

Project Report

Name : *Dhairysheel Deshmukh*
(dhairyasheel06@gmail.com)

Title : *Intelligent Customer Help Desk With Smart Document Understanding*

Category: *Artificial Intelligence Internship at*
[smartinternz.com@2020](https://smartinternz.com/2020)

1. INTRODUCTION

- a. Overview
- b. Purpose

2. LITERATURE SURVEY

- a. Existing Problem
- b. Proposed Solution

3. THEORETICAL ANALYSIS

- a. Block Diagram
- b. Hardware / Software Designing

4. EXPERIMENTAL INVESTIGATIONS

5. FLOWCHART

6. RESULT

7. ADVANTAGES & DISADVANTAGES

8. APPLICATIONS

9. CONCLUSION

10. FUTURE SCOPE

11. BIBLIOGRAPHY APPENDIX

- a. Source Code
- b. Reference

INTRODUCTION

Overview:

We will be able to write an application that leverages multiple WatsonAI 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, IBMWatson
- ⇒ **Functional Requirements:** IBMcloud
- ⇒ **Technical Requirements:** AI,ML,WATSONAI,PYTHON
- ⇒ **Software Requirements:** Watson assistant, Watsondiscovery.
- ⇒ **Project Deliverables:** SmartinternzInternship
- ⇒ **Project Team:** Dhairysheel Deshmukh
- ⇒ **Project Duration:**19days

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 realperson.

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 customer's 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.

Scope of Work

- ★ Create a customer care dialog skill in WatsonAssistant
- ★ 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 WatsonDiscovery
- ★ Build a web application with integration to all these services & deploy the same on IBM CloudPlatform

LITERATURE SURVEY

🔴 Existing Problem:

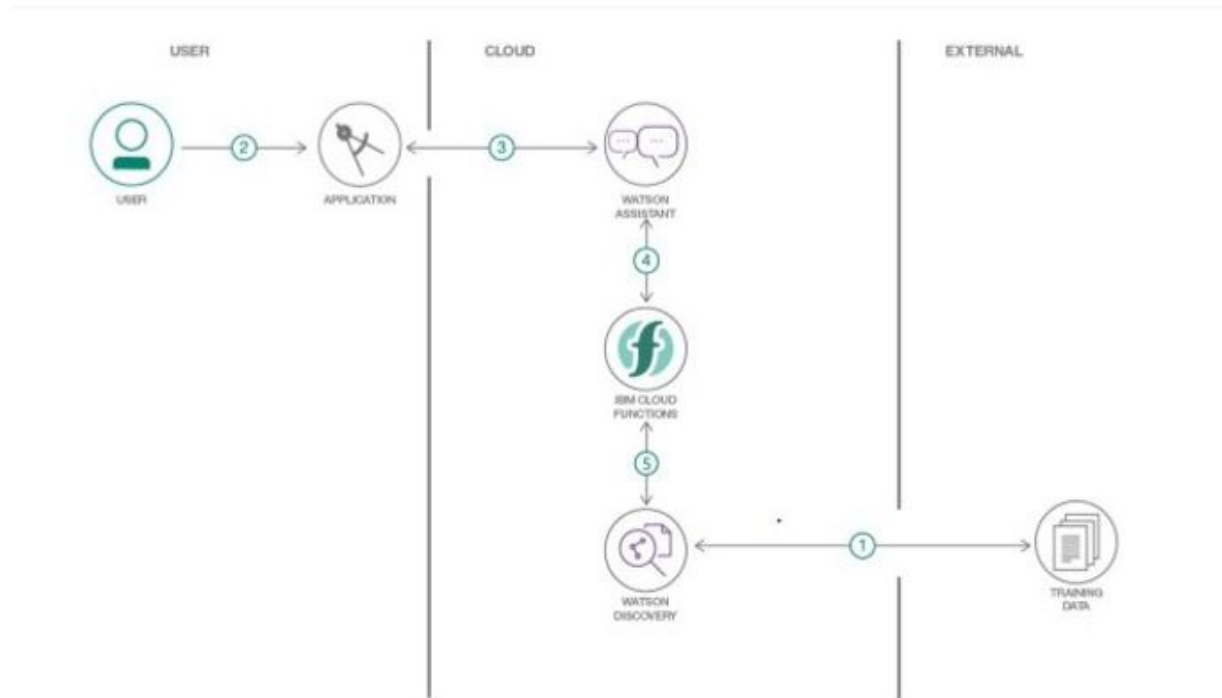
Generally Chatbots means getting input from users and getting only response questions and for some questions the output from bot will be like “try again”, “I don’t understand”, “will you repeat again”, and so on... and directs customer to customer agent but a good customer Chatbot should minimize involvement 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.

🔴 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

🎯 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.

🎯 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

Create IBM Cloudservices

Create the following services:

- ★ Watson Discovery
- ★ Watson Assistant
- ★ Node Red

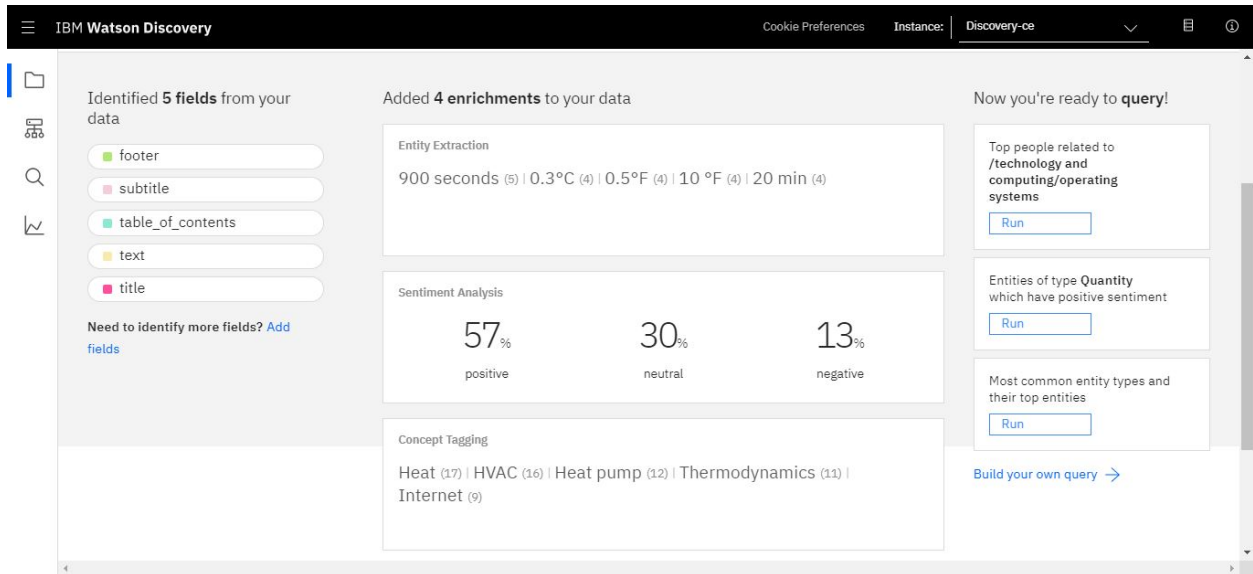
Configure WatsonDiscovery

Import the document

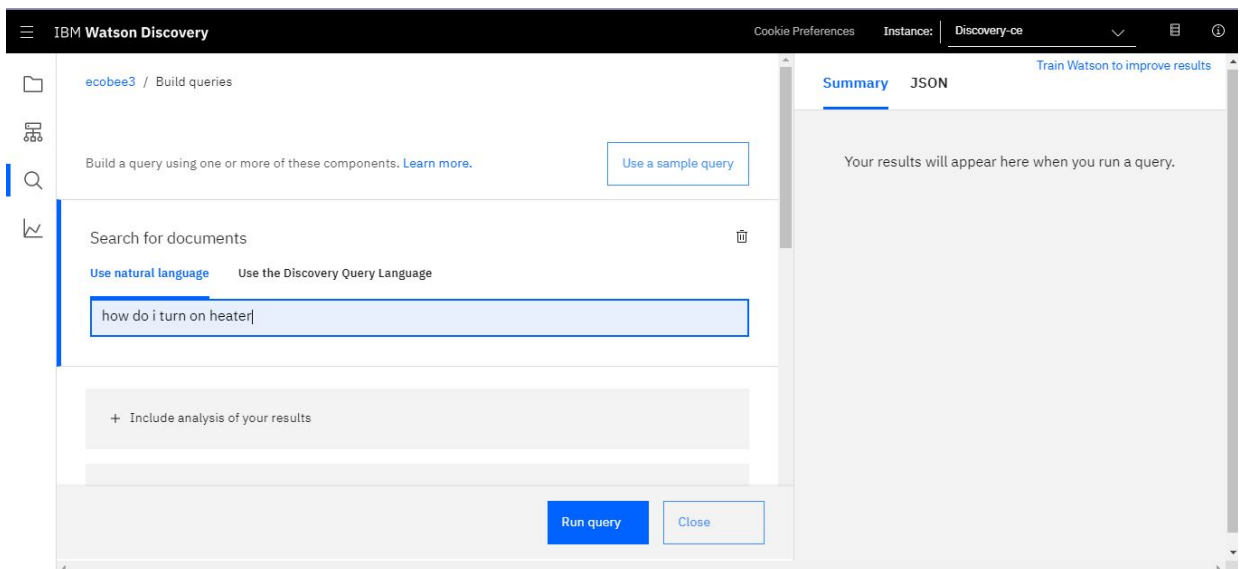
Launch the Watson Discovery tool and create a new data collection by selecting the Upload your own data option. Give the data collection a unique name. When prompted, select and upload the ecobee3_UserGuide.pdf file located in the data directory of your local repo.

