud/discovery/v1');
19
20 // **
21 *
22 * main() will be run when you invoke this action
23 *
24 * @param Cloud Functions actions accept a single parameter, which must be a JSON object.
25 *
26 * @return The output of this action, which must be a JSON object.
27 *
28 */
29 function main(params) {
30   return new Promise(function (resolve, reject) {
31
32     let discovery;
33
34     if (params.iap_apikey){
35       discovery = new DiscoveryV1({
36         'iap_apikey': params.iap_apikey,
37         'url': params.url,
38         'version': '2019-03-25'

```

Activations Collapse Clear

disco-action 1050 ms 6/6/2019, 10:45:14

Activation ID: elbfc0ff21544c85bfc0ff21549c85a1

Results:

```

{
  "matching_results": 14,
  "passages": [],
  "results": [
    {
      "enriched_text": {
        "categories": [
          {
            "label": "/technology and computing/operating systems",
            "score": 0.842285
          },
          {
            "label": "/technology and computing/hardware/computer",
            "score": 0.835879
          },
          {
            "label": "/technology and computing/hardware/computer peripherals/computer monitors",
            "score": 0.832254
          }
        ],
        "concepts": [
          {
            "dbpedia_resource": "http://dbpedia.org/resource/iphone",
            "relevance": 0.917304,
            "text": "iphone"
          },
          {
            "dbpedia_resource": "http://dbpedia.org/resource/Personal_digital_assistant",
            "relevance": 0.887088,
            "text": "Personal digital assistant"
          }
        ]
      }
    }
  ]
}

```

Next, go to the Endpoints panel [1]:

IBM Cloud Search resources and offerings...

Functions / Actions / disco-action Namespace: IBM Cloud Storage_DSX-journey-2 Location: Dallas

disco-action Web Action

Code Parameters Runtime Endpoints 1 Connected Triggers Enclosing Sequences Logs

Web Action

