Table of Contents

1. Overview

The Open Digital Lab provides a safe space for inter-company collaborative proof of concept generation leveraging open standards. With the pace of change in the industry continuing to accelerate, and partnerships with other companies to deliver new services and solutions increasing in importance, there has never been a greater need for an environment for collaborative co-creation of innovative new solutions and services. The TM Forum Open Digital lab hosted by IBM provides this environment where inter-company proof of concepts can be rapidly developed, tested and iterated.

The Open Digital Lab is a member benefit for the 900-member companies who are joined TM Forum. To access the lab, first sign up for a free IBM Cloud account and connect to the TM Forum space where a feast of tools available for rapid innovation. These include but are not limited to:

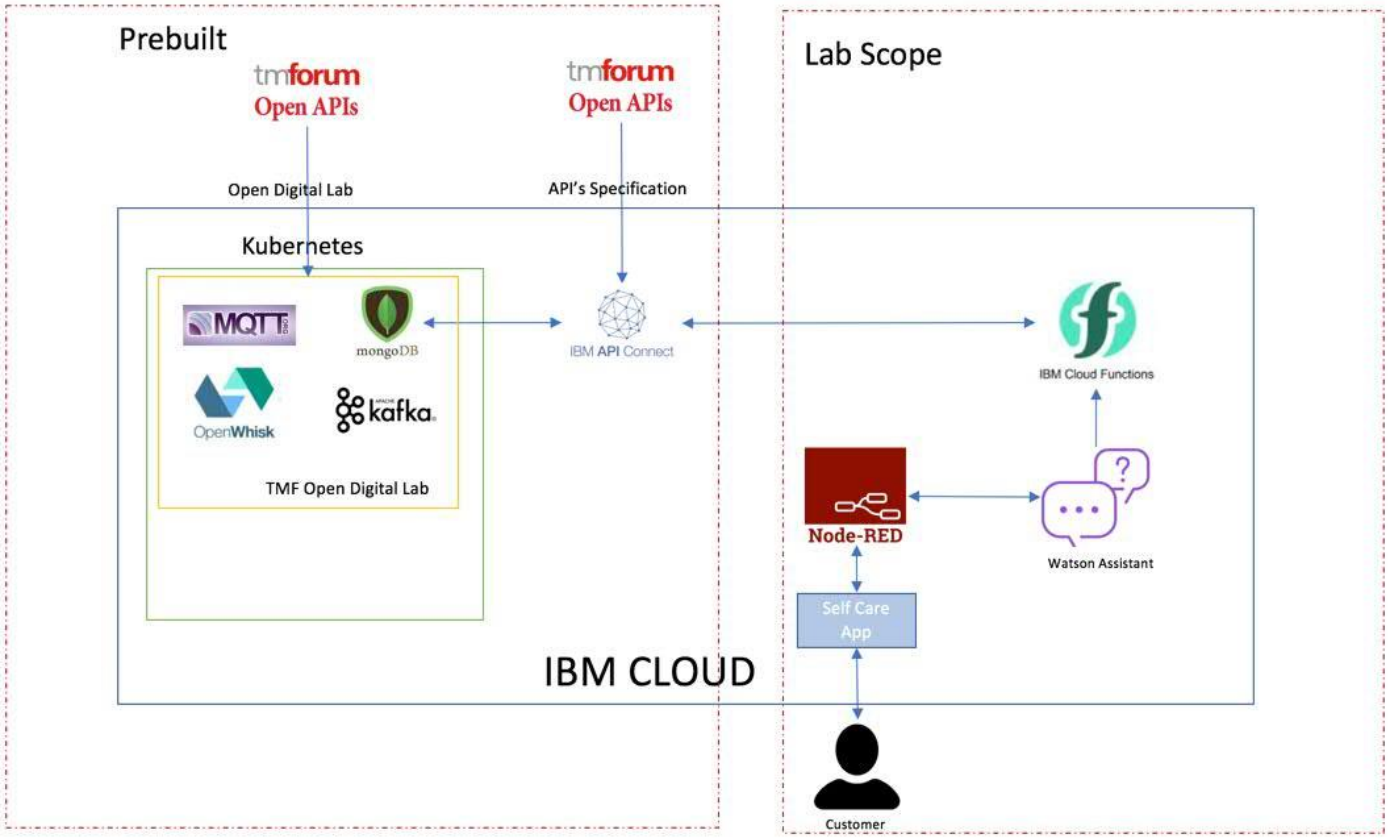
- Live reference implementations of TM Forum's Open APIs
- Easier access to rapid testing of in development or deployed TMF Open APIs
- Access a series of open source technologies
- In a safe place for co-creation and iteration

