

Project Report

NAME	SHIVAM GUPTA (sag72434@gmail.com)
TITLE	Intelligent Customer Help Desk With Smart Document Understanding
CATEGORY	Artificial Intelligence

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

1.

INTRODUCTION

Overview:

We will be able to write an application that leverages multiple Watson AI Services (Discovery, Assistant, Cloud functions 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, Node- RED
- Functional Requirements: IBM cloud
- Technical Requirements: AI,ML,WATSONAI,PYTHON
- Software Requirements: Watson assistant, Watson discovery.
- Project Deliverables: Smartinternz Internship
- Project Team: Shivam Gupta
- Project Duration:29 days

Purpose:

The typical customer care chat-bot 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 owner's manual. So now, instead of "Would you like to speak to a customer representative?" we can return relevant sections of the owner's 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 owner's 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 webaction 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

2. LITERATURE SURVEY

Existing problem:

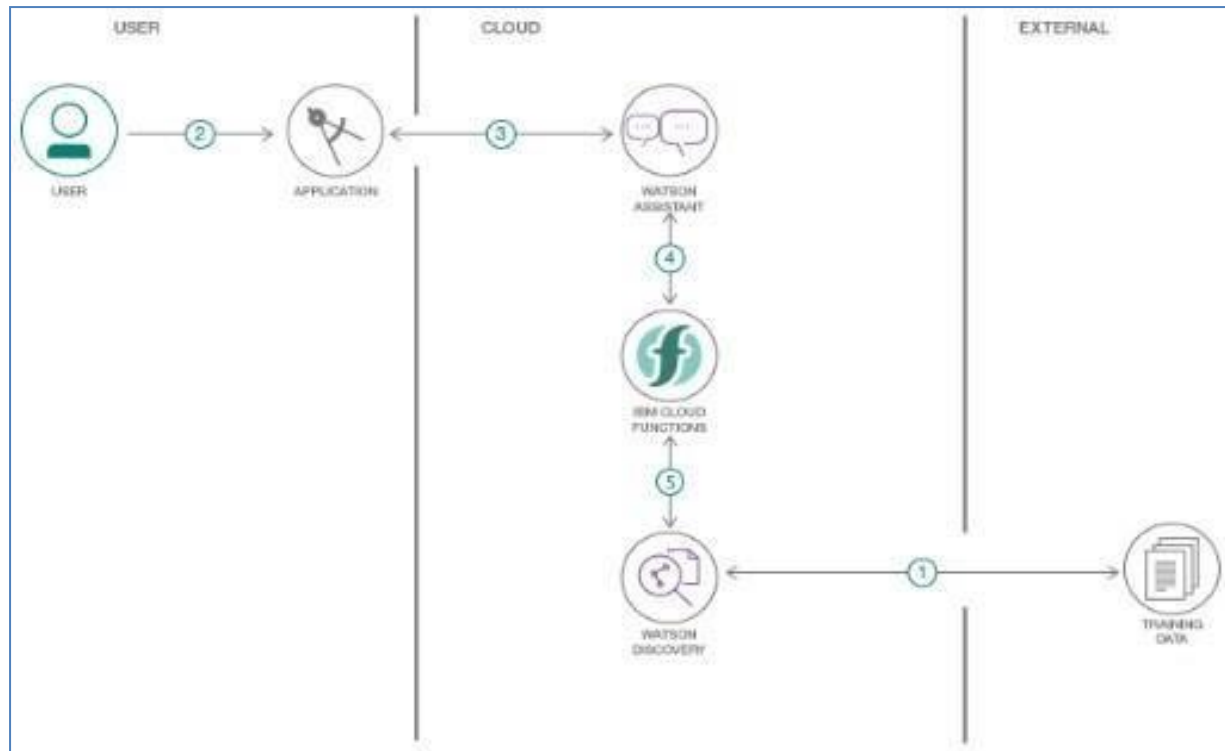
Generally Chat-bots means getting input from users and getting only response questions and for some questions the output from bot will be like "try again", "I don't understand", "will you repeat again", and so on... and directs customer to customer agent but a good customer Chat-bot should minimize involvement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

Proposed solution:

For the above problem to get solved we have to put an virtual agent in chat-bot 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

3. 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 chat-bot 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 Redapp.

4. EXPERIMENTAL INVESTIGATIONS

1. Create IBM Cloud services

Create the following services:

- Watson Discovery
- Watson Assistant
- Node Red
- IBM cloud function


Creation of Node-RED in IBM cloud:

- Step-1: Login to IBM and go to the catalog
- Step-2: Search for node-red and select “Node-RED Starter “ Service
- Step-3: Enter the Unique name and click on create a button
- Note: Your Node-red service is starting
- Step – 5: We have to configure Node red for the first time. Click on next to continue

Welcome to your new Node-RED instance on IBM Cloud

We know you're eager to start wiring up your flows, but first there are a couple of tasks you should do:

- Secure your Node-RED editor
- Browse available IBM Cloud nodes



[Previous](#) [Next](#)

- Step – 6: Secure your node red editor by giving a username and password and click on Next

Secure your Node-RED editor


☒ Secure your editor so only authorised users can access it

Username

Password Must be at least 8 characters

☐ Allow anyone to view the editor, but not make any changes

☐ *Not recommended:* Allow anyone to access the editor and make changes



[Previous](#) [Next](#)

- Step – 7: Click Next to continue


Browse available IBM Cloud nodes

There are lots of nodes available from the community that can be used to add more capabilities to your application. The list below is just a small selection.

You can find many more nodes on the [Flow Library](#).

You can use the Palette Manager built into editor to search for and install nodes. Alternatively, you can also edit your application's package.json file and adding them to the dependencies section.

node-red-dashboard Quickly create dashboards driven by Node-RED	node-red-contrib-ibm-wiotp-device-ops Perform device and gateway operations using the Watson IoT Platform
node-red-contrib-iot-virtual-device Simulate device behavior and use it to run many device instances	node-red-contrib-objectstore Store, delete and restore objects in the ObjectStore service
node-red-contrib-bluemix-hdfs Connect and control Hadoop using HDFS for Analytics	node-red-contrib-ibmpush Send push notifications to mobile devices using the IBM Cloud Push service



[Previous](#) [Next](#)

- Step – 8: Click Finish

Finish the install

You have made the following selections:

- *Not recommended:* Allow anyone to access the editor and make changes

You can change these settings at any time by setting the following environment variables via the IBM Cloud console:

- `NODE_RED_USERNAME` - the username
- `NODE_RED_PASSWORD` - the password
- `NODE_RED_GUEST_ACCESS` - if set to 'true', allows anyone read-only access to the editor



Previous

Finish

- Step – 9: Click on Go to Node-Red flow editor to launch the flow editor

Node-RED

Flow-based programming for the Internet of Things

Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.

This instance is running as an IBM Cloud application, giving it access to the wide range of services available on the platform.

More information about Node-RED, including documentation, can be found at nodered.org.