2 ☒ Enable as Web Action Allow your Cloud Functions actions to handle HTTP events. Learn more about Web Actions.

☐ Raw HTTP handling When enabled your Action receives requests in plain text instead of a JSON body

HTTP METHOD	AUTH	URL
ANY	Public	3 https://us-south.functions.cloud.ibm.com/api/v1/web/IBM%20Cloud%20Storage_DSX-journey-2/default/disco-action

REST API

HTTP METHOD	AUTH	URL
POST	API-KEY	https://us-south.functions.cloud.ibm.com/api/v1/namespaces/IBM%20Cloud%20Storage_DSX-journey-2/actions/disco-action

CURL

4 `curl -u API-KEY -X POST https://us-south.functions.cloud.ibm.com/api/v1/namespaces/IBM%20Cloud%20Storage_DSX-journey-2/actions/disco-action?blocking=true`

Click the checkbox for Enable as Web Action [2]. This will generate a public endpoint URL [3].

Take note of the URL value [3], as this will be needed by Watson Assistant in a future step.

To verify you have entered the correct Discovery parameters, execute the provide curl command [4]. If it fails, re-check your parameter values.

4. Configure Watson Assistant

Launch the Watson Assistant tool and create a new dialog skill. Select the Use sample skill option as your starting point. This dialog skill contains all of the nodes needed to have a typical call center conversation with a user.

Add new intent

The default customer care dialog does not have a way to deal with any questions involving outside resources, so we will need to add this.

Create a new intent that can detect when the user is asking about operating the Ecobee thermostat.

From the Customer Care Sample Skill panel, select the Intents tab.

Click the Create intent button.

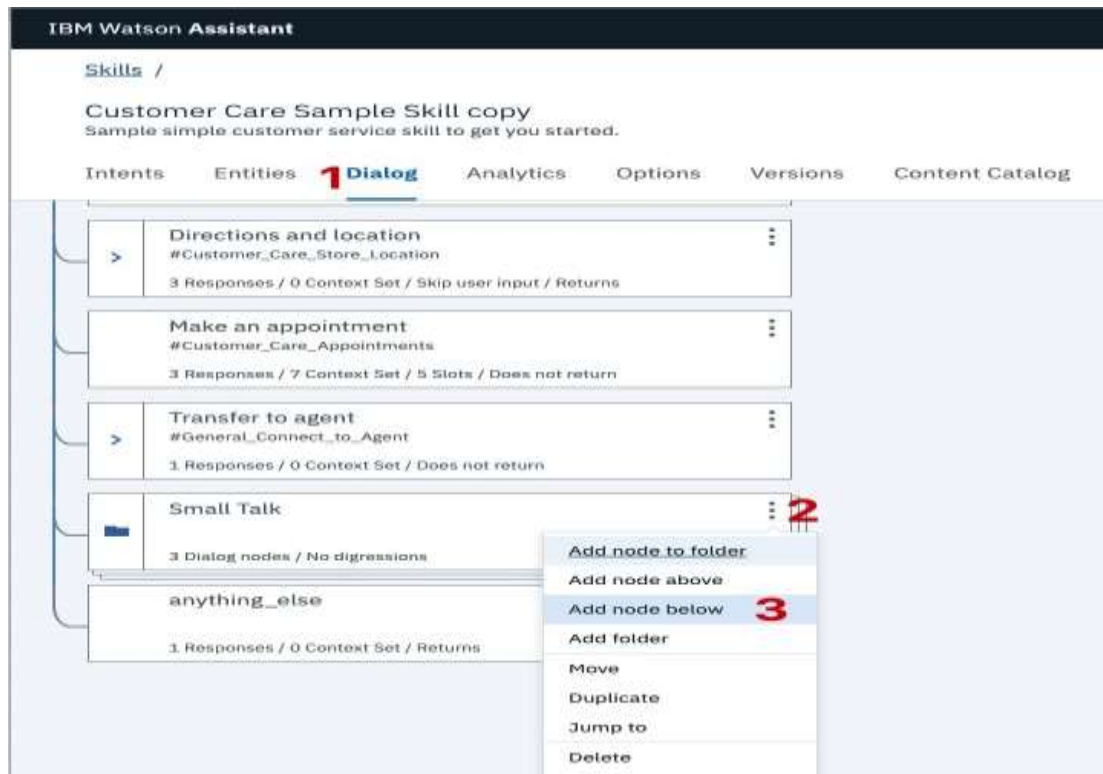
Name the intent #Product_Information, and at a minimum, enter the following example questions to be associated with it.