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 button.

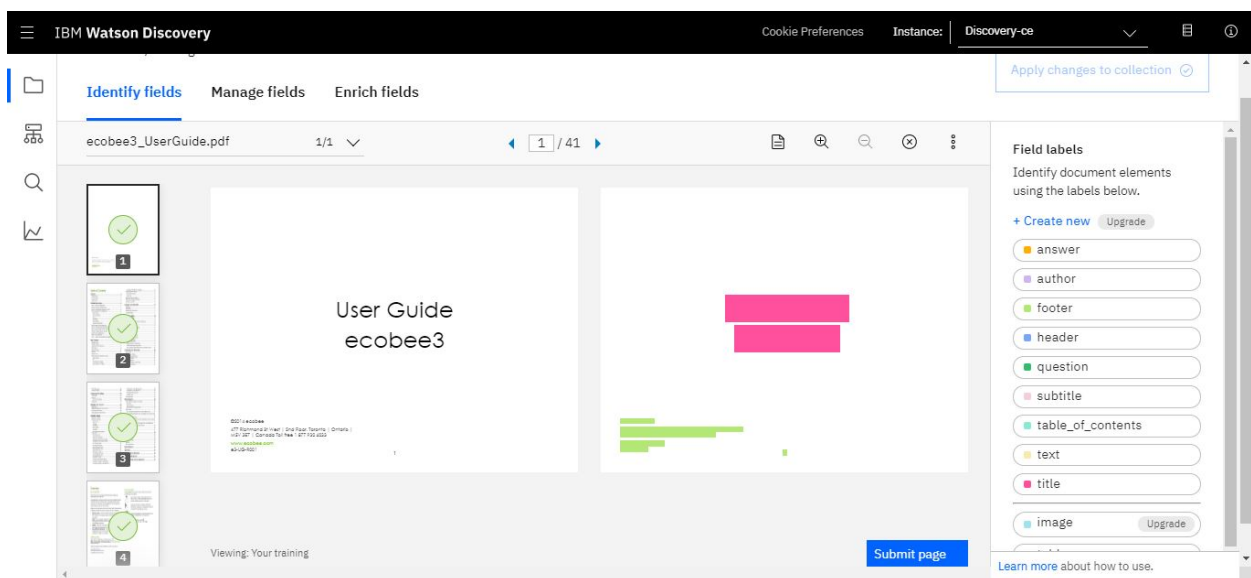


Enter queries related to the operation of the thermostat and view the results. As you will see, the results are not very useful, and in some cases, not even related to the question.

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.

- ★ There is the list of pages in the manual. As each is processed, a green check mark will appear on the page.
- ★ On left hand side is the current page being annotated.

- ★ Right hand side of previous page is where you select text and assign it a label.
- ★ On extreme right hand side there is the list of labels you can assign to the page text.
- ★ Click Submit Page button to submit the page to Discovery.
- ★ Click Apply changes to collection when you have completed the annotation process.

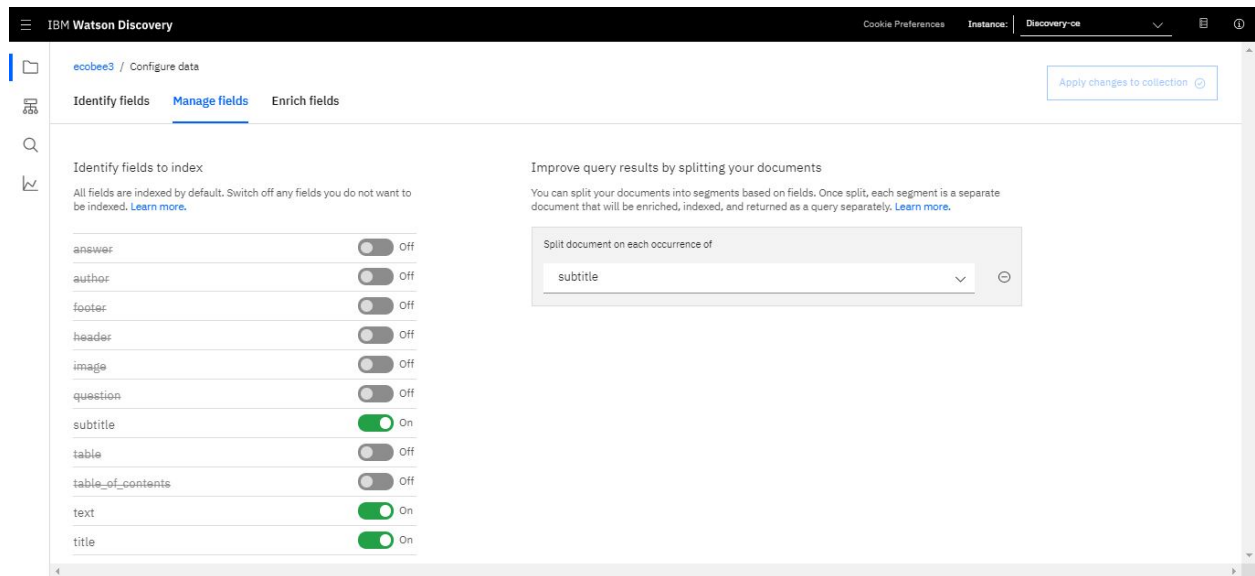
As you go through the annotations one page at a time, Discovery is learning and should start automatically updating the upcoming pages. Once you get to a page that is already correctly annotated, you can stop, or simply click Submit Page to acknowledge it is correct. The more pages you annotate, the 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, you will be asked to reload the document. Choose the same owner's manual .pdf document as before.

Next, click on the Manage fields tab.



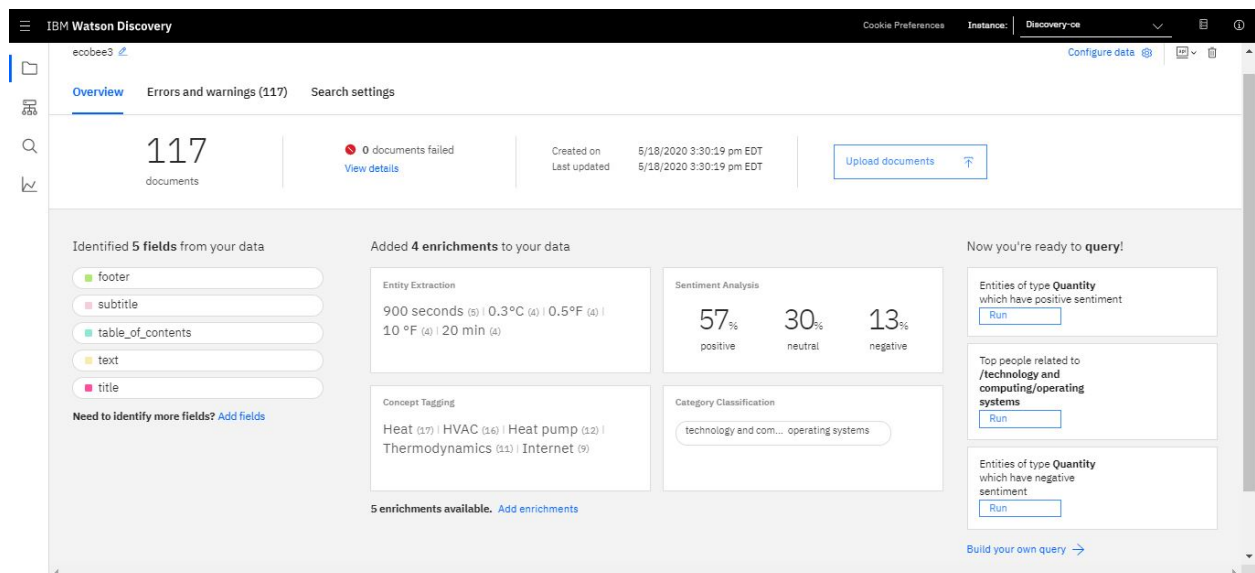
The table below is where you tell Discovery which fields to ignore. Using the on/off buttons, turn off all labels except subtitles and text.