[Go to your Node-RED flow editor](#)

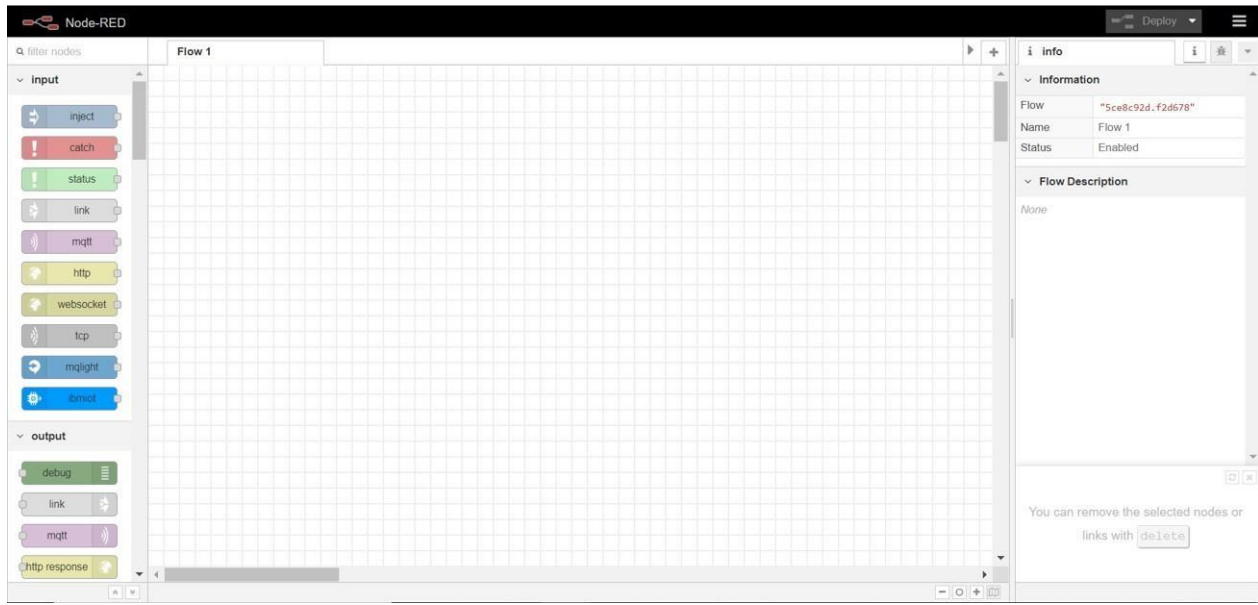
[Learn how to customise Node-RED](#)

Customising your instance of Node-RED

This instance of Node-RED is enough to get you started creating flows.

You may want to customise it for your needs, for example replacing this introduction page with your own, adding http authentication to the flow editor or adding new nodes to the palette.

- Node red editor has various nodes with the respective functionality



Creation of Watson discovery instance in IBM Cloud:

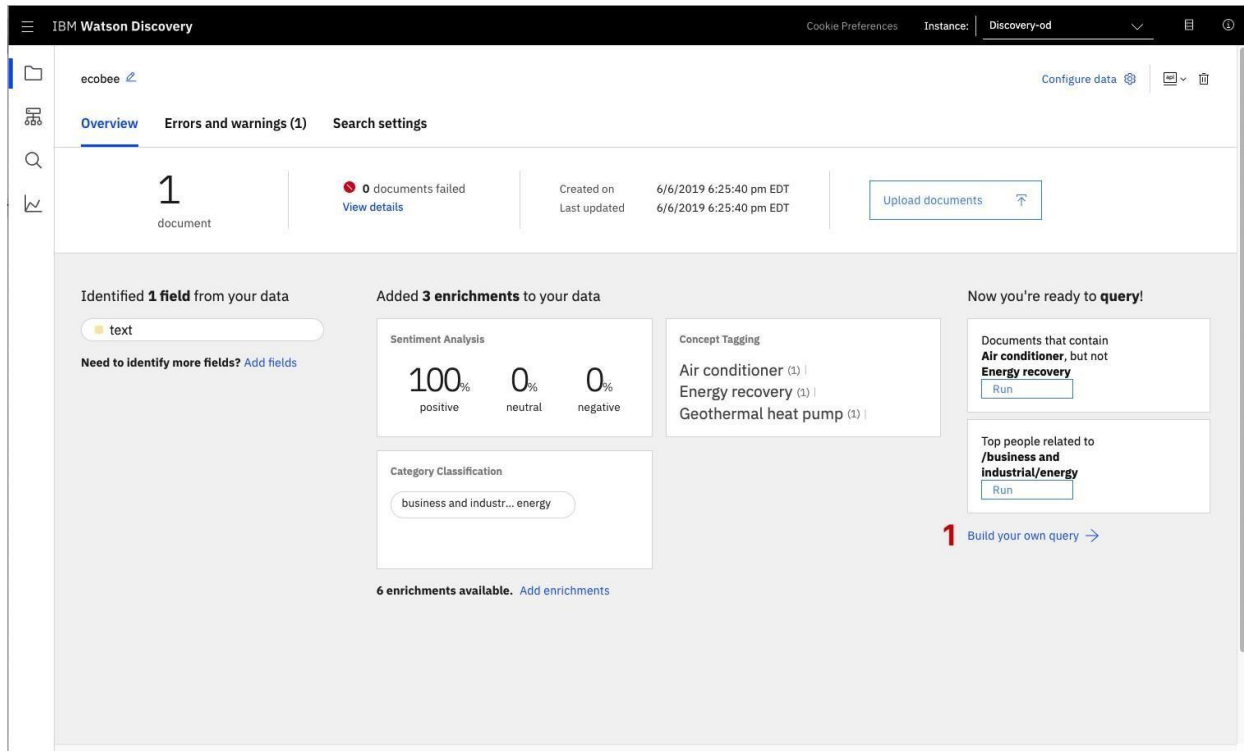
- Import the document

As shown below, 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.

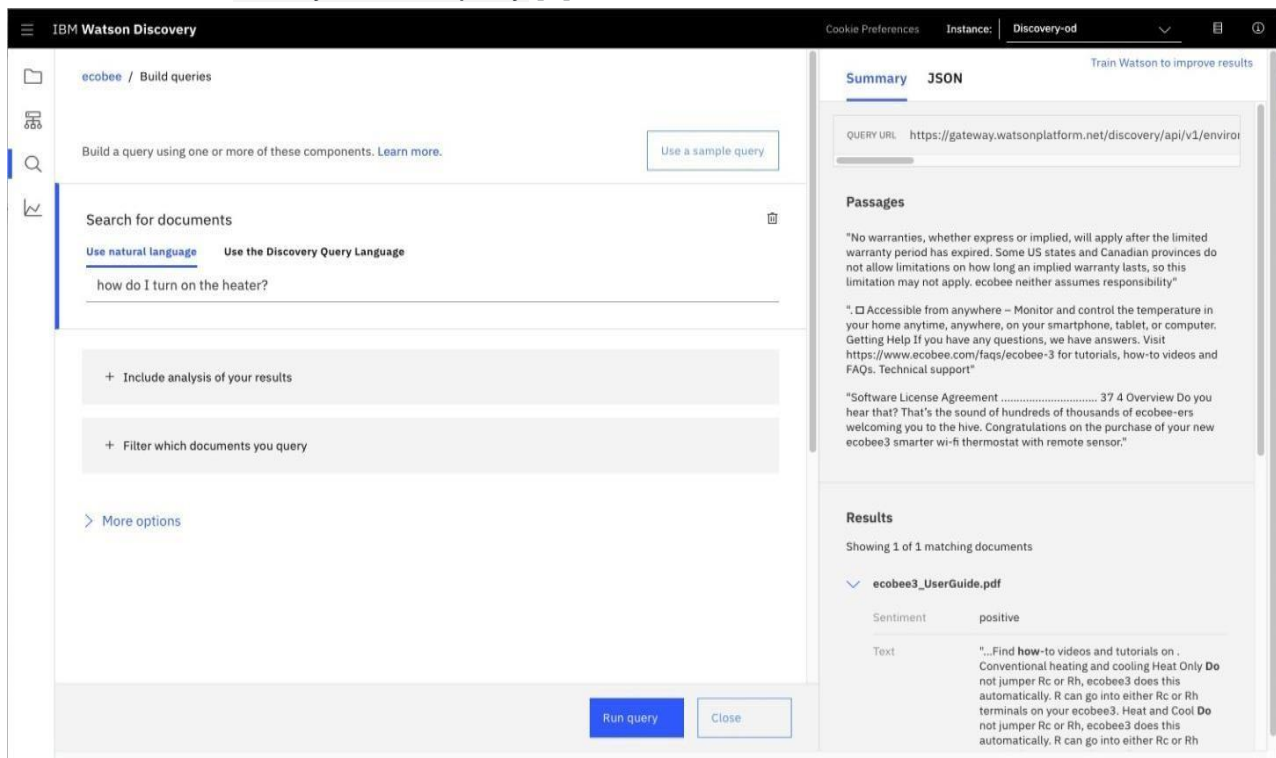
When prompted, select and upload the *NTEG.pdf*.