The screenshot shows the 'Create intent' interface for an intent named '#Product_Information'. The interface includes a header with a back arrow, the intent name, and a 'Try it' button. Below the header, there are three main sections: 'Intent name', 'Description (optional)', and 'Add user example'. The 'Intent name' section shows the name '#Product_Information' and a small icon. The 'Description (optional)' section shows the text 'User wants help using the thermostat'. The 'Add user example' section has a text input field with the placeholder 'Type a user example here' and two buttons: 'Add example' and 'Show recommendations'. Below these sections, there is a table of user examples. The table has three columns: a checkbox, the user example text, and the time it was added. There are three examples listed, all added '2 hours ago'. At the bottom right of the table, there are controls for 'Added', '0 conflicts', a toggle for 'Show only conflicts', and an information icon.

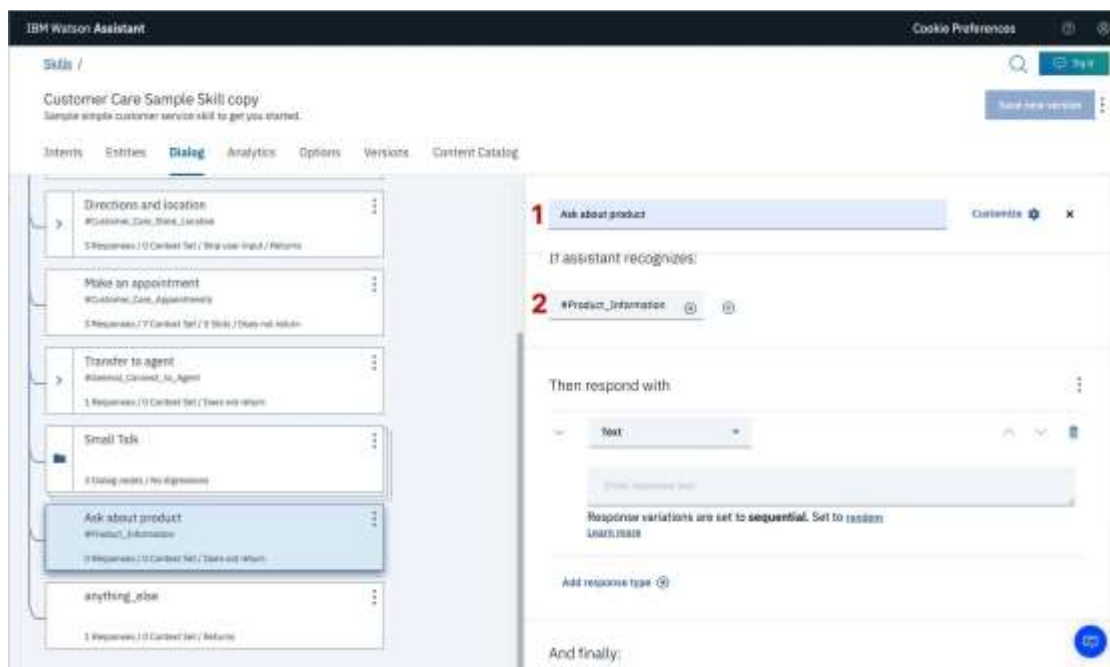
<input type="checkbox"/>	User examples (3) ▼	Added	0 conflicts	Show only conflicts ⓘ
<input type="checkbox"/>	How do I access the settings ✎	2 hours ago		
<input type="checkbox"/>	How do I set the time ✎	2 hours ago		
<input type="checkbox"/>	How do I turn on the heater ✎	2 hours ago		

Create new dialog node

Now we need to add a node to handle our intent. Click on the Dialog [1] tab, then click on the drop down menu for the Small Talk node [2], and select the Add node below [3] option.



Name the node "Ask about product" [1] and assign it our new intent [2].

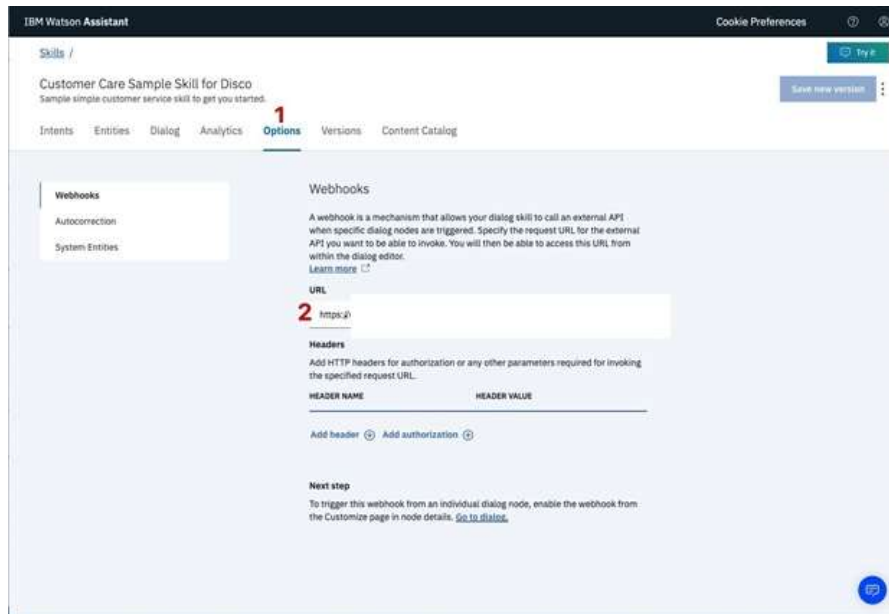


This means that if Watson Assistant recognizes a user input such as "how do I set the time?", it will direct the conversation to this node.

Enable webhook from Assistant

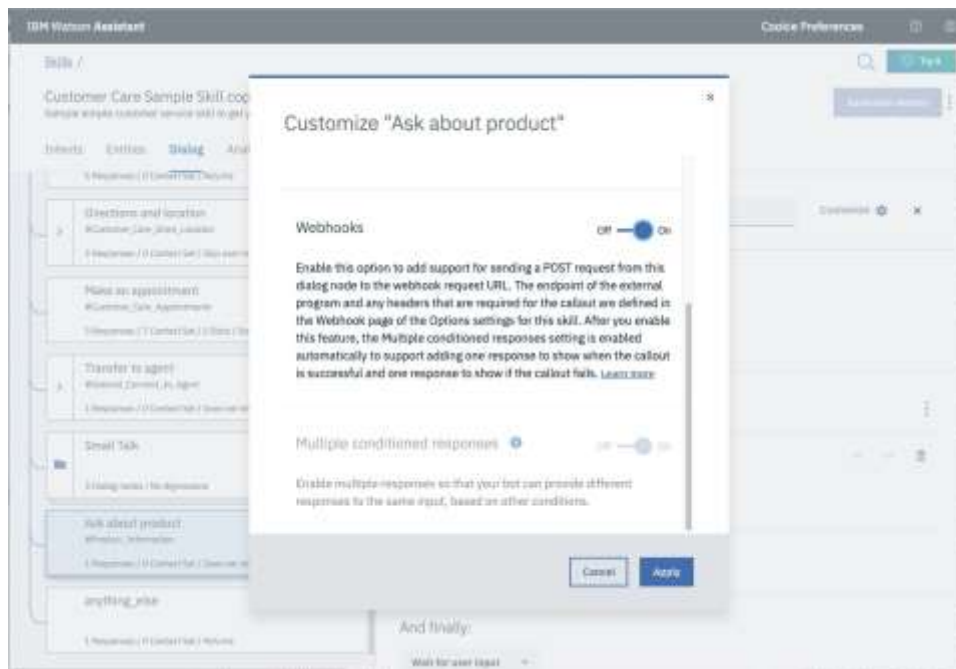
Set up access to our WebHook for the IBM Cloud Functions action you created in Step #4.

Select the Options tab [1]:



Enter the public URL endpoint for your action [2].

Return to the Dialog tab, and click on the Ask about product node. From the details panel for the node, click on Customize, and enable Webhooks for this node:



Click Apply.

