

# **Project Report**

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

**Category:** *Artificial Intelligence*

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

**By :** *KUSHAGRA CHATURVEDY (18ucs229@lnmiit.ac.in)*

**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:** The aim of this project is to develop an application that utilizes 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:** IBM Cloud, IBM Watson, NodeRED
- ❖ **Functional Requirements:** IBM cloud
- ❖ **Technical Requirements:** AI,ML,WATSON AI,PYTHON
- ❖ **Software Requirements:** Watson assistant, Watson discovery.
- ❖ **Project Deliverables:** Smartinternz Internship
- ❖ **Project Team:** Kushagra Chaturvedy
- ❖ **Project Duration:** 29 days

## 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:**

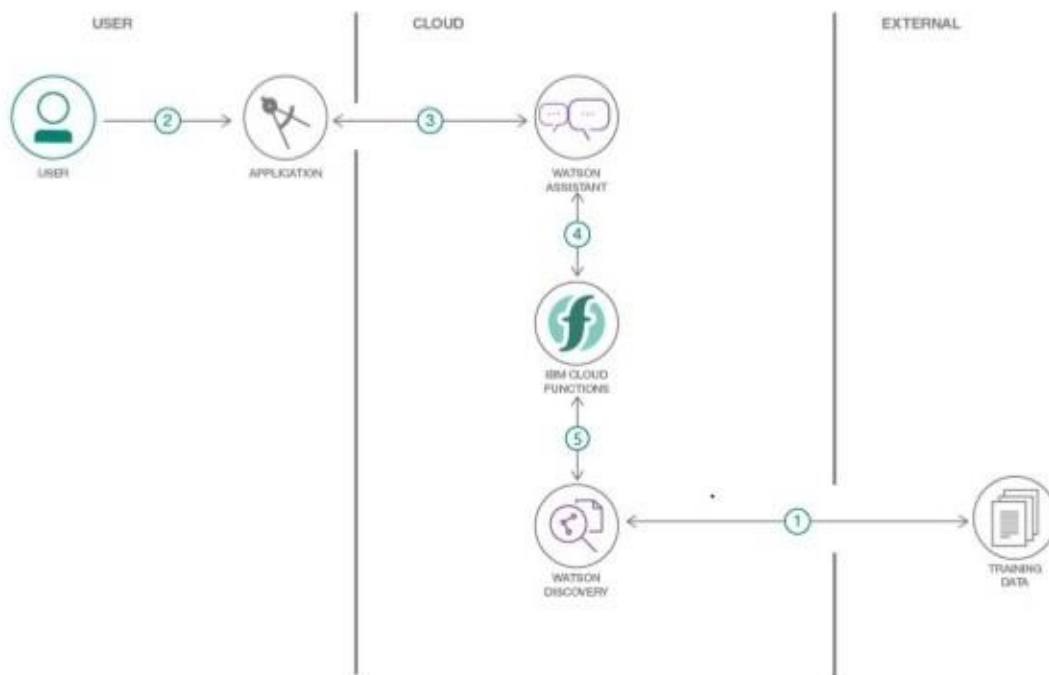
Presently Chatbots are used as a means of getting input from users and getting only response questions and for some questions. In the case where the chatbot is unable to process the user's request, it directs the user to an agent that they can contact for further help. To ensure that the chatbot is effective in processing the user's requests it should be aware of the establishment's details and the details of the product itself and be.

### **2.2 Proposed solution:**

In order for the chatbot to be aware of the product and its functionality, it needs to . The virtual agent should be trained from some insight records based on company background so it can answer queries based on the product or related to company. In this project I used Watson Discovery and Watson Assistant to achieve the above solution. The UI for the chatbot was made using NodeRED.

## **THEORETICAL 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 made from NodeRED. 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 skill.
4. If the user asks a question pertaining to product information, 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

### *Importing the document*

Launch the Watson Discovery tool and create a new data collection by selecting the Upload your own data option. Here we upload the upload the ecobee3\_UserGuide.

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

For now, Smart Document Understanding is not applied to our document and so running queries about the product will not yield any useful information. Once we annotate the various sections of the manual and split the document using subtitles, only then will Watson train on the data properly and identify the necessary keywords.

Click the Build your own query link.

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(Smart Document Understanding)**

From the Discovery collection panel, click the Configure data button (located in the top right corner) to start the SDU process.

We will manually label each part of the document as relevant text, footers, or subtitles. As we 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 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