The side bar is telling Discovery to split the document apart, based on subtitle.

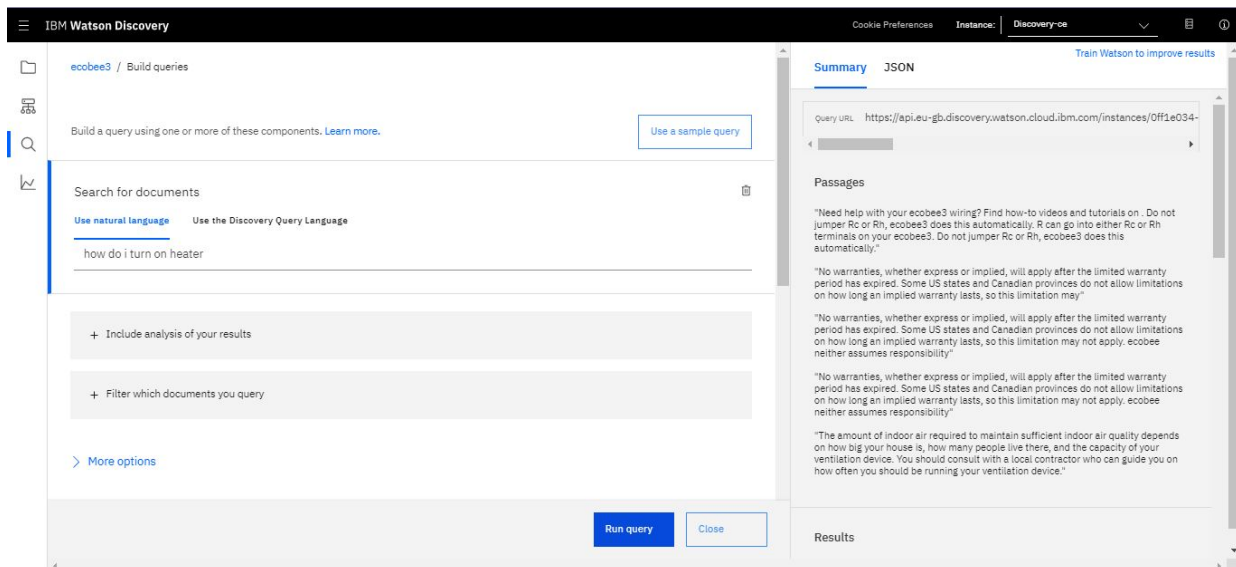
Click Apply changes to collection to submit your changes.

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:



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



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 at side of dustbin symbol located at the top right side of your collection panel:

Cookie Preferences

Instance:

Discovery-ce

Configure data

api

Collection ID

61dd4b53-7501-48d5-b1d9-e4cd2a034b87

Configuration ID

00edb0e9-feab-49d8-971f-126b962d73da

Environment ID

09cbd2bc-e072-4412-90b4-472cc31538a5

it Analysis

7%

30%

13%

tive

neutral

negative

Classification

ology and com... operating systems

Top people related to /technology and computing/operating systems

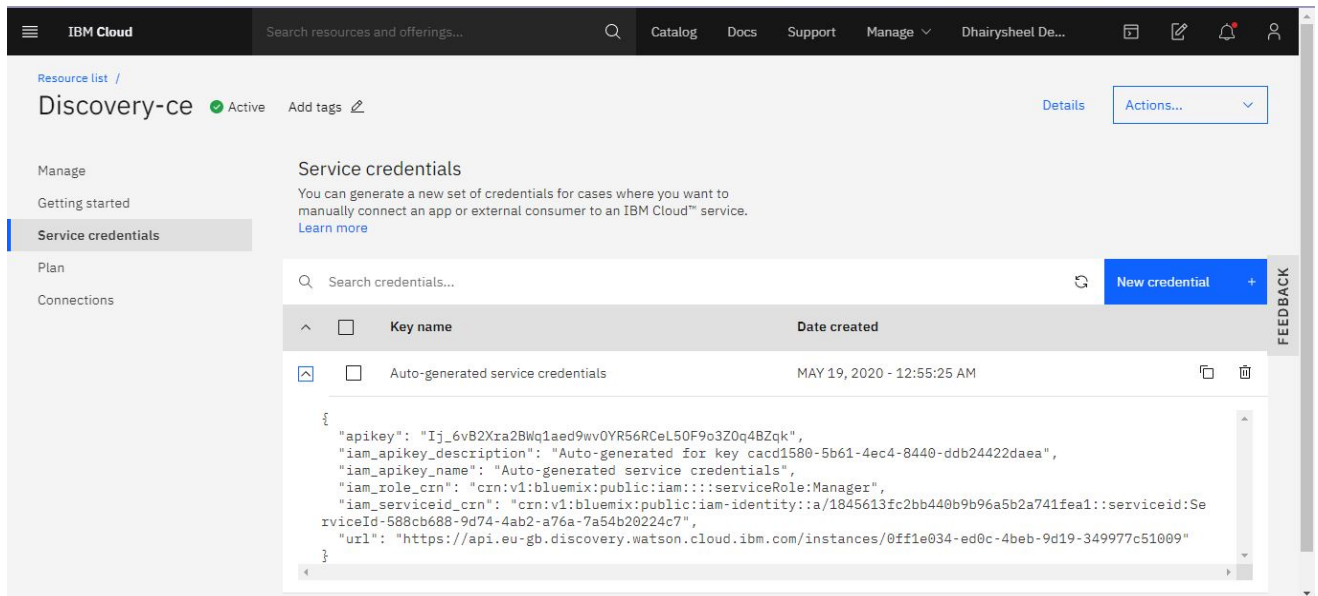
Run

Entities of type Quantity which have negative sentiment

Run

Entities of type Quantity

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

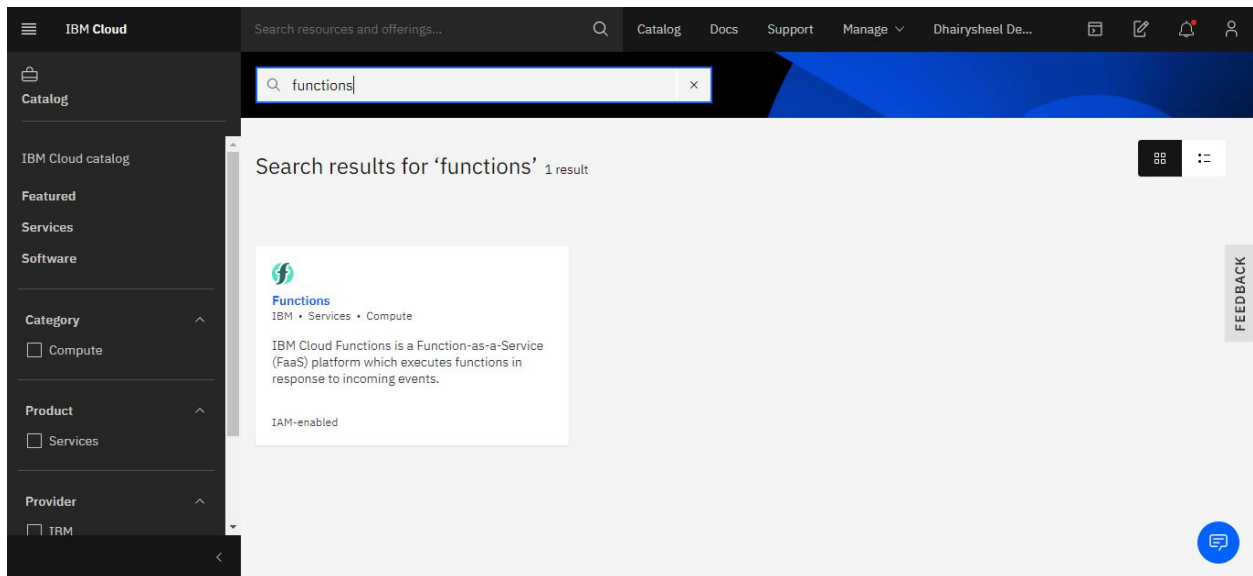


Click the drop-down menu to view the IAM apikey and URL endpoint for your service.