The dialog node should have a Return variable [1] set automatically to \$webhook_result_1. This is the variable name you can use to access the result from the Discovery service query.

The screenshot shows the IBM Watson Assistant Skills editor interface. On the left, a list of dialog nodes is shown, with 'Ask about product' selected. The main area displays the configuration for this node. It starts with a trigger phrase 'Ask about product'. Below this, it says 'If assistant recognizes:' followed by the entity '#Product_Information'. Then, it says 'Then callout to my webhook:'. Under this, there is a 'Parameters' table with one row: 'input' with the value '<?input.text?>'. Below the parameters, there is a 'Return variable' section with one row: '\$webhook_result_1'.

KEY	VALUE
input	<?input.text?>

Return variable
\$webhook_result_1

You will also need to pass in the users question via the parameter input [2]. The key needs to be set to the value: "<?input.text?>"

If you fail to do this, Discovery will return results based on a blank query.

Optionally, you can add these responses to aid in debugging:

Return variable

\$webhook_result_1

Then respond with

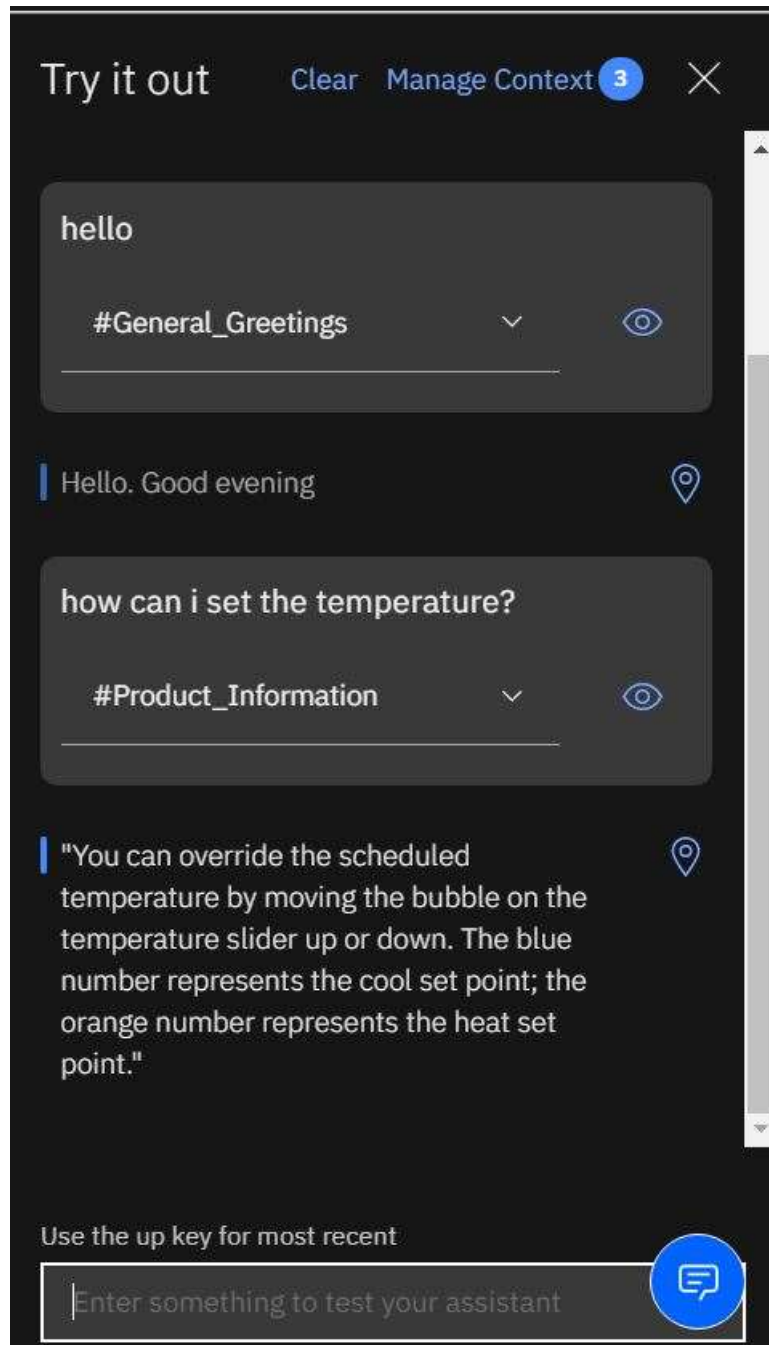
	IF ASSISTANT RECOGNIZES	RESPOND WITH		
1	\$webhook_result_1	\$webhook_result_1		
2	anything_else	Try again later		

Add response

Test in Assistant Tooling

From the Dialog panel, click the Try it button located at the top right side of the panel.

Enter some user input:



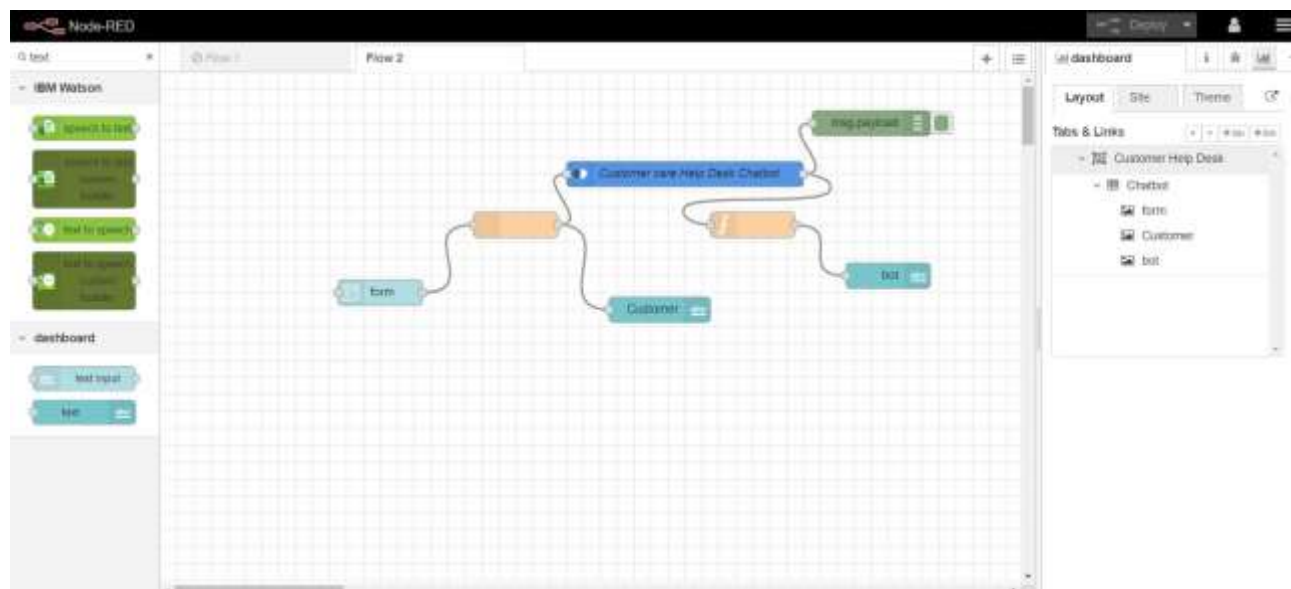
Note that the input "how do I turn on the heater?" has triggered our Ask about product dialog node, which is indicated by the #Product_Information response.

And because we specified that \$webhook_result_1.passages be the response, that value is displayed also. You can also verify that the call was successfully completed by clicking on the

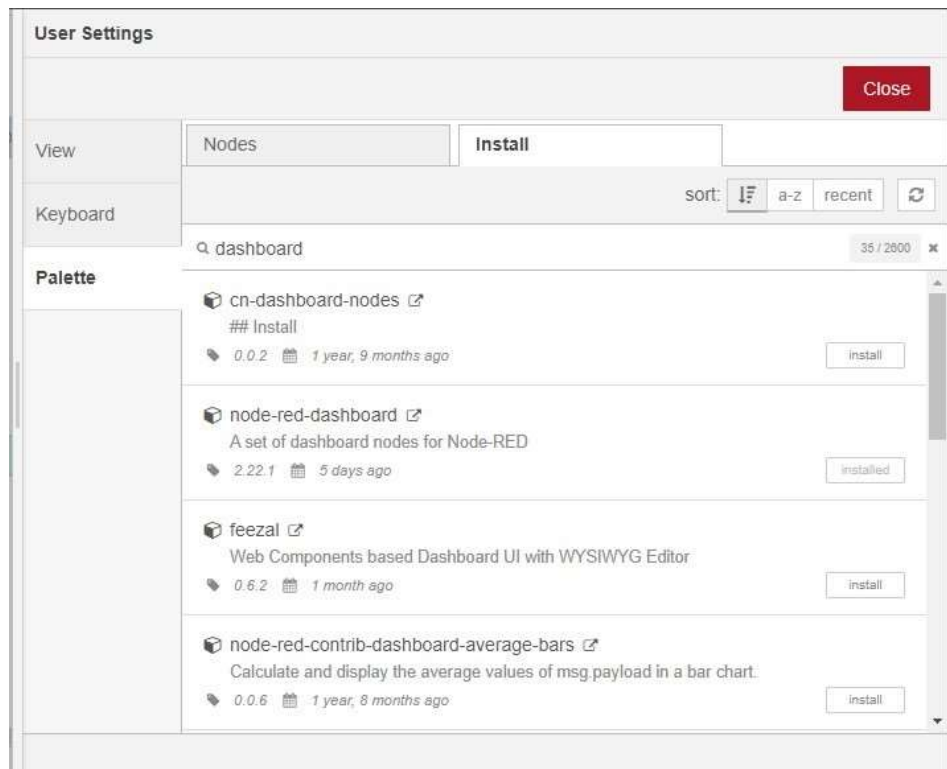
5. Create flow and configure node:

Integration of watson assistant in Node-RED

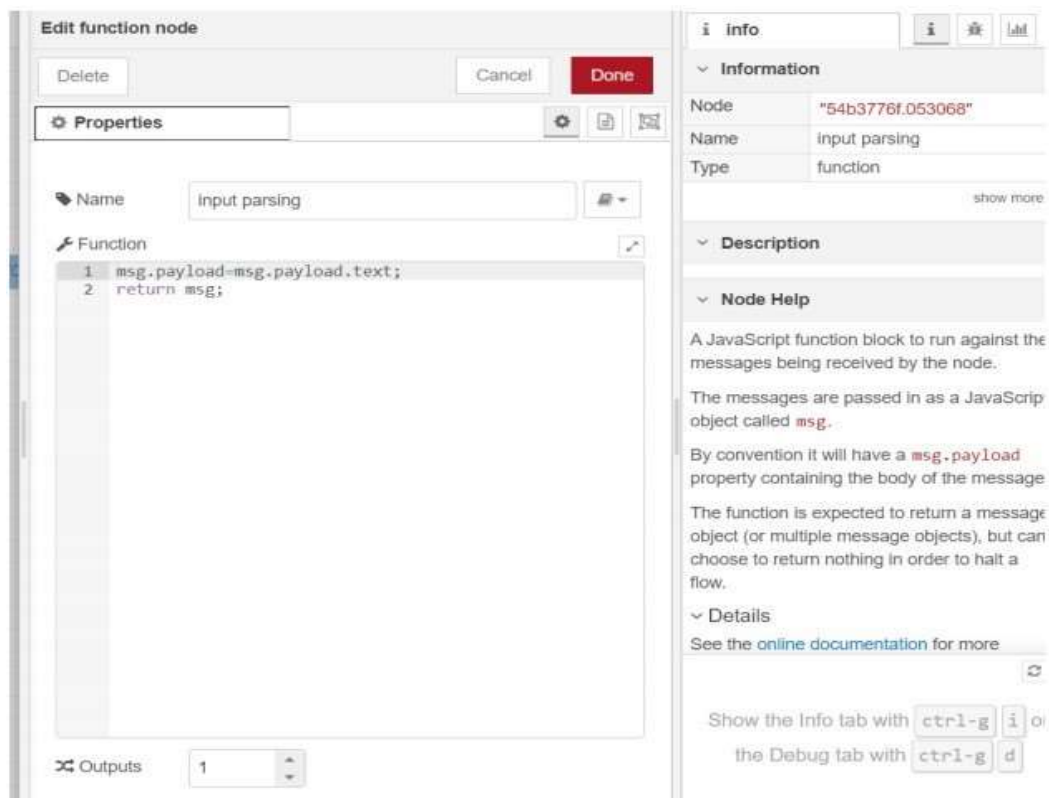
- Double-click on the Watson assistant node
- Give a name to your node and enter the username, password and workspace id of your Watson assistant service
- After entering all the information click on Done
- Drag inject node on to the flow from the Input section
- Drag Debug on to the flow from the output section
- Double-click on the inject node
- Select the payload as a string
- Enter a sample input to be sent to the assistant service and click on done
- Connect the nodes as shown below and click on Deploy



- Open Debug window as shown below
- Click on the button to send input text to the assistant node
- Observe the output from the assistant service node
- The Bot output is located inside "output.text"
- Drag the function node to parse the JSON data and get the bot response
- Double click on the function node and enter the JSON parsing code as shown below and click on done and Connect the nodes as shown below and click on Deploy.
- Re-inject the flow and observe the parsed output
- For creating a web application
 UI we need "dashboard" nodes which should be installed manually.
- Go to navigation pane and click on manage palette



- Click on install
- Search for “node-red-dashboard” and click on install and again click on install on the prompt
- The following message indicates dashboard nodes are installed, close the manage palette
- Search for “Form” node and drag on to the flow
- Double click on the “form” node to configure
- Click on the edit button to add the “Group” name and “Tab” name
- Click on the edit button to add tab name to web application
- Give sample tab name and click on add do the same thing for the group
- Give the label as “Enter your input”, Name as “text” and click on Done
- Drag a function node, double-click on it and enter the input parsing code as shown below



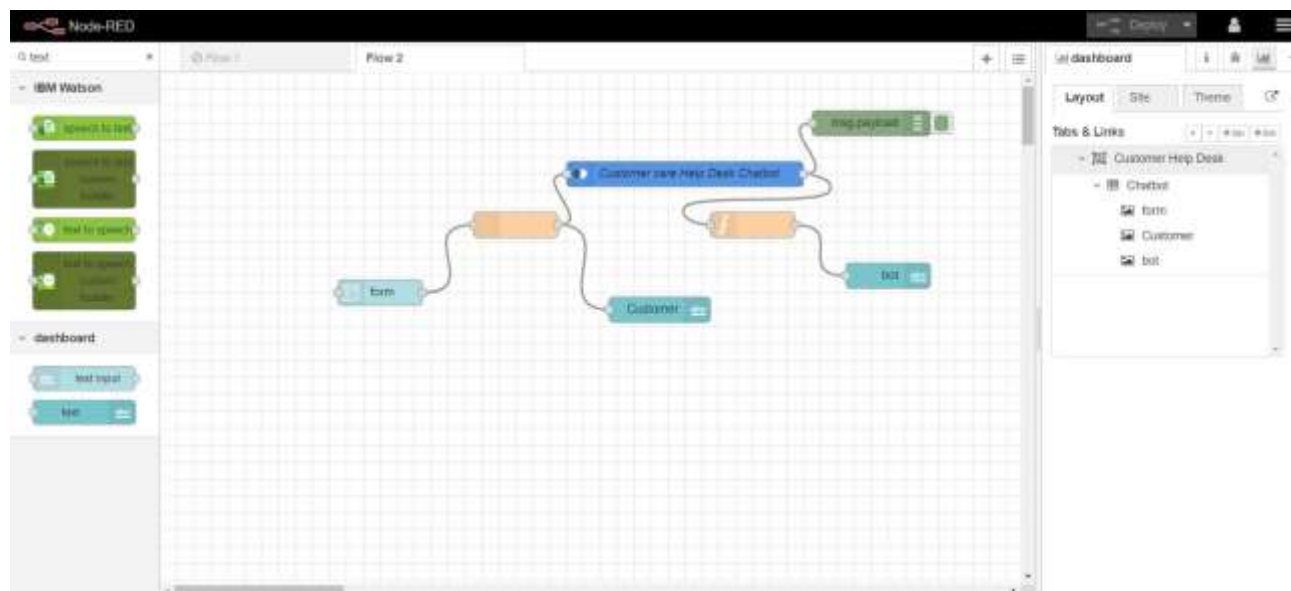
- Click on done