This is a brochure of a travel company which deals with the travelling across northeastern part of our country.

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 screenshot shows the IBM Watson Discovery 'Identify fields' interface. The top bar indicates the instance is 'Discovery-pr'. The left sidebar shows a table of contents for 'NETG.pdf' with pages 1 through 4. The main document viewer shows page 1, which is titled 'Destination Northeast India'. A pink highlight is placed on the title. The right sidebar, titled 'Field labels', lists various document elements that can be assigned to the highlighted text. The elements listed are: answer, author, footer (highlighted in green), header, question, subtitle, table_of_contents, text, title, image, and table. A 'Submit page' button is located at the bottom right of the document viewer.

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.

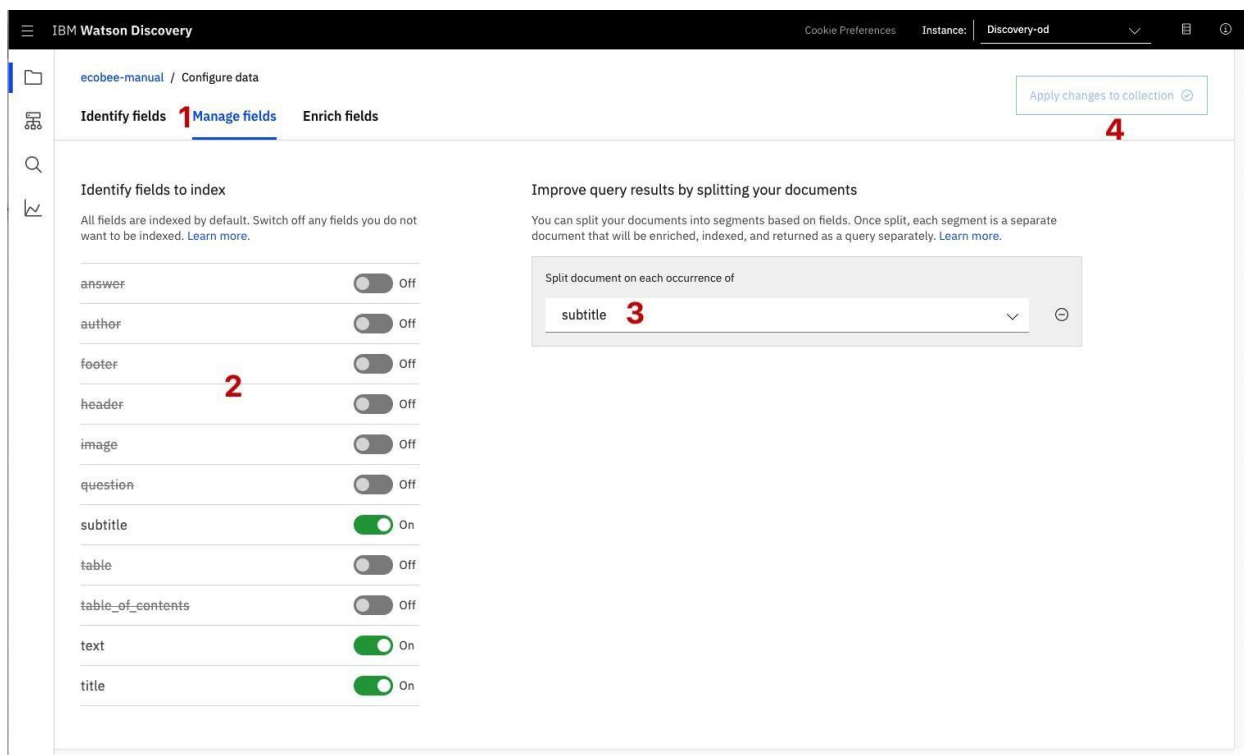
For this specific owner's manual, at a minimum, it is suggested to mark the following:

- 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 in a web browser. The browser's address bar displays the URL: `eu-gb.discovery.watson.cloud.ibm.com/regions/eu-gb/services/crn%3Av1%3Abluemix%3Apublic%3Adiscovery%3Aeu-gb%3Aa%2Fb72a2c064f7a471ea50a1d7f3b92e732...`. The page header includes the IBM Watson Discovery logo and the instance name "Discovery-pr". The left sidebar contains navigation icons for manual, overview, errors and warnings, and search settings. The main content area shows "37 documents" and "0 documents failed". It also displays "Identified 5 fields from your data" (footer, header, subtitle, text, title) and "Added 4 enrichments to your data" (Entity Extraction, Sentiment Analysis). A "Run" button is visible for the "Entities of type Location which have positive sentiment" enrichment. The Windows taskbar at the bottom shows the time as 11:37 PM on 6/13/2020.

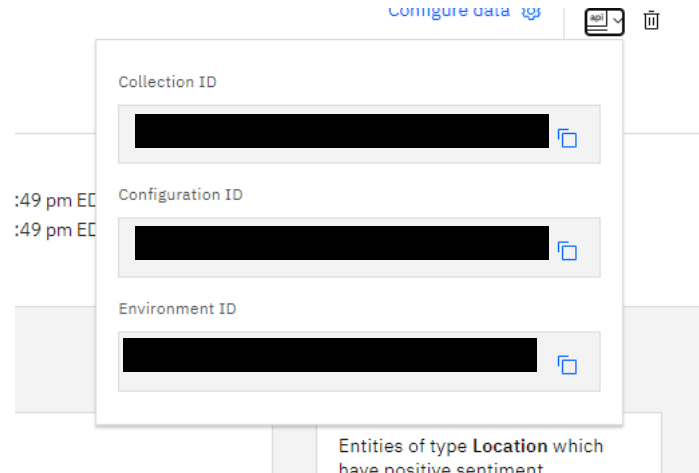
Return to the query panel (click Build your own query) and see how much better the results are.

The screenshot shows the IBM Watson Discovery "Build queries" panel. The left sidebar contains navigation icons for manual, build queries, and search settings. The main content area shows "Build a query using one or more of these components. Learn more." and "Use a sample query". The "Search for documents" section includes "Use natural language" and "Use the Discovery Query Language" options. The "best time to go to noth east" query is entered in the search box. Below the search box are two expandable sections: "Include analysis of your results" and "Filter which documents you query". At the bottom are "Run query" and "Close" buttons. The right sidebar shows the "Summary" tab with the "Query URL" `https://api.eu-gb.discovery.watson.cloud.ibm.com/in/...` and "Passages" of text. The Windows taskbar at the bottom shows the time as 11:37 PM on 6/13/2020.