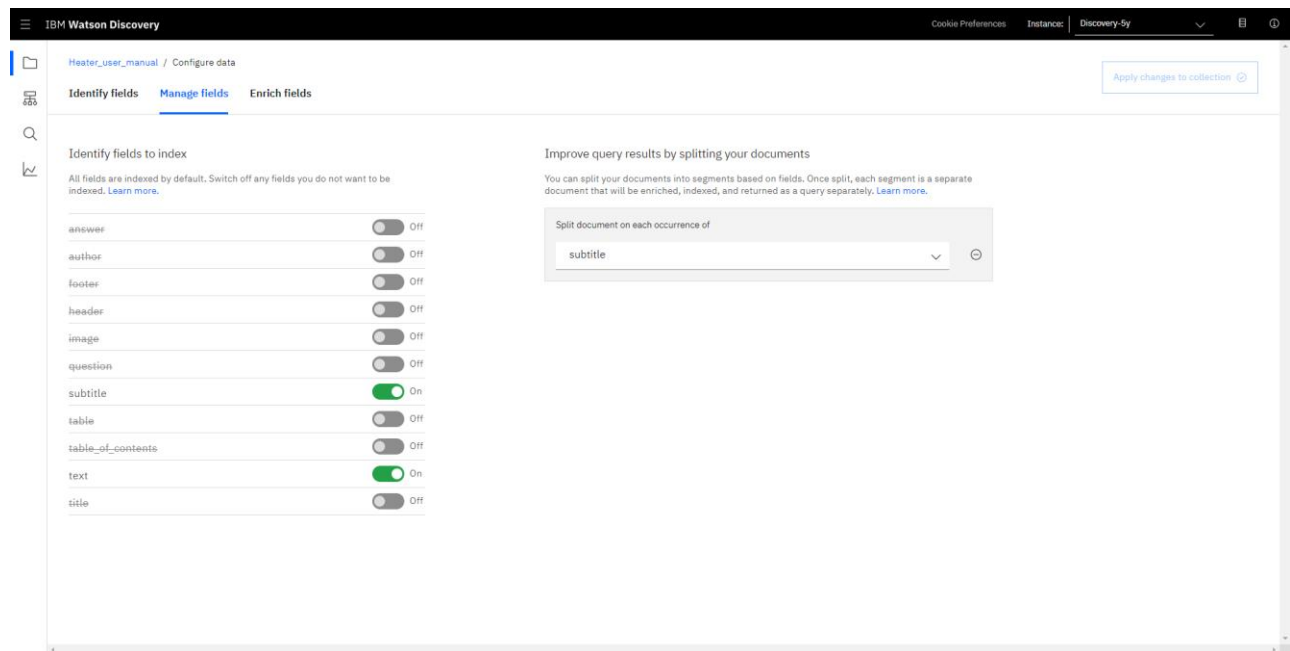
All page numbers as footers

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.



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

Then we split the annotated document into several document using the subtitle.  
Submit your changes.

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



The screenshot shows the IBM Watson Discovery Overview page. At the top, it indicates 131 documents, 0 failed documents, and creation/update timestamps from 5/23/2020. A sidebar on the left contains navigation icons. The main content area is divided into several sections:

- Identified 5 fields from your data:** footer, subtitle, table\_of\_contents, text, title. A link 'Need to identify more fields? Add fields' is present.
- Added 6 enrichments to your data:**
  - Entity Extraction:** 0.3°C (4) | 0.5°F (4) | 10 °F (4) | 900 seconds (4) | 20 min (3)
  - Concept Tagging:** Heat (13) | Netscape (13) | Yahoo! (13) | Internet (13) | HVAC (9)
  - Keyword Extraction:** Main Menu (47) | ecobee3 (43) | Thermostat (41) | Settings (39) | thermostat (22)
  - Sentiment Analysis:** 58% positive, 30% neutral, 12% negative
  - Category Classification:** technology and com... operating systems
  - Semantic Role Extraction:** No preview available
- Now you're ready to query!**
  - Most common entity types and their top entities (Run)
  - Top people related to /technology and computing/operating systems (Run)
  - Documents about 0.3°C as a Quantity with a very negative sentiment (Run)

At the bottom, it states '3 enrichments available. Add enrichments'.

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. The left sidebar has a 'Build queries' section with a 'Use sample query' button. The main area shows the search query 'How do I turn on the heater?' and options to 'Use natural language' or 'Use the Discovery Query Language'. Below the query, there are filters for 'Include analysis of your results' and 'Filter which documents you query', along with a 'More options' link. At the bottom, there are 'Run query' and 'Close' buttons.

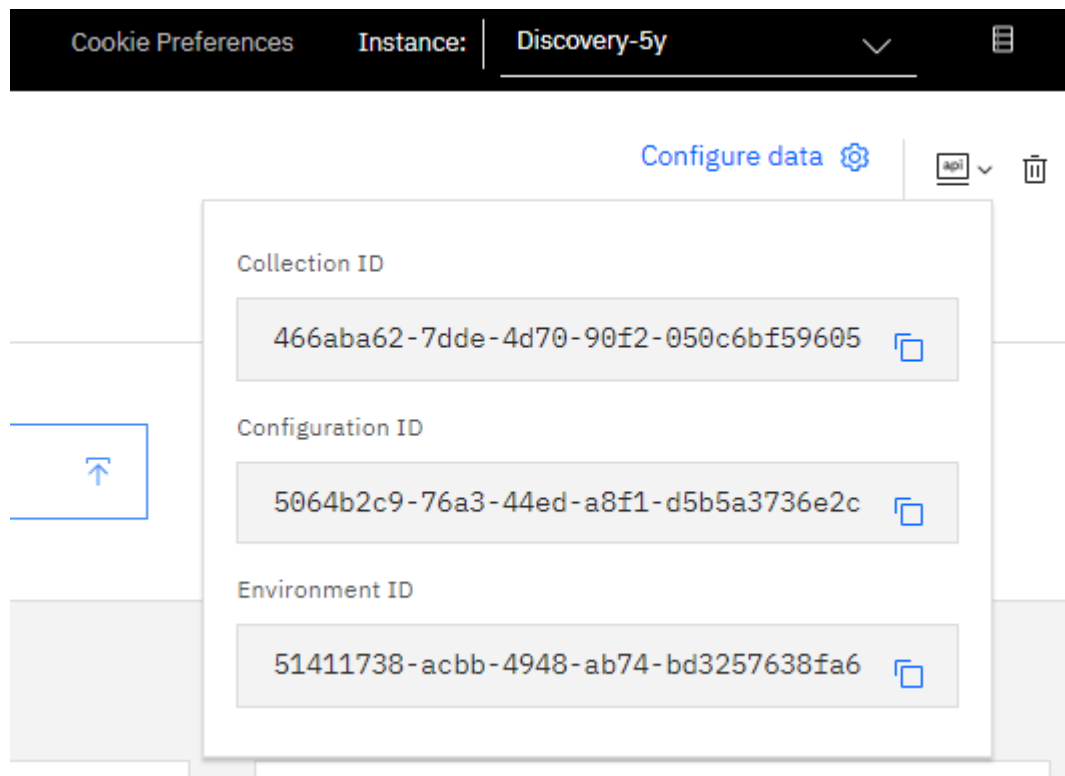
The right panel displays the results for the query 'How do I turn on the heater?'. It shows a 'Summary' tab and a 'JSON' tab. The 'Summary' tab includes a 'Query URL' and a 'Passages' section with several text excerpts. The 'JSON' tab shows a table of results for the document 'ecobee3\_UserGuide.pdf'.

Sentiment	Text
positive	"...Select Screen sleeps when I sleep if you want to make the screen blank during the Sleep activity period..."
positive	"...You can also configure the screen to automatically sleep (i.e. turn off) whenever your ecobee3 enters the Sleep activity period. For example, if your thermostat is located in a bedroom, you may want to blank the screen when you are asleep, whereas if the thermostat is in a hallway, you may want the screen displayed all the time."