- Connect the form output to the input of the function node and output of the function to input of assistant node
- Search for “text” node from the “dashboard” section
- Drag two “text” nodes on to the flow
- Double click on the first text node, change the label as “You” and click on Done
- Double click on the second text node, change the label as “Bot” and click on Done
- Connect the output of “input parsing” function node to “ You” text node and output of “Parsing” function node to the input of “Bot” text node
- Click on Deploy

FLOWCHART

At first go to manage pallette and install dashboard.

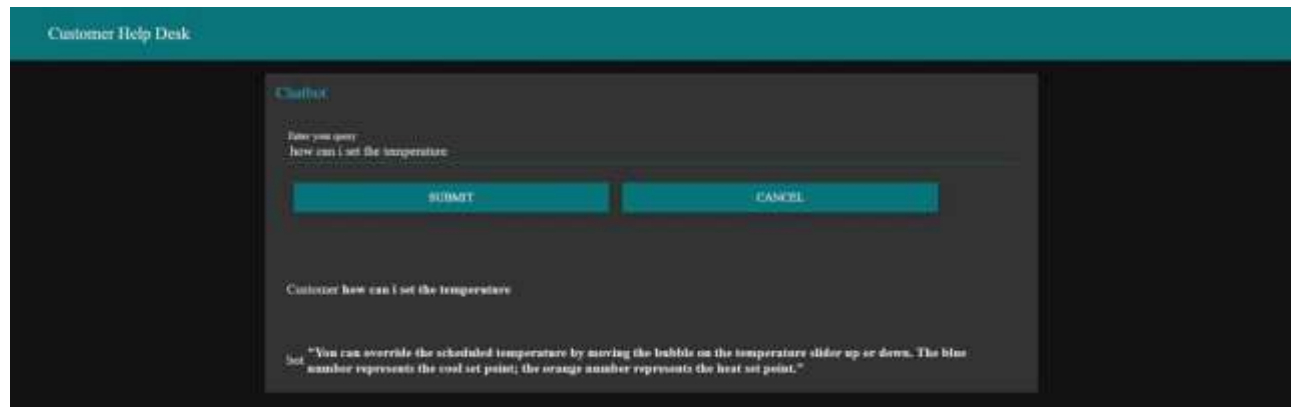
Now, Create the flow with the help of following node:

- ✓ Inject
- ✓ Assistant
- ✓ Debug
- ✓ Function
- ✓ Ui_Form
- ✓ Ui_Text



RESULTS

Finally our Node-RED dash board integrates all the components and displayed in the Dashboard UI by typing URL https://node-red-opton.eu-gb.mybluemix.net/ui/#!/0?socketid=HnjM-B_IBS_zlwGkAAA0 in browser



ADVANTAGES & DISADVANTAGES

Advantages:

- ✓ Companies can deploy chatbots to rectify simple and general human queries .
- ✓ Reduces man power
- ✓ Cost efficient
- ✓ No need to divert calls to customer agent and customer agent can look on other works.

Disadvantages:

- ✓ Some times chatbot can mislead customers
- ✓ Giving same answer for different sentiments.
- ✓ Some times cannot connect to customer sentiments and intentions.

APPLICATIONS

- ✓ It can deploy in popular social media applications like facebook,slack,telegram.
- ✓ Chatbot can deploy any website to clarify basic doubts of viewers.

CONCLUSION

By doing the above procedure and all we successfully created Intelligent helpdesk smart chatbot using Watson assistant, Watson discovery, Node-RED and cloud-functions.

FUTURE SCOPE

We can include watson studio text to speech and speech to text services to access the chatbot handsfree. This is one of the future scope of this project.

BIBLIOGRAPHY

APPENDIX

Source Code

1.Cloud Function(Node.js)