Create IBM Cloud Functions Action

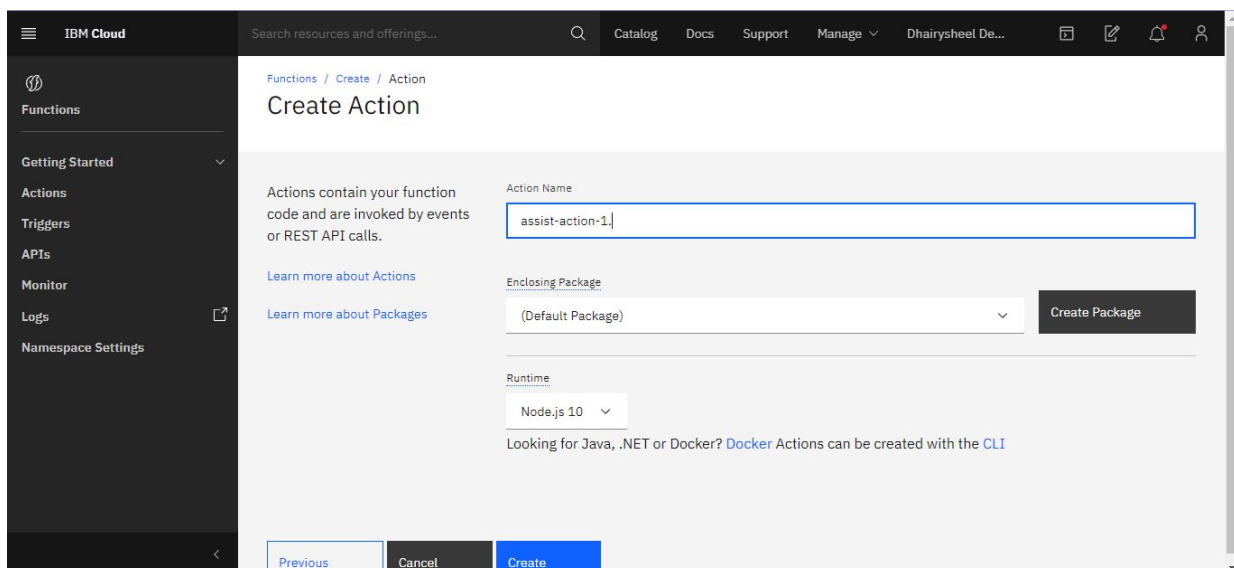
Now let's create the web action that will make queries against our Discovery collection.

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

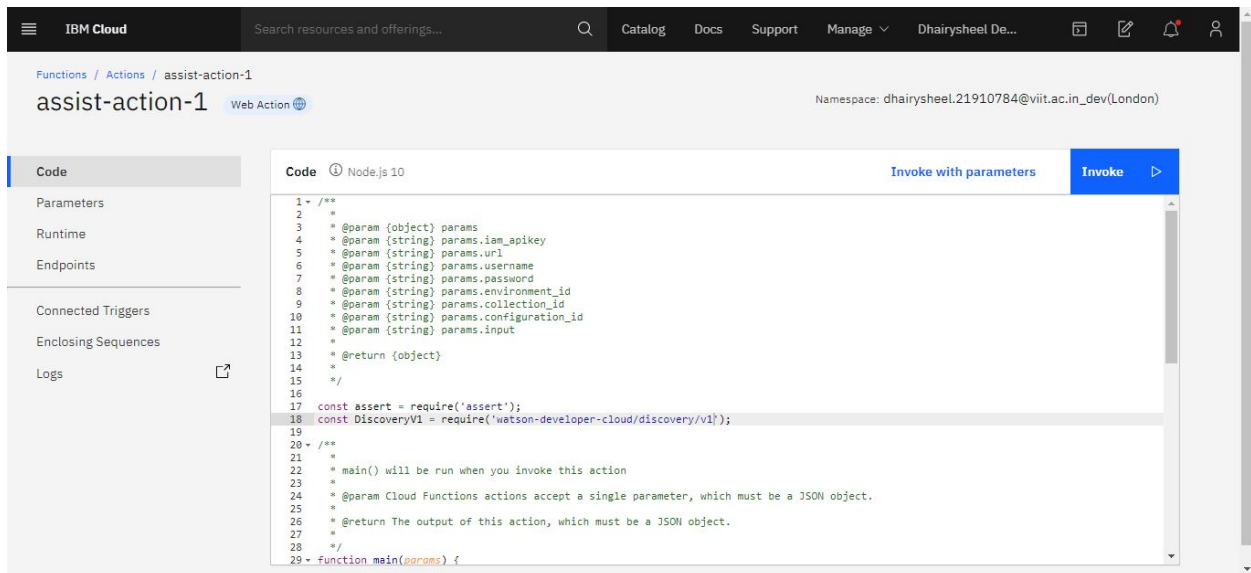


From the Functions main panel, click on the Actions tab. Then click on Create. From the Create panel, select the Create Action option.

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



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



In the code editor window , 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 , it will fail due to credentials not being defined yet. We'll do this next.

Select the Parameters tab :

The screenshot shows the IBM Cloud Functions console. The top navigation bar includes the IBM Cloud logo, a search bar, and links to Catalog, Docs, Support, and Manage. The main header shows the breadcrumb 'Functions / Actions / assist-action-1' and the namespace 'dhairysheel.21910784@viit.ac.in_dev(London)'. On the left, a sidebar contains links to Code, Parameters (selected), Runtime, Endpoints, Connected Triggers, Enclosing Sequences, and Logs. The main content area displays the 'Parameters' tab for the action 'assist-action-1'. It features a table with two columns: 'Parameter Name' and 'Parameter Value'. The table contains four entries: 'url' with a long URL, 'environment_id' with a UUID, 'collection_id' with another UUID, and 'iam_apikey' with a long alphanumeric string. An 'Add Parameter' link is located in the top right corner of the parameters section.

Parameter Name	Parameter Value
url	"https://api.eu-gb.discovery.watson.cloud.ibm.com/instances/0ff1e03"
environment_id	"09cbd2bc-e072-4412-90b4-472cc31538a5"
collection_id	"61dd4b53-7501-48d5-b1d9-e4cd2a034b87"
iam_apikey	"Ij_6vB2Xra2BWq1aed9vvOYR56RCeL5OF9o3ZOq4BZqk"

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... Catalog Docs Support Manage Dhairysheel De...

Functions / Actions / assist-action-1

assist-action-1 Web Action Namespace: dhairysheel.21910784@viit.ac.in_dev(London)

Code Node.js 10 Invoke with parameters Invoke

```
1 /**
2  *
3  * @param {object} params
4  * @param {string} params.iam_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 const assert = require('assert');
17 const DiscoveryV1 = require('watson-developer-cloud/discovery/v1');
18
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
25  *
26  * @return The output of this action, which must be a JSON object.
27  *
28  */
29 function main(params) {
```

Activations Collapse Clear

assist-action-1 849 ms 5/21/2020, 15:56:14

Activation ID: 6639e00b9f34aectb98e00b9f34aectf

Results:

```
{
  "matching_results": 117,
  "passages": [],
  "results": [
    {
      "enriched_text": {
        "categories": {
          "label": "/technology and computing/operating systems",
          "score": 0.929149
        },
        "label": "/technology and computing/hardware",
        "score": 0.878476
      },
      "label": "/technology and computing/hardware/computer components",
      "score": 0.826041
    }
  ]
}
```

Next, go to the Endpoints panel :

IBM Cloud Search resources and offerings... Catalog Docs Support Manage Dhairysheel De...

assist-action-1 Web Action Namespace: dhairysheel.21910784@viit.ac.in_dev(London)

Code Parameters Runtime Endpoints Connected Triggers Enclosing Sequences Logs

Web Action

☒ Enable as Web Action Allow your Cloud Functions actions to handle HTTP events. Web Actions allow to control the response data and type by using a set of URL extensions, such as .json or .html. Learn more about [Web Actions](#).
Note: The Web Action URL below requires to return a dict object that contains a body property.