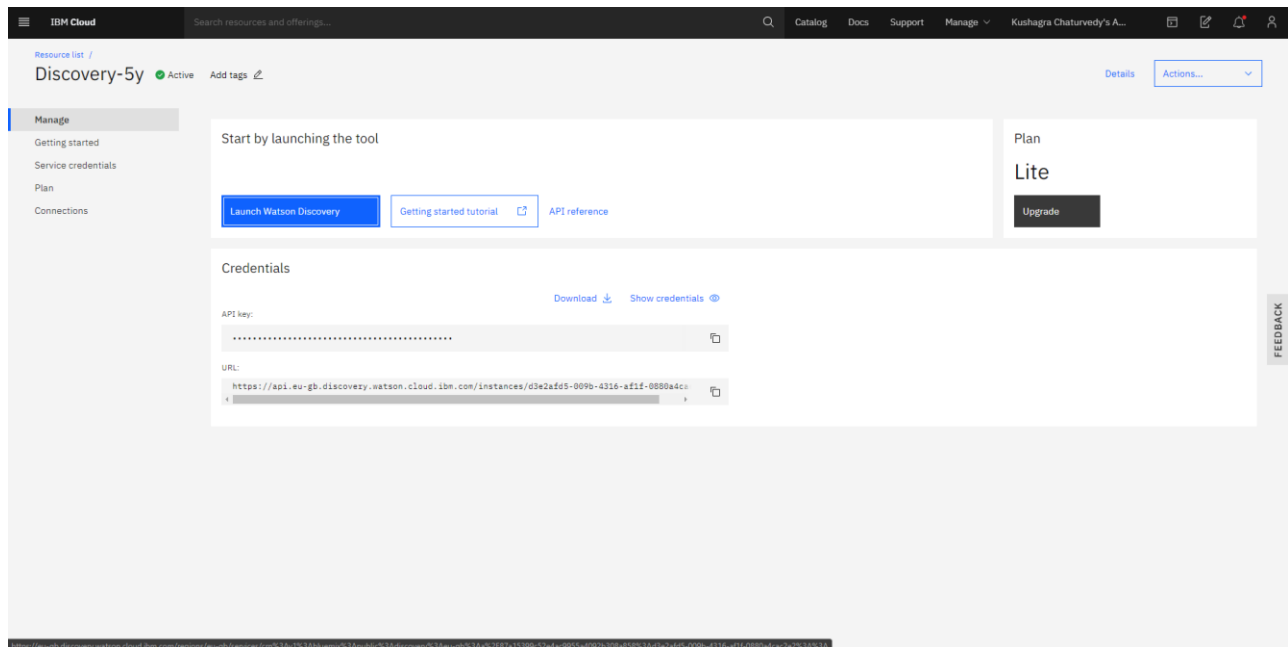
Store credentials for future use

In upcoming steps, you will need to provide the credentials to access your Discovery collection. The values can be found in the following locations. These values will be used later as parameters in our cloud function.

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



For credentials, return to the Dashboard of Watson Discovery Service and you'll find the API key(or the iam\_apikey) and the URL in the fields as shown below.



### 3. Create IBM Cloud Functions action

Now let's create the web action that will make queries against our Discovery collection. In the Catalog Section, search for 'Functions' to select Cloud Functions. Next, choose actions from the menu at the left.

From the Create panel, select the Create Action option.

On the Create Action panel, provide a unique Action Name.

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

Functions / Actions / watson\_disc\_func

watson\_disc\_func Web Action

Namespace: 18ucs229@inmilit.ac.in\_dev(Dallas)

Code

```
1 * /**
2  * @param {Object} params
3  * @param {string} params.iam_apikey
4  * @param {string} params.url
5  * @param {string} params.username
6  * @param {string} params.password
7  * @param {string} params.environment_id
8  * @param {string} params.collection_id
9  * @param {string} params.configuration_id
10 * @param {string} params.input
11 *
12 * @return {Object}
13 */
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     let discovery;
32
33     if (params.iam_apikey) {
34       discovery = new DiscoveryV1({
35         iam_apikey: params.iam_apikey,
36         url: params.url,
37         version: '2019-09-25'
38       });
39     }
40     else {
41       discovery = new DiscoveryV1({
42         username: params.username,
43         password: params.password,
44         url: params.url,
45         version: '2019-09-25'
46       });
47     }
48
49     discovery.query({
50       environment_id: params.environment_id,
51       collection_id: params.collection_id,
52     });
53   });
54 }
```

Invoke

In the code editor window, cut and paste in the code from the discovery-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:

Functions / Actions / watson\_disc\_func

watson\_disc\_func Web Action

Namespace: 18ucs229@inmilit.ac.in\_dev(Dallas)

Parameters

Parameter Name	Parameter Value
url	"https://api.eu-gb.discovery.watson.cloud.ibm.com/instances/d3e2afd5-009b-4316-af1f-0880a4cac2e2"
environment_id	"51411738-acbb-4948-ab74-bd3257638fa6"
collection_id	"466aba62-7dde-4d70-90f2-050c6bf59605"
iam_apikey	"NuY5c50ngJuLxARixDpnIAiGX9KhC2roKmOrfNuuAY0"

Add Parameter

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:

The screenshot shows the AWS Lambda console for a function named `watson_disc_func`. The **Code** panel displays the following JavaScript code:

```
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
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
30 function main(params) {
31   return new Promise(function (resolve, reject) {
32     let dtcDiscovery;
33
34     if (params.iam_apikey) {
35       dtcDiscovery = new DiscoveryV1({
36         iam_apikey: params.iam_apikey,
37         url: params.url,
38         version: '2019-03-28'
39       });
40     } else {
41       dtcDiscovery = new DiscoveryV1({
42         username: params.username,
43         password: params.password,
44         url: params.url,
45         version: '2019-03-28'
46       });
47     }
48
49     dtcDiscovery.query({
50       environment_id: params.environment_id,
51       collection_id: params.collection_id,
52
```