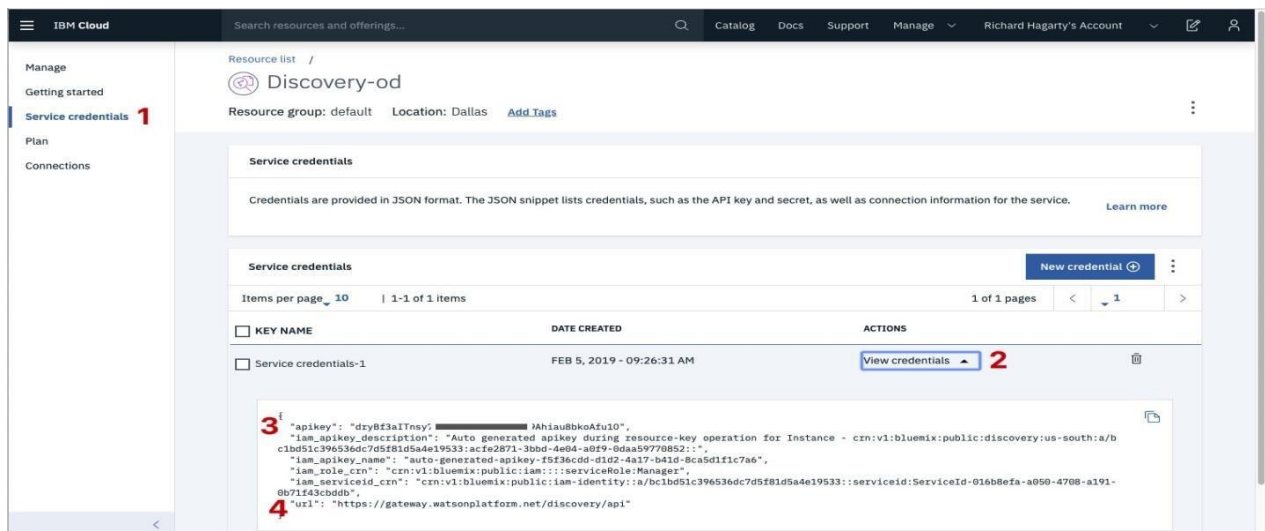
Store credentials for future use

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:



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

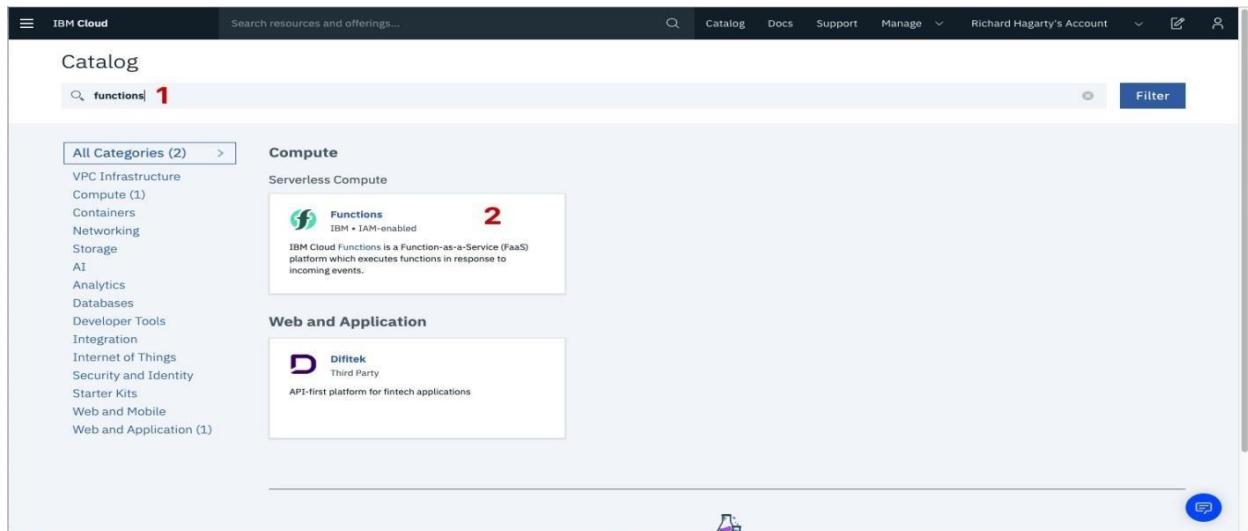


Click the View credentials [2] drop-down menu to view the IAM apikey [3] and URL

endpoint [4] for your service.

Creating IBM cloud functions:

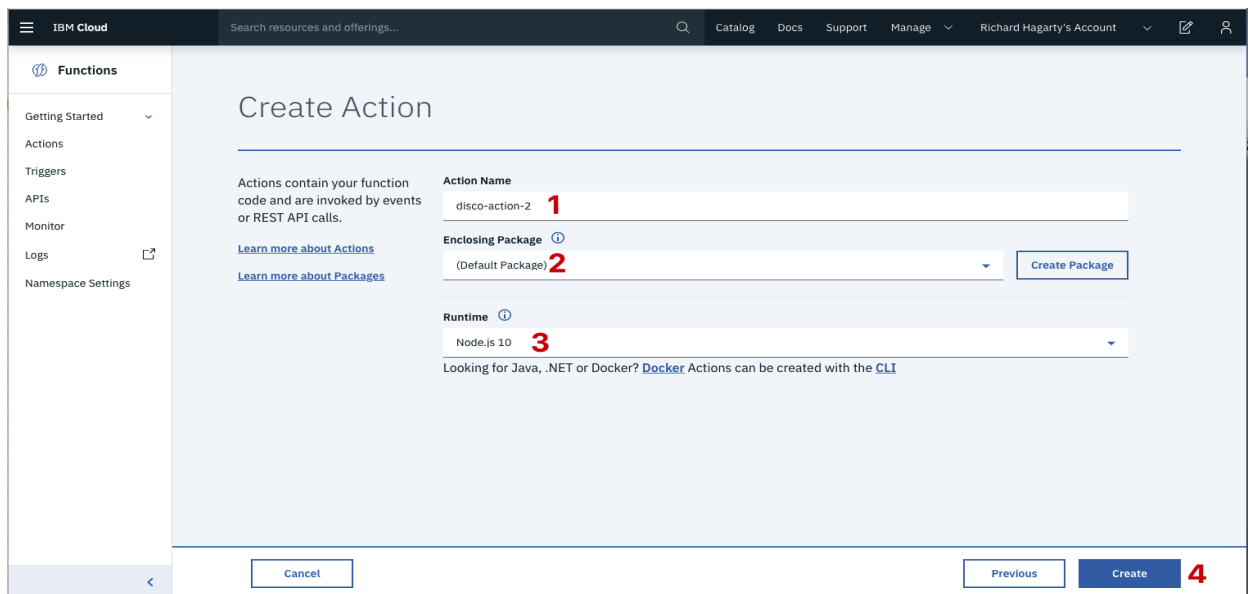
Now let's create the web action that will make queries against our Discovery collection. Start the IBM Cloud Functions service by selecting Create Resource from the IBM Cloud dashboard. Enter functions as the filter [1], then select the Functions card [2]:



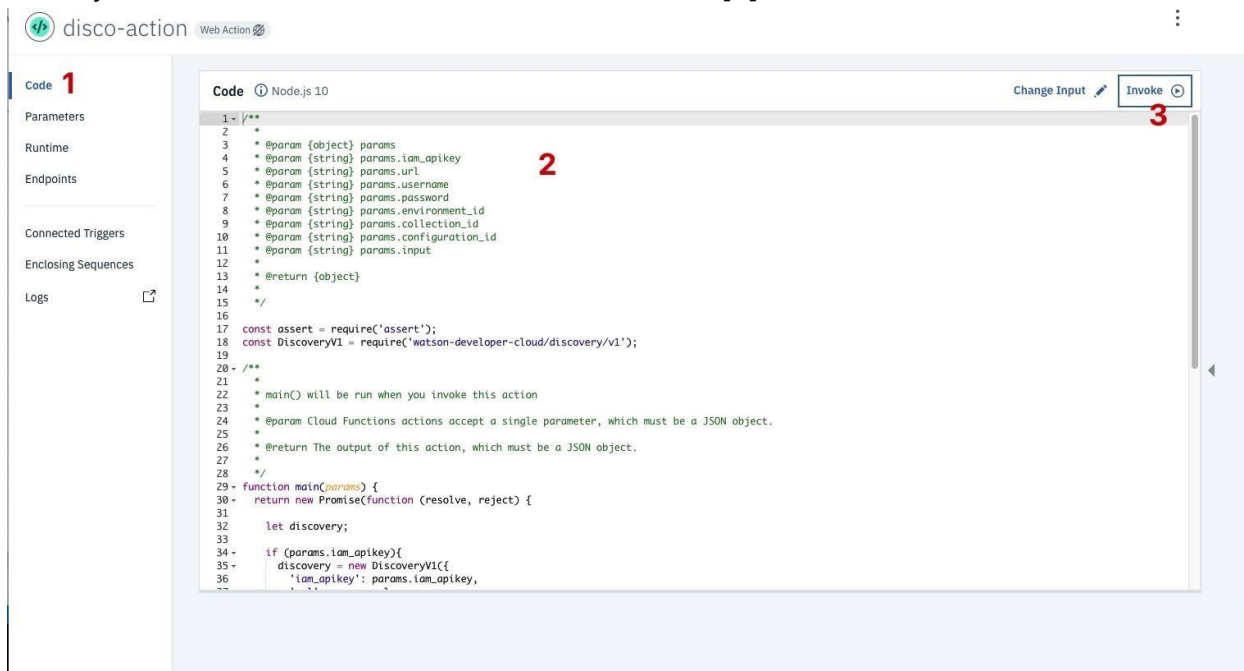
From the Functions main panel, click on the Actions tab. Then click on Create.

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.



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

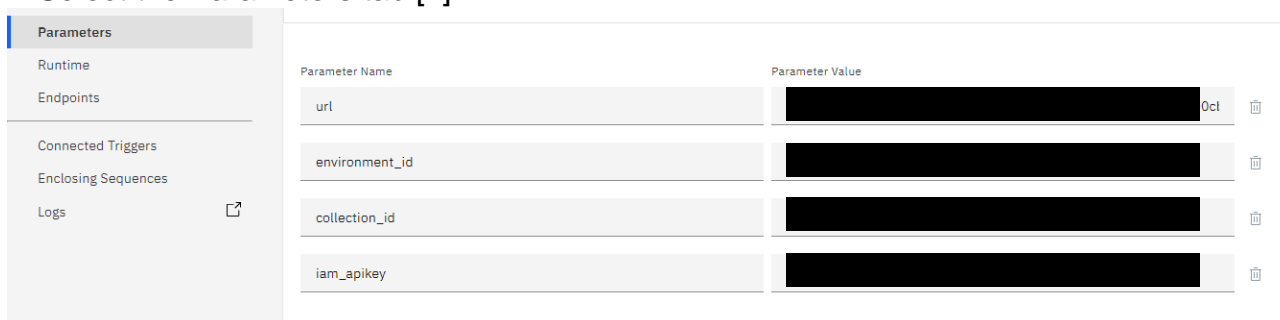


In the code editor window [2], cut and paste in the code from the disco-action.js file found in the actions directory of your local repo. The code is pretty straight-forward - it simply connects to the Discovery service, makes a query against the collection, then returns the response.

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]:



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.

Note: Make sure to enclose your values in double quotes.

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 IBM Cloud Functions console for the 'disco-action' function. The left sidebar contains a navigation menu with 'Code', 'Parameters', 'Runtime', 'Endpoints', 'Connected Triggers', 'Enclosing Sequences', and 'Logs'. The main area is divided into two panels. The left panel, titled 'Code', shows the Node.js code for the function, which uses the 'discovery' service. The right panel, titled 'Activations', shows the execution results for the function, including the activation ID, timestamp, and a JSON response containing matching results and enriched text.

```

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 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': '2019-03-25'
38       });
39     }
40
41     // ... (rest of the code)

```

The 'Activations' panel shows the following results:

```

Activation ID: e1bfc0ff21544e85bfc0ff21549e85a1
Results: {
  "matching_results": 14,
  "passages": [],
  "results": [
    {
      "enriched_text": {
        "categories": [
          {
            "label": "/technology and computing/operating systems",
            "score": 0.842265
          },
          {
            "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": 0.917306,
            "text": "iphone"
          },
          {
            "dbpedia_resource": "http://dbpedia.org/resource/Personal_digital_assistant",
            "relevance": 0.887088,
            "text": "Personal digital assistant"
          }
        ]
      }
    }
  ]
}

```

The screenshot shows the IBM Cloud Functions console for the 'discovery-function' function. The left sidebar contains a navigation menu with 'Code', 'Parameters', 'Runtime', 'Endpoints', 'Connected Triggers', 'Enclosing Sequences', and 'Logs'. The main area is divided into two panels. The left panel, titled 'Web Action', shows the configuration for the function, including the 'Enable as Web Action' checkbox, the 'Raw HTTP handling' checkbox, and the 'HTTP METHOD' and 'AUTH' settings. The right panel, titled 'REST API', shows the configuration for the function, including the 'HTTP METHOD' and 'AUTH' settings, and the 'CURL' command.

The 'Web Action' panel shows the following configuration:

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

The 'REST API' panel shows the following configuration:

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

The 'CURL' panel shows the following command:

```

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 provided curl command [4]. If it fails, re-check your parameter values.

NOTE: An IBM Cloud Functions service will not show up in your dashboard resource list.

To return to your defined Action, you will need to access Cloud Functions by selecting Create Resource from the main dashboard panel (as shown at the beginning of this step).

Configure Watson Assistant:

As shown below, 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 #Information, and at a minimum, enter the following example questions to be associated with it.

The screenshot shows the Watson Assistant configuration interface for a new intent named "#Information". At the top, there's a breadcrumb navigation with a back arrow and the text "#Information". To the right, it says "Last updated: a day ago" and includes icons for download, delete, search, and a "Try it" button. Below this is a section titled "User example" with the instruction "Add unique examples of what the user might say. (Pro tip: Add at least 5 unique examples to help Watson understand)". There's a text input field with the placeholder "Type a user example here, e.g. I want to pay my credit card bill". Below the input field are two buttons: "Add example" and "Show recommendations". A toggle switch for "Annotate entities" is set to "Off", with a link "What's this?". Below this is a table of user examples. The table has two columns: "User examples (22) ↑" and "Added ↑↓". The first three rows are visible, each with a checkbox, a text example, and a date. At the bottom, it says "Showing 1-22 of 22 examples" and "1 of 1 pages".

User examples (22) ↑	Added ↑↓
<input type="checkbox"/> events in august-september	a day ago
<input type="checkbox"/> events in december-january	a day ago
<input type="checkbox"/> events in february march	a day 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.

The screenshot displays the IBM Watson Assistant interface. At the top, a dark blue header reads "IBM Watson Assistant". Below it, a breadcrumb trail shows "Skills /". The main heading is "Customer Care Sample Skill copy" with a subtitle "Sample simple customer service skill to get you started." A navigation bar contains tabs: "Intents", "Entities", "1 Dialog" (highlighted with a red '1'), "Analytics", "Options", "Versions", and "Content Catalog". The main area lists five dialog nodes: "Directions and location", "Make an appointment", "Transfer to agent", "Small Talk", and "anything_else". The "Small Talk" node is selected, and its context menu is open, showing options: "Add node to folder", "Add node above", "Add node below" (highlighted with a blue bar and a red '3'), "Add folder", "Move", "Duplicate", "Jump to", and "Delete". A red '2' points to the three-dot menu icon for the "Small Talk" node.