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

HTTP Method	Auth	URL
ANY	Public	https://eu-gb.functions.cloud.ibm.com/api/v1/web/dhairysheel.21910784%40viit.ac.in_dev/default/assist-action-1

REST API

HTTP Method	Auth	URL
-------------	------	-----

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

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

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

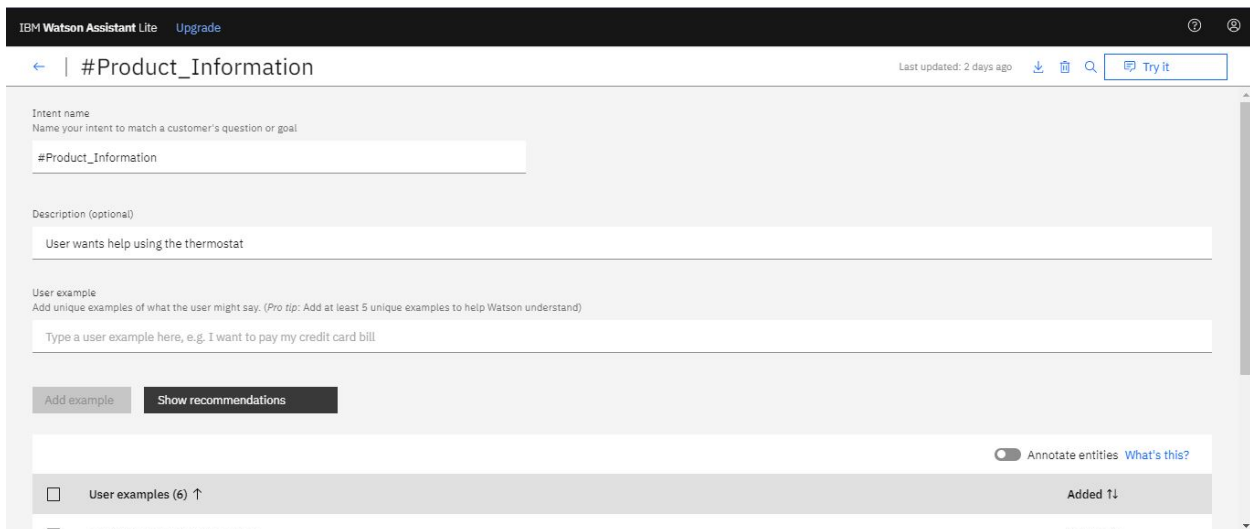
The default customer care dialog does not have a way to deal with any questions involving outside resources, so we will need to add this.

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.

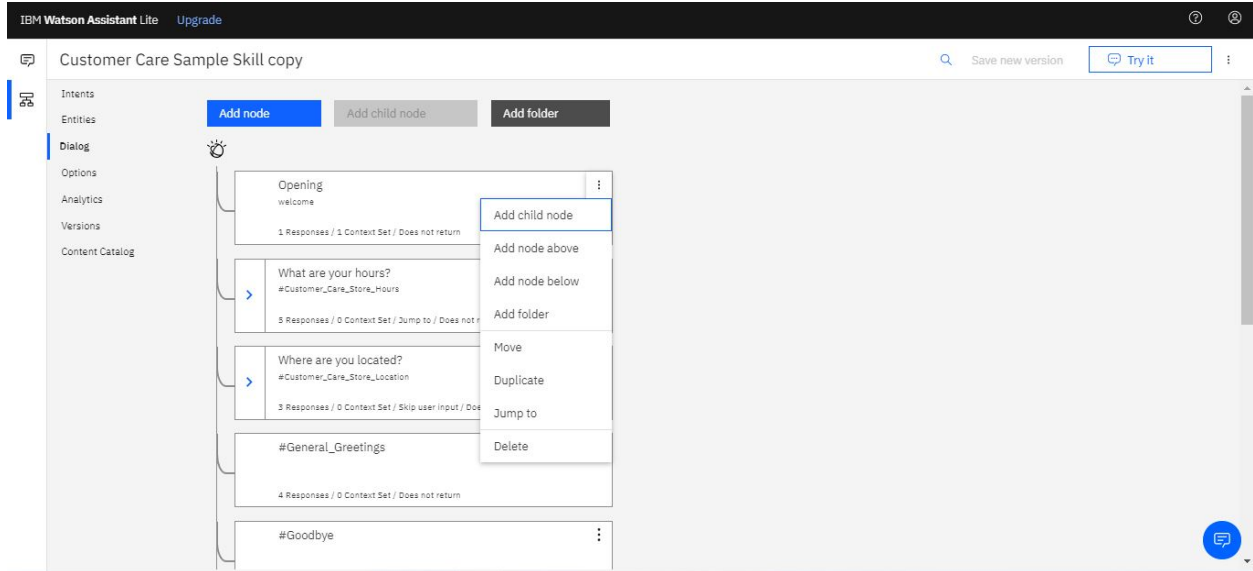


The screenshot shows the IBM Watson Assistant Lite interface. At the top, there's a header with "IBM Watson Assistant Lite" and an "Upgrade" button. Below the header, the breadcrumb navigation shows "← | #Product_Information". The main content area is divided into sections for configuring the intent:

- Intent name:** A text input field containing "#Product_Information".
- Description (optional):** A text input field containing "User wants help using the thermostat".
- User example:** A text input field containing "Type a user example here, e.g. I want to pay my credit card bill".
- Buttons:** "Add example" and "Show recommendations".
- Entity Annotation:** A toggle switch labeled "Annotate entities" with a link "What's this?".
- User examples list:** A section titled "User examples (6) ↑" with a button "Added 11".

Create new dialog node

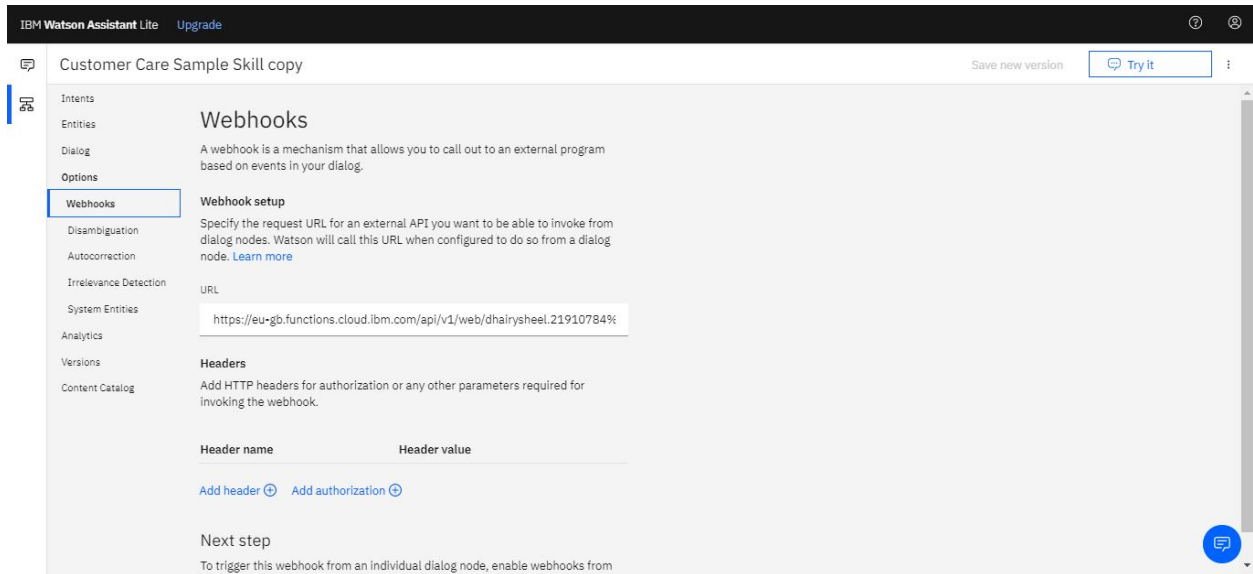
Now we need to add a node to handle our intent. Click on the Dialog tab, then click on the drop down menu for the Small Talk node, and select the Add node below [option.



Enable webhook from Assistant

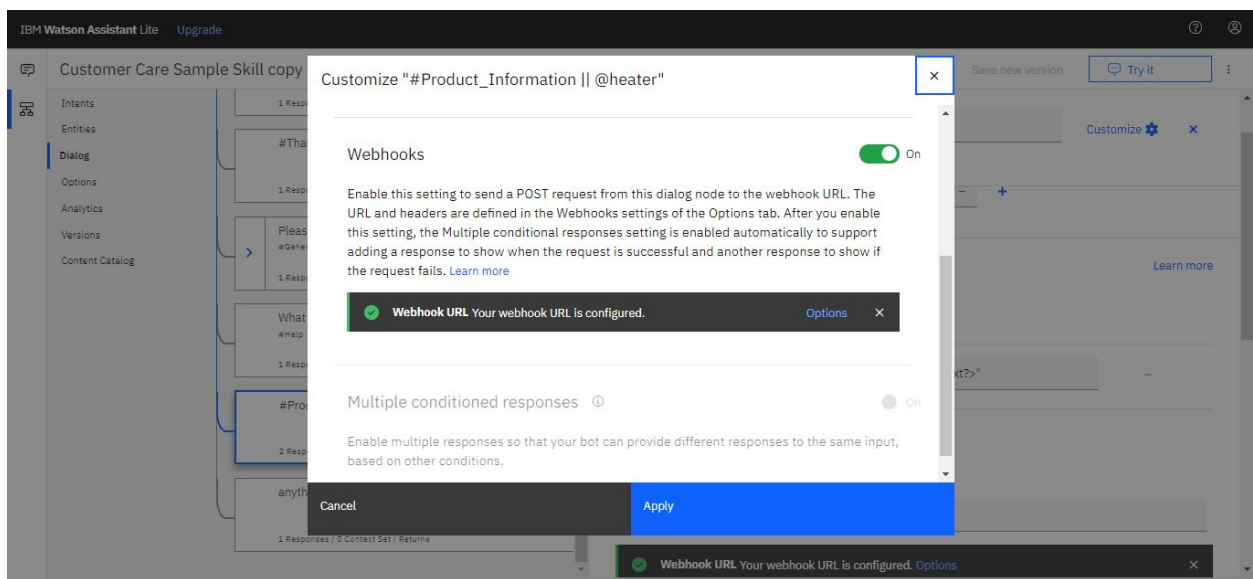
Set up access to our WebHook for the IBM Cloud Functions action you created in Step4.