The **Activations** panel shows the results of the function invocation, including the **Activation ID** and the **Results** returned from the Discovery service:

```
{
  "matching_results": 133,
  "passages": 13,
  "results": [
    {
      "enriched_text": [
        {
          "categories": [
            {
              "label": "/Business and Industrial/energy",
              "score": 0.89535
            },
            {
              "label": "/Business and Industrial/green solutions",
              "score": 0.64353
            },
            {
              "label": "/Business and Industrial/business operations",
              "score": 0.63324
            }
          ],
          "concepts": [
            {
              "discovery_resource": "http://dbpedia.org/resource/Fahrenheit",
              "relevance": 0.88227,
              "text": "Fahrenheit"
            },
            {
              "discovery_resource": "http://dbpedia.org/resource/Temperature",
              "relevance": 0.80400,
              "text": "Temperature"
            },
            {
              "discovery_resource": "http://dbpedia.org/resource/Celsius",
              "relevance": 0.84400,
              "text": "Celsius"
            },
            {
              "discovery_resource": "http://dbpedia.org/resource/Absolute_zero",
              "relevance": 0.78623,
              "text": "Absolute zero"
            },
            {
              "discovery_resource": "http://dbpedia.org/resource/Boiling_point",
              "relevance": 0.74636
            }
          ]
        }
      ]
    }
  ]
}
```

Next, go to the Endpoints panel:

The screenshot shows the IBM Cloud Functions console for the function 'watson\_disc\_func'. The left sidebar contains navigation links: Code, Parameters, Runtime, Endpoints (highlighted), Connected Triggers, Enclosing Sequences, and Logs. The main panel is titled 'Web Action' and includes a checkbox 'Enable as Web Action' which is checked. Below this is a table for HTTP endpoints:

HTTP Method	Auth	URL
ANY	Public	https://us-south.functions.cloud.ibm.com/api/v1/web/18ucs229%40nmiit.ac.in_dev/default/watson_disc_func

Below the HTTP endpoints table is a section for 'REST API' with another table:

HTTP Method	Auth	URL
POST	API-KEY	https://us-south.functions.cloud.ibm.com/api/v1/namespaces/18ucs229%40nmiit.ac.in_dev/actions/watson_disc_func

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

Take note of the Web Action URL value, as this will be needed by the Watson Assistant when we configure it for webhooks.

#### 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 the intent `#Product_Information`. The interface includes a header with a back arrow, the intent name, and a 'Try it' button. Below the header, there are sections for 'Intent name', 'Description (optional)', and 'User example'. The 'User example' section contains a list of 18 examples, with the first 7 visible. The examples are:

Example	Added
How can I start the heater?	a day ago
How do I access the settings?	a day ago
How do I set the timer?	a day ago
How do I start the heater?	a day ago
How do I turn on the heater?	a day ago
How do I turn on the thermostat?	a day ago

At the bottom, there is a pagination bar showing 'Showing 1-18 of 18 examples' and a '1 of 1 pages' indicator.

### Create new dialog node



Now we need to add a node to handle our intent. Add a new dialog node called details about product.

The screenshot shows the 'Create new dialog node' interface. It displays a list of three dialog nodes, each with a title, an intent name, and a status bar indicating the number of responses, context set, and return status.

Dialog Node	Intent	Status
What can I do	#Help	1 Responses / 0 Context Set / Does not return
Details about product	#Product_Information	2 Responses / 0 Context Set / Does not return
anything_else		1 Responses / 0 Context Set / Returns

Assign the newly created intent to it.

Details about product|


Customize  


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

Settings

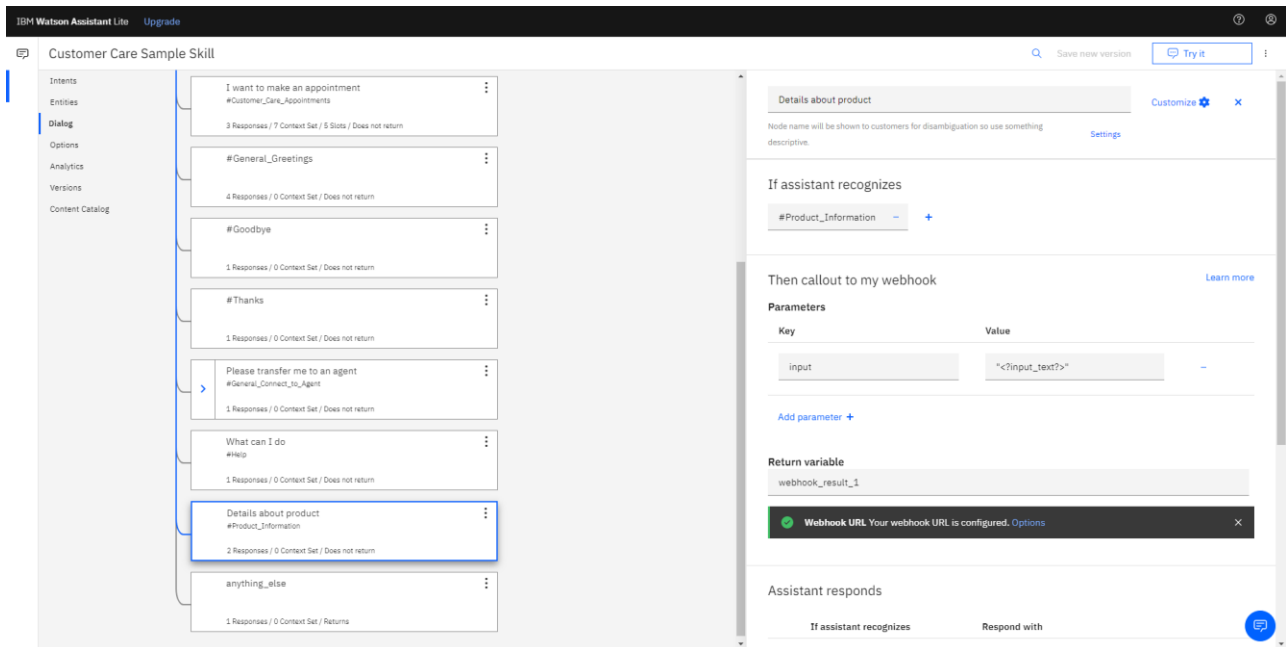
If assistant recognizes

#Product\_Information









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.