IBM Watson Assistant

Skills /

Customer Care Sample Skill copy
Sample simple customer service skill to get you started.

Intents Entities **1** Dialog Analytics Options Versions Content Catalog

Directions and location
#Customer_Care_Store_Location
3 Responses / 0 Context Set / Skip user input / Returns

Make an appointment
#Customer_Care_Appointments
3 Responses / 7 Context Set / 5 Slots / Does not return

Transfer to agent
#General_Connect_to_Agent
1 Responses / 0 Context Set / Does not return

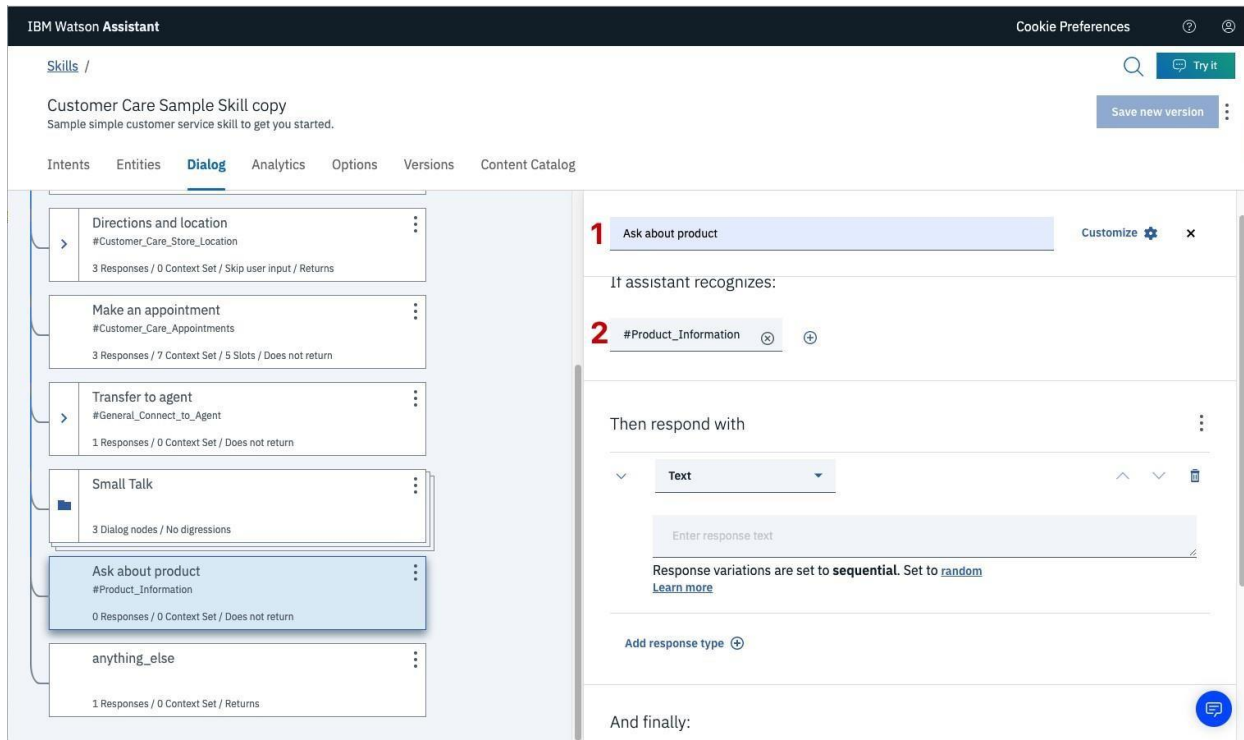
Small Talk
3 Dialog nodes / No digressions

anything_else
1 Responses / 0 Context Set / Returns

Add node to folder
Add node above
Add node below **3**
Add folder
Move
Duplicate
Jump to
Delete

2

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

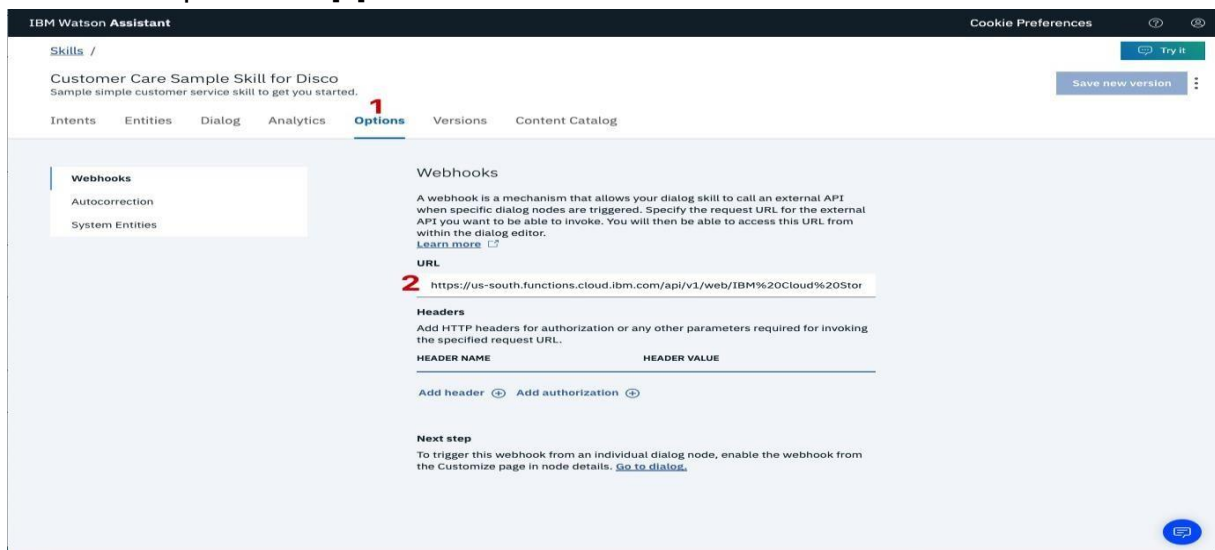


This means that if Watson Assistant recognizes a user input such as "how do I set the time?", it will direct the conversation to this node.

Enable webhook from Assistant

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

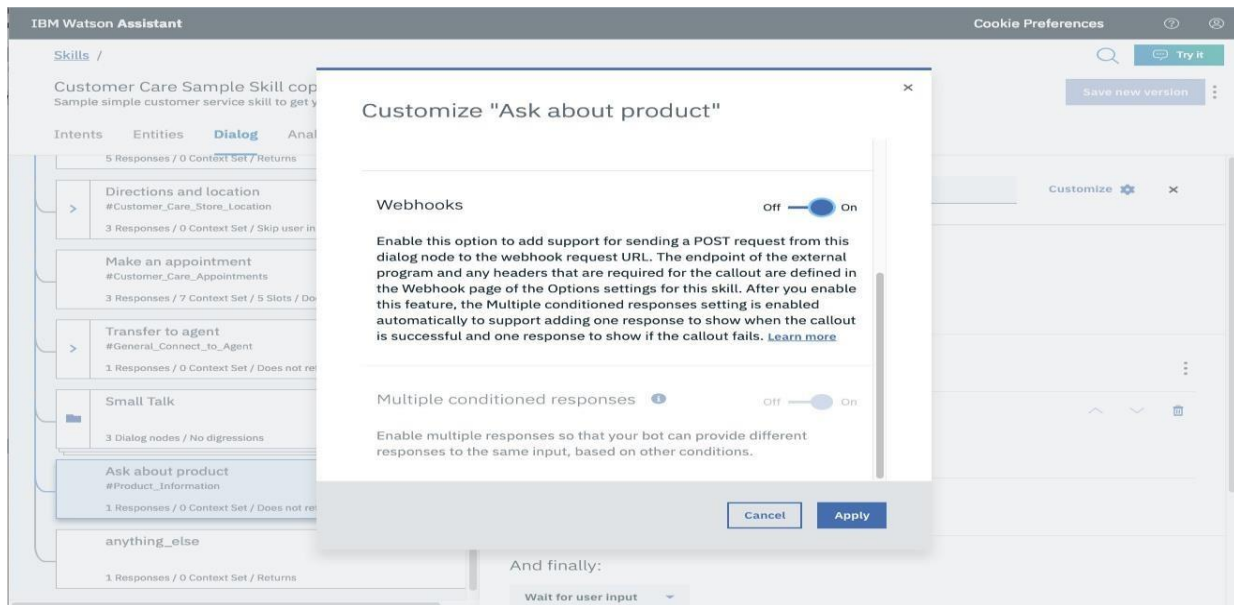
Select the Options tab [1]:



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

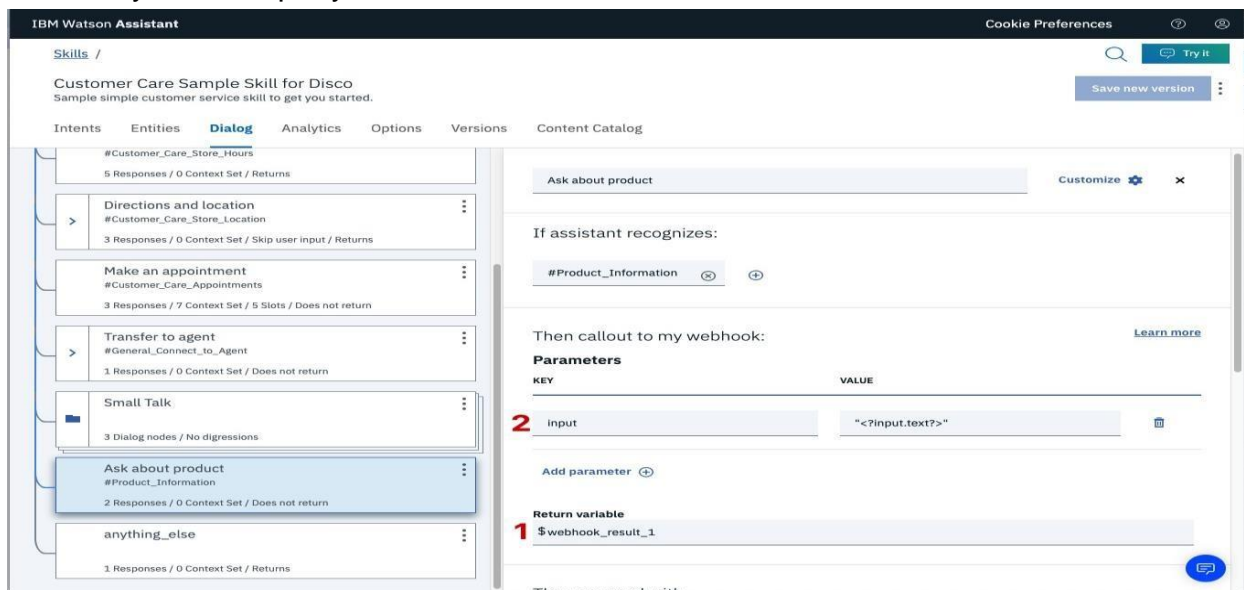
Important: Add .json to the end of the URL to specify the result should be in JSON format.

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.



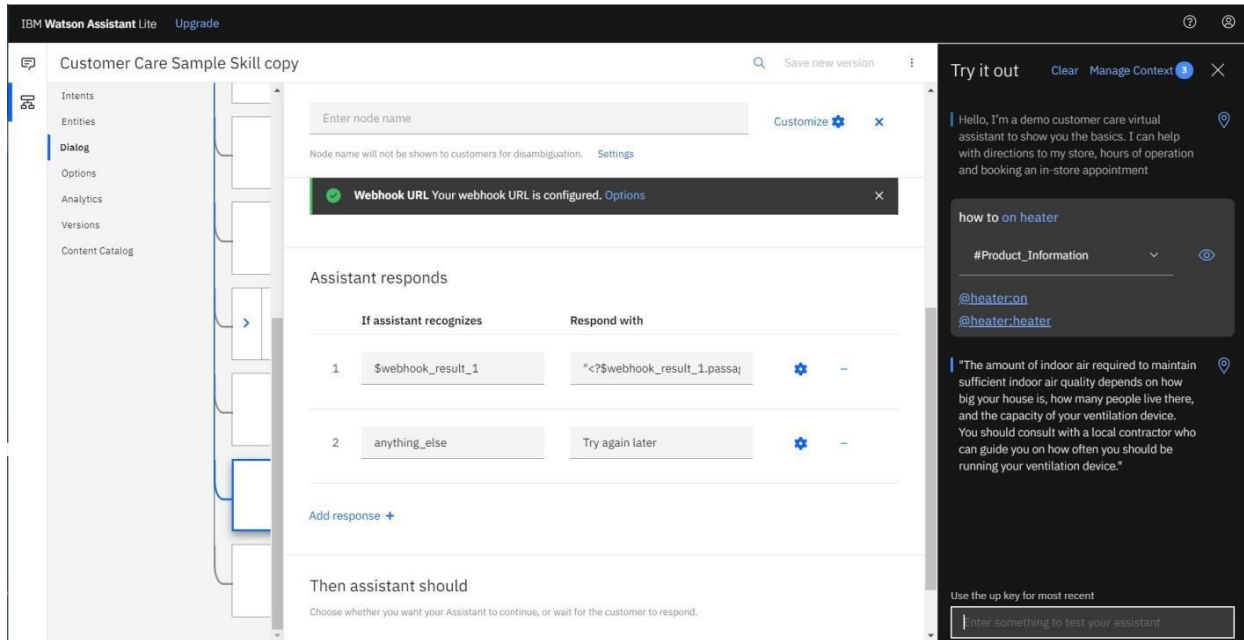
You will also need to pass in the users question via the parameter input [2]. The key needs to be set to the value:

"<?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:

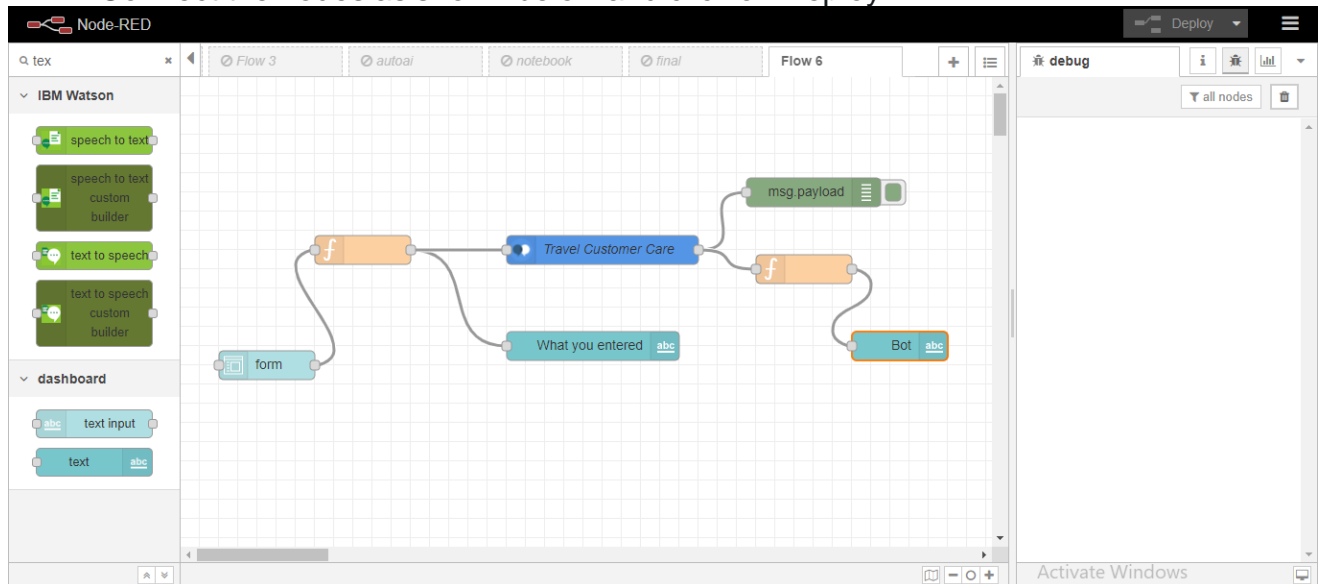
Add Add "<?\$webhook_result_1.passages[0].passage_text?>" in respond with in Assistant responds block as shown below.



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
- Connect the nodes as shown below and click on Deploy



- Re-inject the flow and observe the parsed output

We are done integrating Watson assistant service to Node-red. In the next lab, we will create a web application using Node-red for the chatbot. 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

Edit function node

Delete
Cancel
Done

⚙ Properties

Name

Function