Select the Options tab :



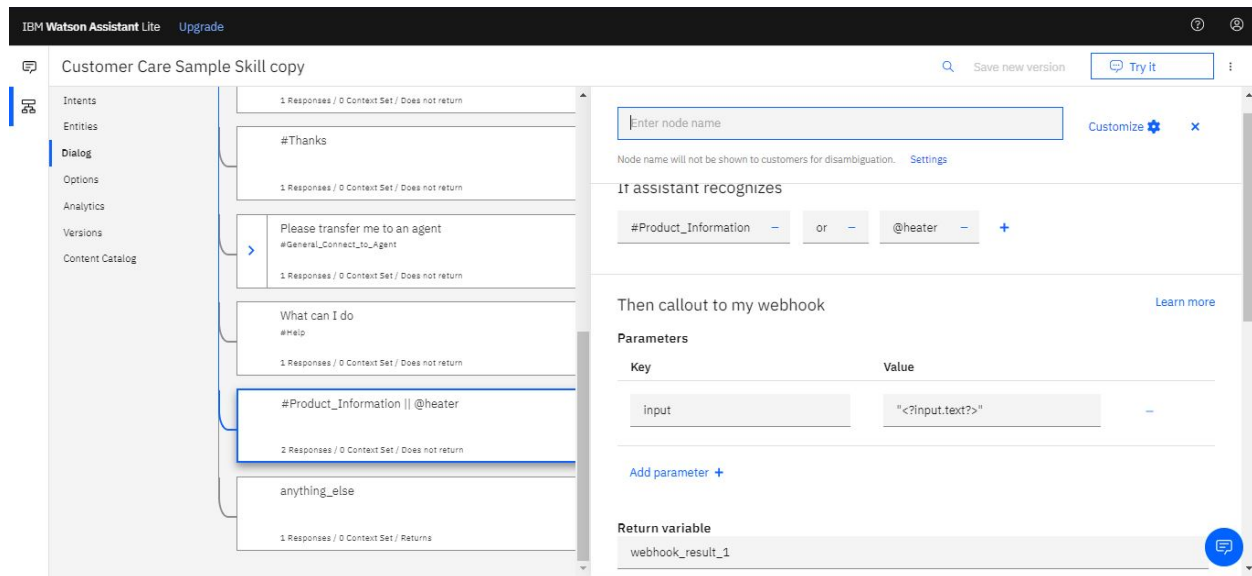
Enter the public URL endpoint for your action.

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:



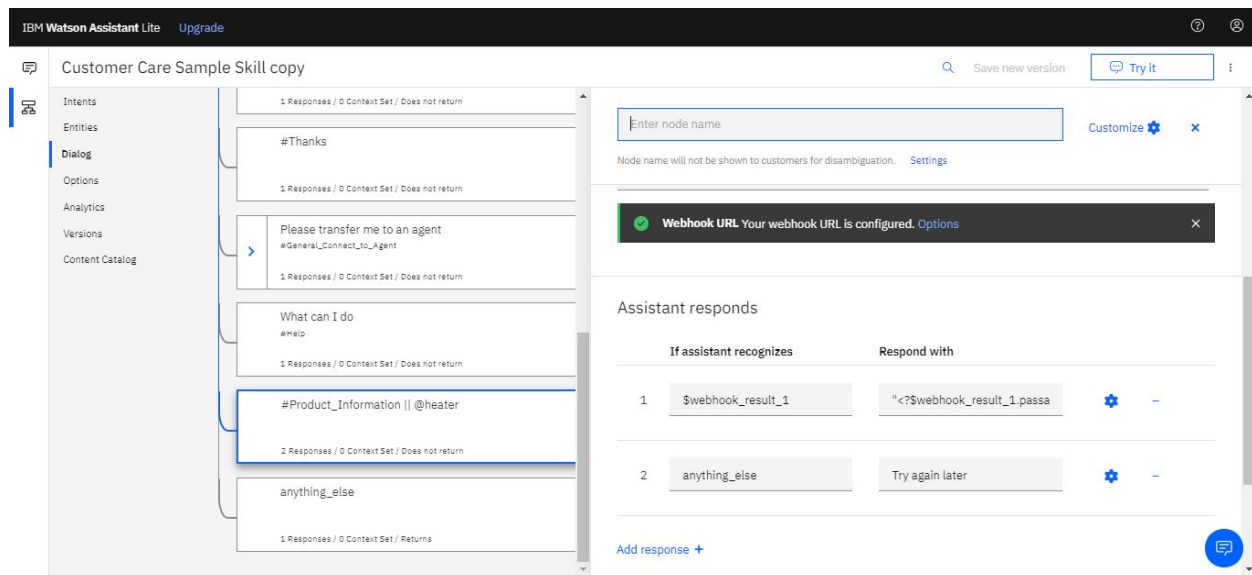
Click Apply.

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



You will also need to pass in the users question via the parameter input . 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:



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.

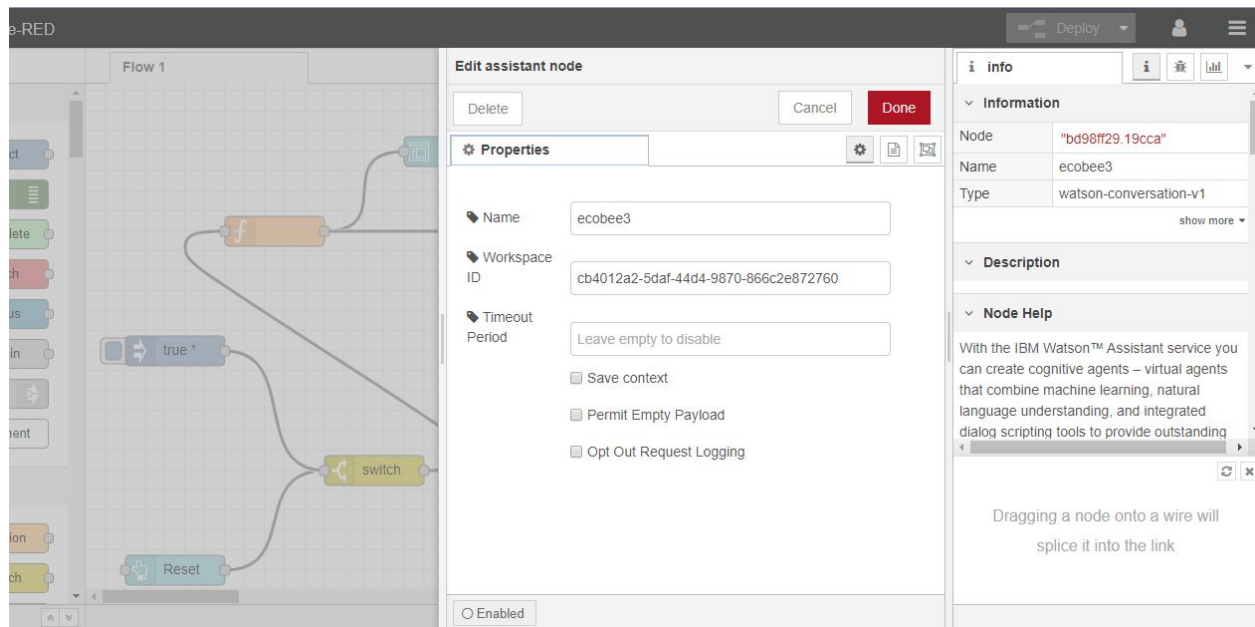
You can also verify that the call was successfully completed by clicking on the Manage Context button at the top right. The response from the Discovery query will be stored in the

\$webhook_result_1 variable:

Create Flow and Configure Node:

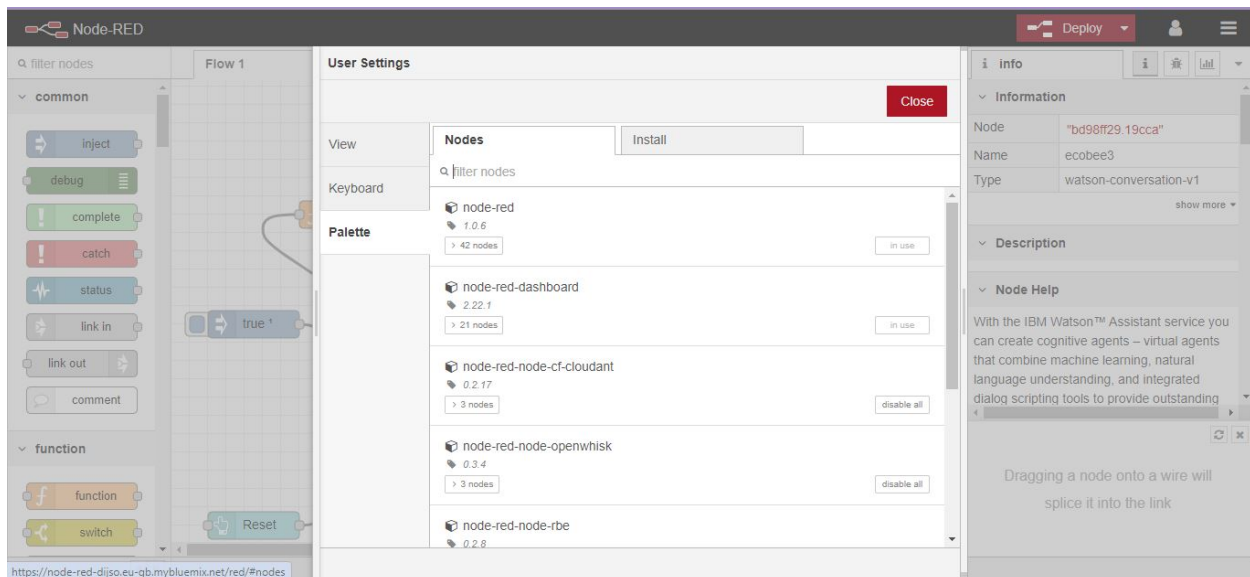
Integration of watson assistant in Node-RED

1. Double-click on the Watson assistant node
2. Give a name to your node and enter the work space id of your Watson assistant service

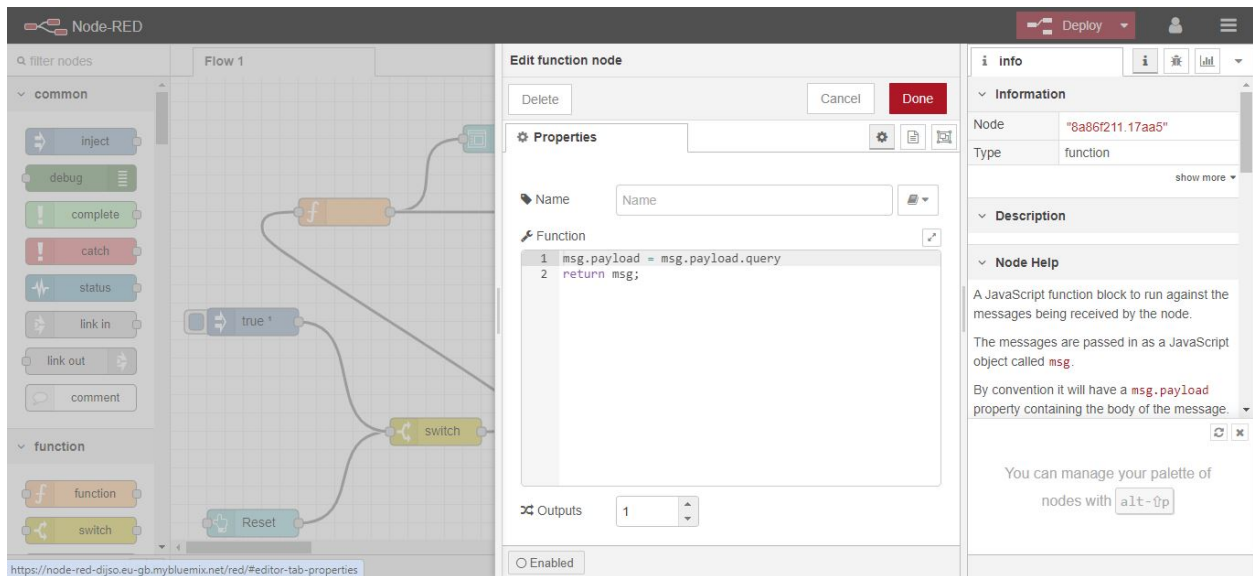


3. After entering all the information click on Done
4. Drag inject node on to the flow from the Input section
5. Drag Debug on to the flow from the output section
6. Double-click on the inject node
7. Select the payload as a string
8. Enter a sample input to be sent to the assistant service and click on done
9. Connect the nodes and click on Deploy
10. Open Debug window as shown below
11. Click on the button to send input text to the assistant node
12. Observe the output from the assistant service node

13. The Bot output is located inside "output.text"
14. Drag the function node to parse the JSON data and get the bot response
15. Double click on the function node and enter the JSON parsing code as shown below and click on done • Connect the nodes as shown below and click on Deploy
16. Re-inject the flow and observe the parsed output
17. For creating a web application
UI we need "dashboard" nodes which should be installed manually.
18. Go to navigation pane and click on manage palette



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



29. Click on done
30. Connect the form output to the input of the function node and output of the function to input of assistant node
31. Search for "text" node from the "dashboard" section
32. Drag two "text" nodes on to the flow
33. Double click on the first text node, change the label as "You" and click on Done
34. Double click on the second text node, change the label as "Bot" and click on Done
35. 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
36. Click on Deploy

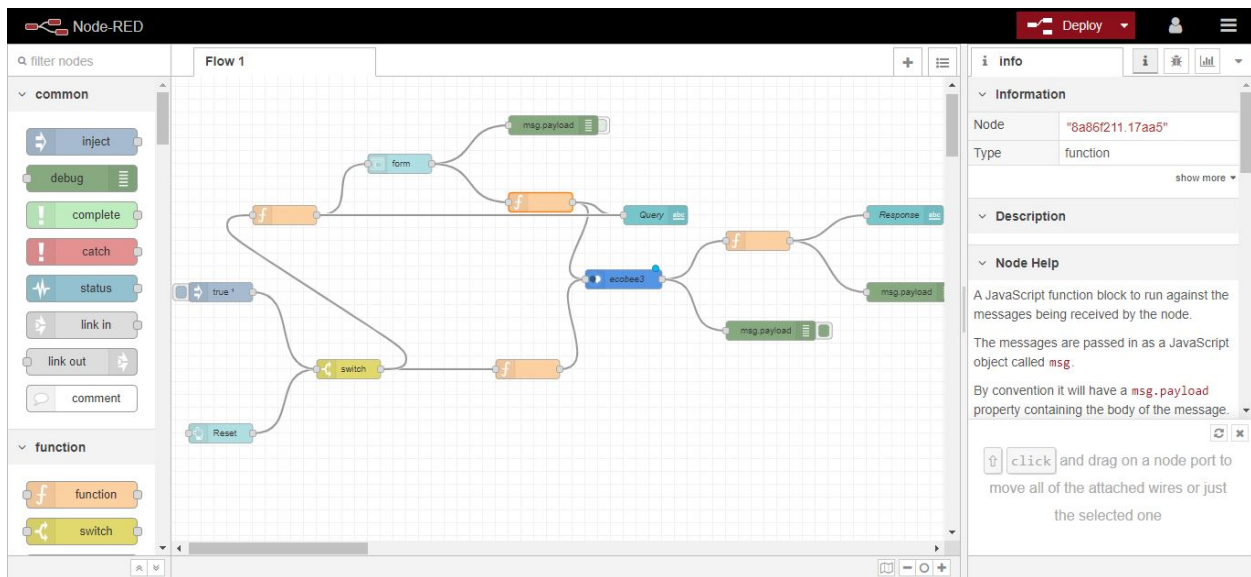
FLOWCHART

At first go to manage palette and install dashboard.

Now Create the flow with the help of following node:

1. Inject
2. Assistant
3. Debug

4. Function
5. Ui_Form
6. 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-dijso.eu-gb.mybluemix.net/ui/> in browser

The screenshot shows the 'Intelligent Customer Help Desk With Smart Document Understanding' interface. On the left, a 'Chatbox' contains a 'RESET' button, a 'User Input' field with the text 'how to start thermostat', and 'SUBMIT' and 'CANCEL' buttons. On the right, the 'Bot Response' section displays two articles. The first article, 'Smart Recovery', explains how the system learns heating and cooling patterns and provides a 4-step guide for enabling Smart Recovery. The second article, 'Going on Vacation', describes the vacation feature for conserving energy and provides a 6-step guide for setting up a vacation event.