### Enabling webhooks from Assistant

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

Select the Options tab and select webhooks from the sub menu:



## Customer Care Sample Skill

[Intents](#)[Entities](#)[Dialog](#)[Options](#)**Webhooks**[Disambiguation](#)[Autocorrection](#)[Irrelevance Detection](#)[System Entities](#)[Analytics](#)[Versions](#)[Content Catalog](#)

## Webhooks

A webhook is a mechanism that allows you to call out to an external program based on events in your dialog.

### Webhook setup

Specify the request URL for an external API you want to be able to invoke from dialog nodes. Watson will call this URL when configured to do so from a dialog node. [Learn more](#)

URL

`https://us-south.functions.cloud.ibm.com/api/v1/web/18ucs229%40lnmiit.i`

### Headers

Add HTTP headers for authorization or any other parameters required for invoking the webhook.

Header name

Header value

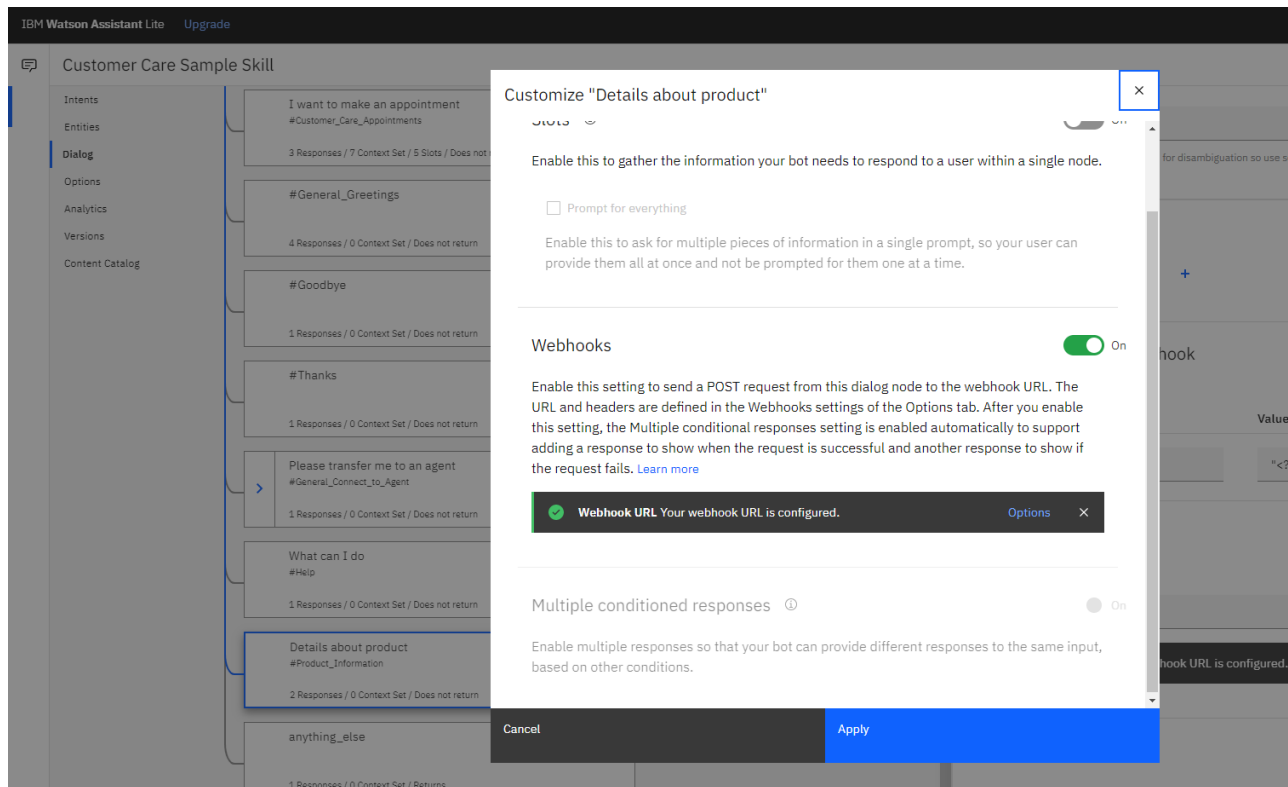
[Add header](#)  [Add authorization](#) 

### Next step

To trigger this webhook from an individual dialog node, enable webhooks from the Customize page of the node. [Go to dialog](#)

Enter the public URL endpoint for your action.



Return to the Dialog tab, and click on the Details 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 `$webhook_result_1`. This is the variable name you can use to access the result from the Discovery service query. Also set the response to the variable to `"<?$webhook_result_1.passages[0].passage_text?>"`. This will select the required output from the other entries in the array in the variable.


Details about product


Customize  

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

Settings



Parameters

Key	Value	
input	"<?input_text?>"	





Add parameter 

Return variable

webhook\_result\_1

 **Webhook URL** Your webhook URL is configured. [Options](#) 

Assistant responds

	If assistant recognizes	Respond with		
1	\$webhook_result_1	"<?\$webhook_result_1.passage"		
2	anything_else	I'm afraid I don't understand. I		

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?>"

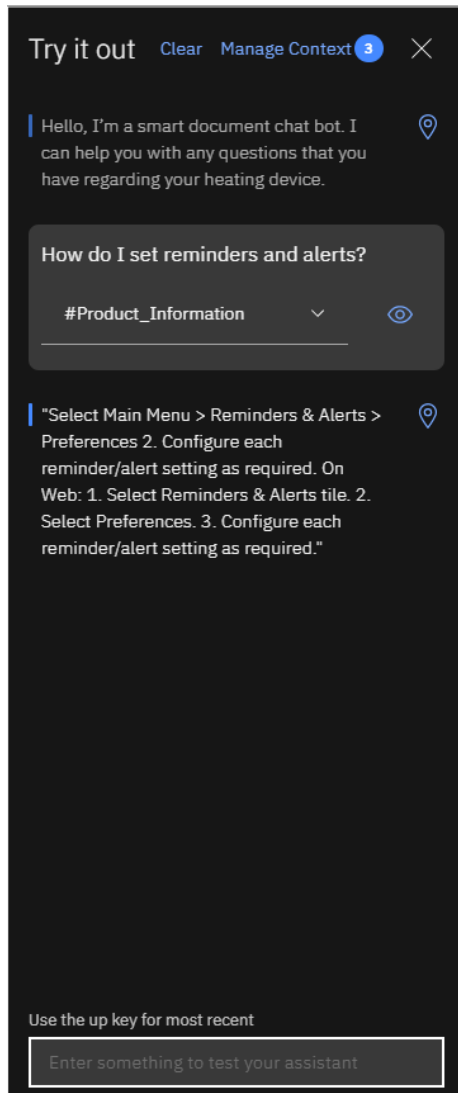
If you fail to do this, Discovery will return results based on a blank query.