```
/**  
  
 *  
  
 * @param {object} params  
  
 * @param {string} params.iam_apikey  
  
 * @param {string} params.url  
  
 * @param {string} params.username  
  
 * @param {string} params.password  
  
 * @param {string} params.environment_id  
  
 * @param {string} params.collection_id  
  
 * @param {string} params.configuration_id  
  
 * @param {string} params.input  
  
 *
```

```
* @return {object}
```

```
*
```

```
*/
```

```
const assert = require('assert');
```

```
const DiscoveryV1 = require('watson-developer-cloud/discovery/v1');
```

```
/**
```

```
*
```

```
* main() will be run when you invoke this action
```

```
*
```

```
* @param Cloud Functions actions accept a single parameter, which must be a JSON object.
```

```
*
```

```
* @return The output of this action, which must be a JSON object.
```

```
*
```

```
*/
```

```
function main(params) {
```

```
  return new Promise(function (resolve, reject) {
```

```
    let discovery;
```

```
if (params.iam_apikey){

    discovery = new DiscoveryV1({

        'iam_apikey': params.iam_apikey,

        'url': params.url,

        'version': '2020-05-09'

    });

}

else {

    discovery = new DiscoveryV1({

        'username': params.username,

        'password': params.password,

        'url': params.url,

        'version': '2020-05-11'

    });

}


discovery.query({

    'environment_id': params.environment_id,

    'collection_id': params.collection_id,
```

```

    'natural_language_query': params.input,

    'passages': true,

    'count': 3,

    'passages_count': 3

  }, function(err, data) {

    if (err) {

      return reject(err);

    }

    return resolve(data);