```

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

```

🔗 Outputs

1

info

Information

Node"54b3776f.053068"
Nameinput parsing
Typefunction
show more

Description

Node Help

A JavaScript function block to run against the messages being received by the node.

The messages are passed in as a JavaScript object called `msg`.

By convention it will have a `msg.payload` property containing the body of the message

The function is expected to return a message object (or multiple message objects), but can choose to return nothing in order to halt a flow.

Details

See the [online documentation](#) for more

Show the Info tab with `ctrl-g` `i` or

the Debug tab with `ctrl-g` `d`

- 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

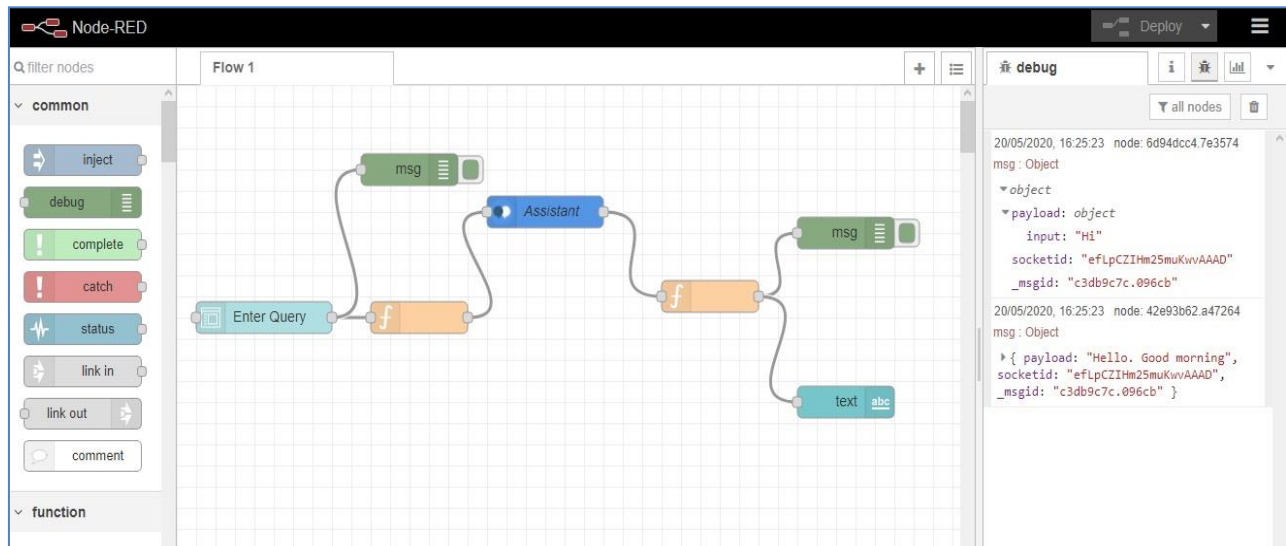
5.

FLOWCHART

1. Create flow and configure node:

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



6.

RESULTS

Finally our Node-RED dash board integrates all the components and displayed in the Dashboard UI by typing URL- <https://node-red-qxgkl.eu-gb.mybluemix.net/ui/#!/0?socketid=SZQ2GmhqOpXvnH-qAAAF> in browser

Home

Customer Help

Enter Query

Enter *

Hi

SUBMIT CANCEL

Output

Hello. Good morning

7. ADVANTAGES AND 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.

8.

APPLICATIONS

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

9.

CONCLUSION

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

10.

FUTURE SCOPE

We can include Watson studio text to speech and speech to text services to access the chat-bot hands-free. This is one of the future scope of this project.

11.

BIBILOGRAPHY

APPENDIX

Source code:

```
[{"id":"60ce372a.beadc8","type":"ui_text","z":"70581e6d.2a424","group":"44138696.337b18",
"order":3,"width":10,"height":4,"name":"","label":"Bot","format":"{{msg.payload}}","layout":"col
-
center","x":750,"y":280,"wires":[]},{id:"44138696.337b18","type":"ui_group","z":"","name":"C
HATBOT","tab":"52e10654.c5b3d8","order":1,"disp":true,"width":10,"collapse":false},{id:"52
e10654.c5b3d8","type":"ui_tab","z":"","name":"travel
chatbot","icon":"dashboard","disabled":false,"hidden":false}]
```

Cloud function **Node.js 10** code for discovery integration webhook generation:

```
/**
 *
 * @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);
    });
  });
}

```

}