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:

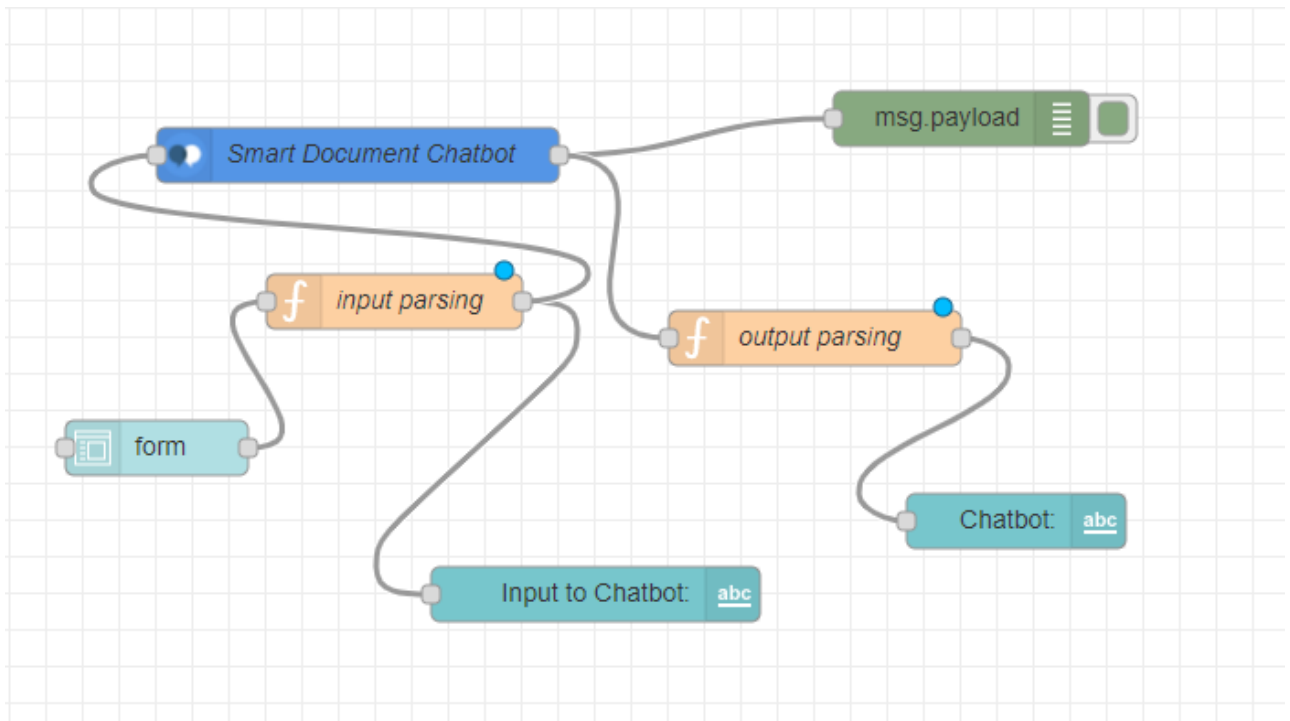


Note that the input "how do I turn on the heater?" has triggered our Details about Product dialog node, which is indicated by the #Product\_Information response.

### Create flow and configure node:

#### Integration of Watson Assistant in Node-RED

- Create the flow given below and modify the form to create the Groups and Tabs for the interface:



- Double click on the Watson assistant node and enter the API key, URL and workspace id of your Watson assistant service

## Edit assistant node


Delete


Cancel


Done


### Properties





 Name

 Username

 Password

 API Key

 Service Endpoint

 Workspace ID

 Timeout Period

☒ Save context

☐ Multiple Users

☐ Permit Empty Payload

☐ Opt Out Request Logging

- After entering all the information click on Done.
- Next, double click on the input parsing cloud function and enter the code as shown below.