  });

});

}

```

2. Node Red (flow.json)

```

[
  {
    "id": "7253a121.16642",
    "type": "tab",
    "label": "Flow 1",
    "disabled": false,
    "info": ""
  },
  {
    "id": "b1b11140.4e4ef",
    "type": "inject",
    "z": "7253a121.16642",
    "name": "",
    "topic": "",
    "payload": "Hello Node-RED!",
    "payloadType": "str",
    "repeat": "",
    "crontab": "",
    "once": false,

```

```
"onceDelay": "",
"x": 141,
"y": 61,
"wires": [
  [
    "2371449b.4bf2cc"
  ]
]
},
{
  "id": "f2f2649a.0d0d98",
  "type": "debug",
  "z": "7253a121.16642",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "payload",
  "targetType": "msg",
  "x": 670,
  "y": 140,
  "wires": []
},
{
  "id": "e150cc25.d7aa",
  "type": "function",
  "z": "7253a121.16642",
  "name": "input parsing",
  "func": "msg.payload=msg.payload.text;\nreturn msg;",
  "outputs": 1,
  "noerr": 0,
  "x": 270,
  "y": 240,
  "wires": [
    [
      "2371449b.4bf2cc",
      "49234419.64d55c"
    ]
  ]
},
{
  "id": "d3215cca.6b835",
  "type": "ui_form",
  "z": "7253a121.16642",
  "name": "",
  "label": "",
  "group": "a790cfa9.6f041",
```

```
"order": 1,
"width": "0",
"height": "0",
"options": [
  {
    "label": "Enter your query",
    "value": "text",
    "type": "text",
    "required": true,
    "rows": null
  }
],
"formValue": {
  "text": ""
},
"payload": "",
"submit": "submit",
"cancel": "cancel",
"topic": "",
"x": 107.5,
"y": 302,
"wires": [
  [
    "e150cc25.d7aa"
  ]
]
},
{
  "id": "49234419.64d55c",
  "type": "ui_text",
  "z": "7253a121.16642",
  "group": "a790cfa9.6f041",
  "order": 2,
  "width": "0",
  "height": "0",
  "name": "",
  "label": "You",
  "format": "{{msg.payload}}",
  "layout": "col-center",
  "x": 453.5,
  "y": 306,
  "wires": []
},
{
  "id": "d2a61bef.00f978",
  "type": "ui_text",
  "z": "7253a121.16642",
  "group": "d2f10b81.4883c8",
```

```

    "order": 1,
    "width": "6",
    "height": "4",
    "name": "",
    "label": "Bot",
    "format": "{{msg.payload}}",
    "layout": "col-center",
    "x": 688.5,
    "y": 237,
    "wires": []
  },
  {
    "id": "29aac8ee.b98c68",
    "type": "function",
    "z": "7253a121.16642",
    "name": "parsing",
    "func": "msg.payload = msg.payload.output.generic[0].text;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 470,
    "y": 160,
    "wires": [
      [
        "f2f2649a.0d0d98",
        "d2a61bef.00f978"
      ]
    ]
  },
  {
    "id": "2371449b.4bf2cc",
    "type": "watson-assistant-v2",
    "z": "7253a121.16642",
    "name": "My first assistant",
    "service-endpoint": "https://api.eu-gb.assistant.watson.cloud.ibm.com/instances/ae44476e-77ab-4578-9d11-23464ea66634",
    "assistant_id": "d3fa58b9-7946-4784-b120-5e025c58c9ba",
    "debug": false,
    "restart": false,
    "return_context": true,
    "alternate_intents": false,
    "multisession": true,
    "timeout": "",
    "optout-learning": false,
    "x": 391.5,
    "y": 100,
    "wires": [
      [
        "29aac8ee.b98c68",

```



```
        "caf5ac15.c978d"
    ]
]
},
{
    "id": "caf5ac15.c978d",
    "type": "debug",
    "z": "7253a121.16642",
    "name": "",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "x": 557.5,
    "y": 61,
    "wires": []
},
{
    "id": "a790cfa9.6f041",
    "type": "ui_group",
    "z": "",
    "name": "",
    "tab": "27e055be.28bb7a",
    "order": 1,
    "disp": true,
    "width": "6",
    "collapse": false
},
{
    "id": "d2f10b81.4883c8",
    "type": "ui_group",
    "z": "",
    "name": "",
    "tab": "27e055be.28bb7a",
    "order": 3,
    "disp": true,
    "width": "6",
    "collapse": false
},
{
    "id": "27e055be.28bb7a",
    "type": "ui_tab",
    "z": "",
    "name": "Product",
    "icon": "dashboard",
    "order": 2,
```

```
"disabled": false,  
"hidden": false  
}  
]
```

Reference:

1. https://www.ibm.com/cloud/architecture/tutorials/cognitive_discovery
2. <https://cloud.ibm.com/docs/assistant?topic=assistant-getting-started>
3. <https://developer.ibm.com/recipes/tutorials/how-to-create-a-watson-chatbot-on-nodered/>
4. <http://www.iotgyan.com/learning-resource/integration-of-watson-assistant-to-node-red>
5. <https://github.com/IBM/watson-discovery-sdu-with-assistant>
6. <https://youtu.be/K6P7hSDCCdY>