ADVANTAGES & DISADVANTAGES

🔴 **Advantages:**

7. Companies can deploy chatbots to rectify simple and general human queries.
8. Reduces manpower
9. Cost efficient
10. No need to divert calls to customer agent and customer agent can look on other works.

🔴 **Disadvantages:**

11. Some times chatbot can mislead customers
12. Giving same answer for different sentiments.
13. Some times cannot connect to customer sentiments and intentions.

APPLICATIONS

- a. It can deploy in popular social media applications like facebook, slack, telegram.
- b. 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.

FUTURESCOPE

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.

BIBILOGRAPHY

APPENDIX

Source Code

1. CloudFunction(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': '2019-03-25'
      });
    }
    else {
      discovery = new DiscoveryV1({
        'username': params.username,
        'password': params.password,
        'url': params.url,
        'version': '2019-03-25'
      });
    }

    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": "8fc7dd91.e8301",  
    "type": "tab",  
    "label": "Flow 1",  
    "disabled": false,  
    "info": ""  
  },  
  {  
    "id": "a582cd5.431273",  
    "type": "debug",  
    "z": "8fc7dd91.e8301",  
    "name": "",  
    "active": false,  
    "tosidebar": true,  
    "console": false,  
    "tostatus": false,  
    "complete": "payload",  
    "targetType": "msg",  
    "x": 610,  
    "y": 180,  
    "wires": []  
  },  
  {  
    "id": "91009782.fd69b8",  
    "type": "ui_text",  
    "z": "8fc7dd91.e8301",  
    "group": "4e52fc0f.c72334",  
    "order": 2,  
    "width": 10,  
    "height": 2,  
    "name": "Query",  
    "label": "User Input",
```

```
"format": "{{msg.payload}}",
"layout": "col-center",
"x": 790,
"y": 240,
"wires": [],
"inputLabels": [
  "asd"
]
},
{
  "id": "6df013ed.e3da4c",
  "type": "watson-conversation-v1",
  "z": "8fc7dd91.e8301",
  "name": "ecobee3",
  "workspaceid": "cb4012a2-5daf-44d4-9870-866c2e872760",
  "multiuser": false,
  "context": false,
  "empty-payload": false,
  "service-endpoint": "",
  "timeout": "",
  "optout-learning": false,
  "x": 780,
  "y": 400,
  "wires": [
    [
      "cd283a8e.47aa38",
      "51948dd3.5e3d74"
    ]
  ]
},
{
  "id": "53a46b5f.557e34",
  "type": "ui_text",
  "z": "8fc7dd91.e8301",
  "group": "3a610ff8.ae3c5",
  "order": 1,
  "width": 15,
```

```

    "height": "20",
    "name": "Response",
    "label": "",
    "format": "{{msg.payload}}",
    "layout": "row-spread",
    "x": 1160,
    "y": 240,
    "wires": []
  },
  {
    "id": "d9867dcf.caae",
    "type": "debug",
    "z": "8fc7dd91.e8301",
    "name": "",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "x": 1170,
    "y": 360,
    "wires": []
  },
  {
    "id": "cd283a8e.47aa38",
    "type": "function",
    "z": "8fc7dd91.e8301",
    "name": "",
    "func": "msg.payload.text=\"\\\"; \\nif(msg.payload.context.webhook_result_1)\\n{ \\n
for(var i in msg.payload.context.webhook_result_1.results)\\n  { \\n
msg.payload.text=msg.payload.text+\"<br>\"+msg.payload.context.webhook_result_1.r
esults[i].subtitle;\\n
msg.payload.text=msg.payload.text+\"<br><br>\"+msg.payload.context.webhook_result
_1.results[i].text+\"<br>\\\"; \\n  } \\n  msg.payload=msg.payload.text; \\n  \\n} \\nelse
\\nmsg.payload = msg.payload.output.text[0];\\nreturn msg; ",
    "outputs": 1,

```



```
"noerr": 0,
"x": 970,
"y": 320,
"wires": [
  [
    "d9867dcf.caae",
    "53a46b5f.557e34"
  ]
]
},
{
  "id": "2845b07a.afb5",
  "type": "ui_form",
  "z": "8fc7dd91.e8301",
  "name": "",
  "label": "",
  "group": "4e52fc0f.c72334",
  "order": 4,
  "width": 10,
  "height": 2,
  "options": [
    {
      "label": "Enter your Query",
      "value": "query",
      "type": "text",
      "required": true,
      "rows": null
    }
  ],
  "formValue": {
    "query": ""
  },
  "payload": "",
  "submit": "Submit",
  "cancel": "Cancel",
  "topic": "",
  "x": 390,
```

```
"y": 320,
"wires": [
  [
    "a582cd5.431273",
    "908688ec.465ab8"
  ]
]
},
{
  "id": "908688ec.465ab8",
  "type": "function",
  "z": "8fc7dd91.e8301",
  "name": "",
  "func": "msg.payload = msg.payload.query\\nreturn msg;",
  "outputs": 1,
  "noerr": 0,
  "x": 590,
  "y": 320,
  "wires": [
    [
      "6df013ed.e3da4c",
      "91009782.fd69b8"
    ]
  ]
},
{
  "id": "51948dd3.5e3d74",
  "type": "debug",
  "z": "8fc7dd91.e8301",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "false",
  "x": 990,
  "y": 480,
```

```
"wires": []
},
{
  "id": "991544b0.752cd8",
  "type": "ui_button",
  "z": "8fc7dd91.e8301",
  "name": "",
  "group": "4e52fc0f.c72334",
  "order": 1,
  "width": 10,
  "height": 1,
  "passthru": false,
  "label": "Reset",
  "tooltip": "",
  "color": "",
  "bgcolor": "cyan",
  "icon": "",
  "payload": "true",
  "payloadType": "bool",
  "topic": "",
  "x": 230,
  "y": 480,
  "wires": [
    [
      "8f5ecc90.2f644"
    ]
  ]
},
{
  "id": "8f5ecc90.2f644",
  "type": "switch",
  "z": "8fc7dd91.e8301",
  "name": "",
  "property": "payload",
  "propertyType": "msg",
  "rules": [
    {
```

```
        "t": "true"
    }
],
"checkall": "false",
"repair": false,
"outputs": 1,
"x": 410,
"y": 480,
"wires": [
    [
        "3960d45f.1d4dec",
        "c3c0db1a.b1ae68"
    ]
]
},
{
    "id": "3960d45f.1d4dec",
    "type": "function",
    "z": "8fc7dd91.e8301",
    "name": "",
    "func": "msg.payload = \"welcome\\\"\\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 570,
    "y": 480,
    "wires": [
        [
            "6df013ed.e3da4c"
        ]
    ]
},
{
    "id": "779a9f83.d21de",
    "type": "inject",
    "z": "8fc7dd91.e8301",
    "name": "",
    "topic": "",
```

```
"payload": "true",
"payloadType": "bool",
"repeat": "",
"crontab": "",
"once": true,
"onceDelay": "0.0",
"x": 210,
"y": 380,
"wires": [
  [
    "8f5ecc90.2f644"
  ]
],
},
{
  "id": "c3c0db1a.b1ae68",
  "type": "function",
  "z": "8fc7dd91.e8301",
  "name": "",
  "func": "msg.payload = \"\\\"\\nreturn msg;\"",
  "outputs": 1,
  "noerr": 0,
  "x": 190,
  "y": 240,
  "wires": [
    [
      "2845b07a.afb5",
      "91009782.fd69b8"
    ]
  ]
},
{
  "id": "4e52fc0f.c72334",
  "type": "ui_group",
  "z": "",
  "name": "Chatbox",
  "tab": "5a1fa65c.febfb8",
```

```

    "order": 1,
    "disp": true,
    "width": "10",
    "collapse": false
  },
  {
    "id": "3a610ff8.ae3c5",
    "type": "ui_group",
    "z": "",
    "name": "Bot Response",
    "tab": "5a1fa65c.febfb8",
    "order": 2,
    "disp": true,
    "width": "15",
    "collapse": false
  },
  {
    "id": "5a1fa65c.febfb8",
    "type": "ui_tab",
    "z": "",
    "name": "Intelligent Customer Help Desk With Smart Document Understanding",
    "icon": "dashboard",
    "disabled": false,
    "hidden": false
  }
]

```

Reference:

- a. https://www.ibm.com/cloud/architecture/tutorials/cognitive_discovery
- b. <https://cloud.ibm.com/docs/assistant?topic=assistant-getting-started>
- c. <https://developer.ibm.com/recipes/tutorials/how-to-create-a-watson-chatbot-on-nodered/>

- d. <http://www.iotgyan.com/learning-resource/integration-of-watson-assistant-to-node-red>
- e. <https://github.com/IBM/watson-discovery-sdu-with-assistant>
- f. <https://www.youtube.com/watch?v=Jpr3wVH3FVA>