**Edit function node**

Delete Cancel Done

⚙️ **Properties**

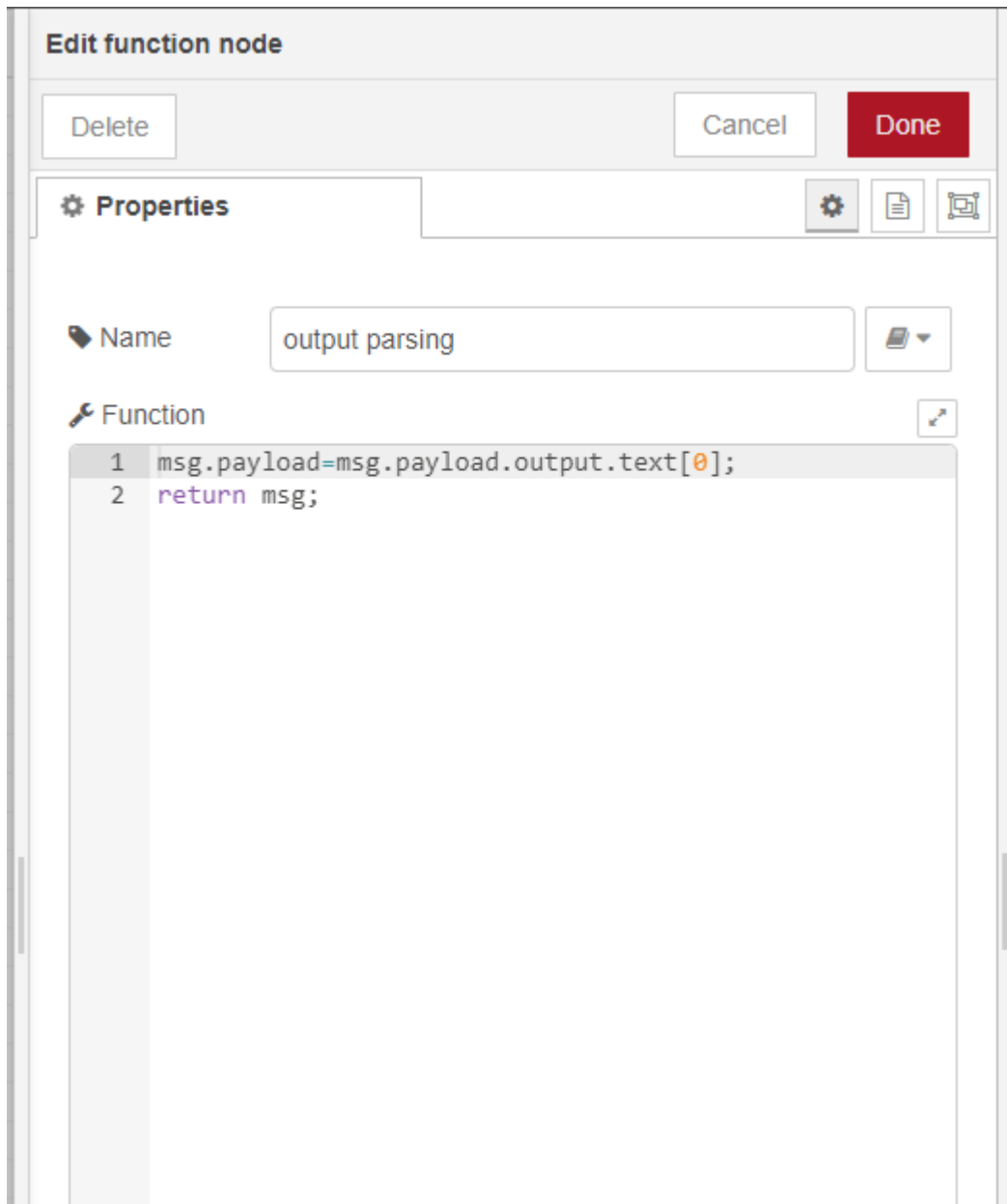
🔑 Name input parsing

🔧 Function

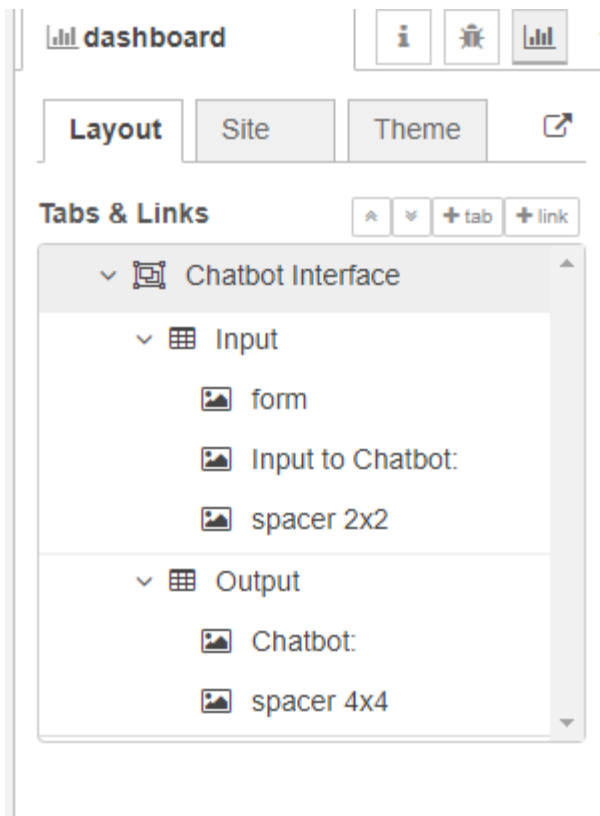
```
1 msg.payload=msg.payload.text;
2 return msg;
```

- Then double click on the output parsing node and enter the code below:





- Set the layout of the groups and set the theme:



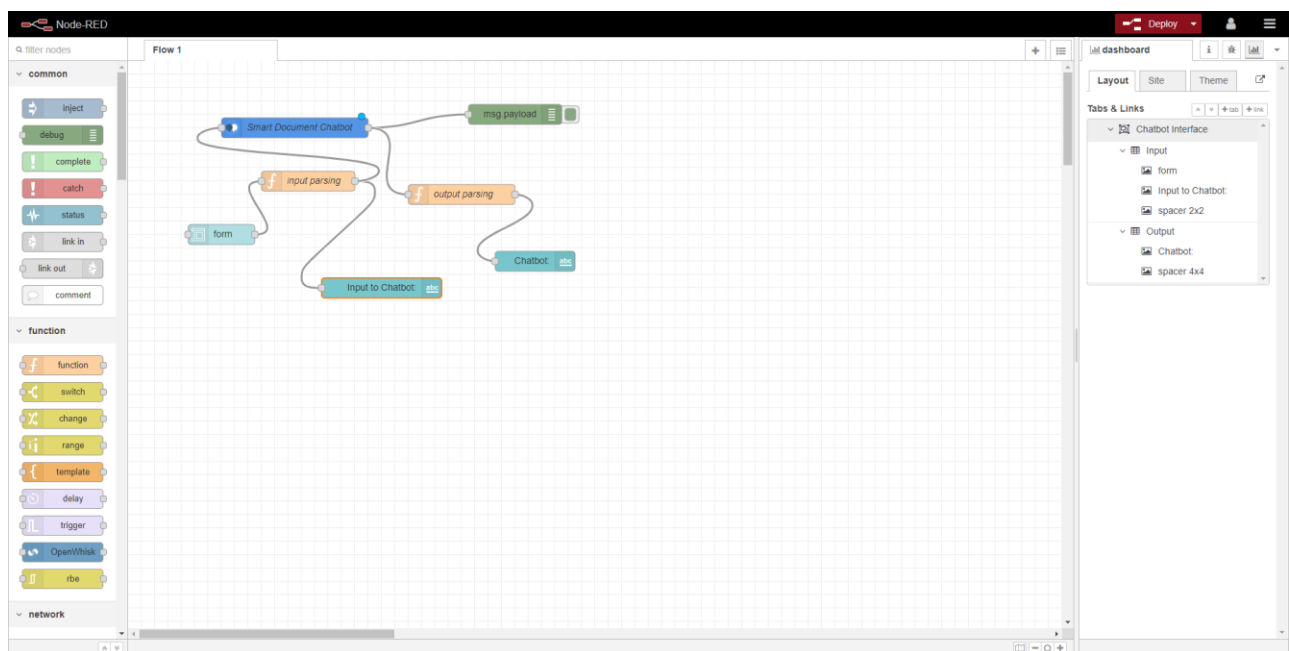
- Finally click on the deploy button.

## FLOWCHART

First go to manage palette 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 the URL <https://node-red-zyayx.mybluemix.net/ui/> in the browser



## **ADVANTAGES & DISADVANTAGES**

### **Advantages:**

- ✓ Companies can deploy chatbots to rectify simple and general human queries and automating the task of user interaction thereby saving resources.
- ✓ Reduces man power.
- ✓ Cost efficient
- ✓ No need to divert calls to customer agent and customer agent can look on other works.

### **Disadvantages:**

- ✓ Sometimes chatbot can mislead customers. They have to be trained extensively in order to be deemed to be at industry standards.
- ✓ Giving same answer for different sentiments.
- ✓ Sometimes the chatbot may not understand the customer's 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.

## **BIBILOGRAPHY**

## **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': '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": "4129cd99.0667f4",
    "type": "tab",
    "label": "Flow 1",
    "disabled": false,
    "info": ""
  },
  {
    "id": "e8624de1.e3fba8",
    "type": "ui_form",
    "z": "4129cd99.0667f4",
    "name": "",
    "label": "",
    "group": "aae28a50.4bac88",
    "order": 1,
    "width": 0,
    "height": 0,
    "options": [
      {
        "label": "Input to chatbot:",
        "value": "text",
        "type": "text",
        "required": true,
        "rows": null
      }
    ],
    "formValue": {
      "text": ""
    },
    "payload": "",
    "submit": "submit",
    "cancel": "cancel",
    "topic": "",
    "x": 140,
    "y": 260,
    "wires": [

```



```

    [
      "5ed34f0a.2288f"
    ]
  ],
  {
    "id": "5ed34f0a.2288f",
    "type": "function",
    "z": "4129cd99.0667f4",
    "name": "input parsing",
    "func": "msg.payload=msg.payload.text;\nreturn msg;",
    "outputs": 1,
    "noerr": 0,
    "x": 270,
    "y": 180,
    "wires": [
      [
        "b5e1f5ee.86867",
        "f1f813d0.d9d398"
      ]
    ]
  },
  {
    "id": "b5e1f5ee.86867",
    "type": "ui_text",
    "z": "4129cd99.0667f4",
    "group": "aae28a50.4bac88",
    "order": 2,
    "width": 0,
    "height": 0,
    "name": "",
    "label": "Input to Chatbot:",
    "format": "{{msg.payload}}",
    "layout": "row-spread",
    "x": 380,
    "y": 340,
    "wires": []
  },
  {
    "id": "f1f813d0.d9d398",
    "type": "watson-conversation-v1",
    "z": "4129cd99.0667f4",
    "name": "Smart Document Chatbot",
    "workspaceid": "50eb71d0-1065-4677-b93d-7c8c8322363f",
    "multiuser": false,
    "context": true,
    "empty-payload": false,
    "service-endpoint": "https://api.us-south.assistant.watson.cloud.ibm.com/instances/dccd889e-113e-

```

```
47bb-b737-47cb0e08915a",
  "timeout": "",
  "optout-learning": false,
  "x": 250,
  "y": 100,
  "wires": [
    [
      "7a860a6a.c2de7c",
      "a04f6ac3.e8d248"
    ]
  ]
},
{
  "id": "7a860a6a.c2de7c",
  "type": "debug",
  "z": "4129cd99.0667f4",
  "name": "",
  "active": true,
  "tosidebar": true,
  "console": false,
  "tostatus": false,
  "complete": "false",
  "x": 580,
  "y": 80,
  "wires": []
},
{
  "id": "a04f6ac3.e8d248",
  "type": "function",
  "z": "4129cd99.0667f4",
  "name": "output parsing",
  "func": "msg.payload=msg.payload.output.text[0];\nreturn msg;",
  "outputs": 1,
  "noerr": 0,
  "x": 500,
  "y": 200,
  "wires": [
    [
      "b99a9381.346ad"
    ]
  ]
},
{
  "id": "b99a9381.346ad",
  "type": "ui_text",
  "z": "4129cd99.0667f4",
  "group": "df740db4.e9c958",
  "order": 1,
```

```

    "width": 0,
    "height": 0,
    "name": "",
    "label": "Chatbot:",
    "format": "{{msg.payload}}",
    "layout": "row-center",
    "x": 610,
    "y": 300,
    "wires": []
  },
  {
    "id": "aae28a50.4bac88",
    "type": "ui_group",
    "z": "",
    "name": "Input",
    "tab": "90c79892.5268a",
    "order": 1,
    "disp": true,
    "width": "6",
    "collapse": false
  },
  {
    "id": "df740db4.e9c958",
    "type": "ui_group",
    "z": "",
    "name": "Output",
    "tab": "90c79892.5268a",
    "order": 2,
    "disp": true,
    "width": "15",
    "collapse": false
  },
  {
    "id": "90c79892.5268a",
    "type": "ui_tab",
    "z": "",
    "name": "Chatbot Interface",
    "icon": "dashboard",
    "disabled": false,
    "hidden": false
  }
]

```

## **Reference:**

1. [https://www.ibm.com/cloud/architecture/tutorials/cognitive\\_discovery](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://www.youtube.com/watch?v=Jpr3wVH3FVA>