The first wave of innovative projects to use the Open Digital lab are the proof of concept catalyst projects being developed for Digital Transformation World 2018 & 2019.

2. Scope

The below architecture shows the scope for self-care application. The container which is hosted on IBM Cloud consists of node-red, mongodB, MQTT, kafka and openwhisk. TM Forum APIs are replicated using IBM API Connect and these APIs are invoked using IBM Functions (openwhisk). The scope of this lab is to create a sample self-care application which invokes TM Forum APIs in a web chat application using node-red, openwhisk and Watson Assistant(Conversation). So, the lab scope includes creation of IBM Functions to utilize pre-built TM Forum APIs and then create a Watson Assistant service (Conversation) which calls the IBM Functions to get the results. Finally, to create a node-red application from the pre-built flow to link web chat application, Watson Assistant and openwhisk and provide response to customers.

3. Architecture



4. TMF Open API Program

4.1. *Introduction*

OpenAPI is a specification for describing REST APIs. With OpenAPI, instead of XML, you have set of JSON objects, with a specific schema that defines their naming, order, and contents. This JSON file (often expressed in YAML instead of JSON) describes each part of your API. By describing your API in a standard format, publishing tools can programmatically ingest the information about your API and display each component in a stylized, interactive display.

4.2. API's Specification

Basically, a swagger file describes your entire API, including:

- Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.

As an integral part of our lab, we were using three TMF APIs from TMF OPEN API TABLE. The suite of APIs we were using for our use case are:

- Account Management API (TMF 666)
- Usage Management API (TMF 635)
- Usage Consumption Management API (TMF 677).

Every API of the open API table has links which would give API specifications, endpoints, schema definitions in the form of PDFs.

4.3. Customer Data

The dataset that was used for our billing use case is in accordance with the schema definitions of the TMF APIs. The APIs used for our use case as discussed above has various paths and schema definitions. BillingAccount, Usage, UsageConsumptionReport datasets were used based on the flow we built for our use case. So, sample data with appropriate values is generated using the schema to test the use case. The sample data is available on the project github repository.

4.4. Deployment

4.4.1. TMF Open Digital Lab (Developer sandbox)

As a part of our lab, sandbox that contains an environment which has all the tools needed for the lab has been deployed on kubernetes cluster on IBM Cloud. Services include Node.js application, node-red app, mongodb, openwhisk (IBM Functions), mqtt, kafka which requires zookeeper. The lab will require to push some components to a public or private repository. For simplicity we are using docker hub. Most of the components used are pre-built images on docker hub, but some require images to be built using **docker cli.**

Mongodb, a document-based database, is deployed on the **kubernetes cluster** using an official mongo image. Persistent volumes is used for mongodb storage within the cluster such that when a pod restarts, we don't lose the data with it.

4.4.2. IBM API Connect

IBM API Connect integrates IBM API Management and IBM **Strong Loop** into a single package with built-in gateway, which allows you to create, secure, run and manage APIs and Microservices. IBM API Connect utilizes Strong Loop capabilities to rapidly build API's and Microservices using Node.js – Loopback and Express frameworks. It uses Model driven approach to create API's, map models to back-end systems using available connectors.

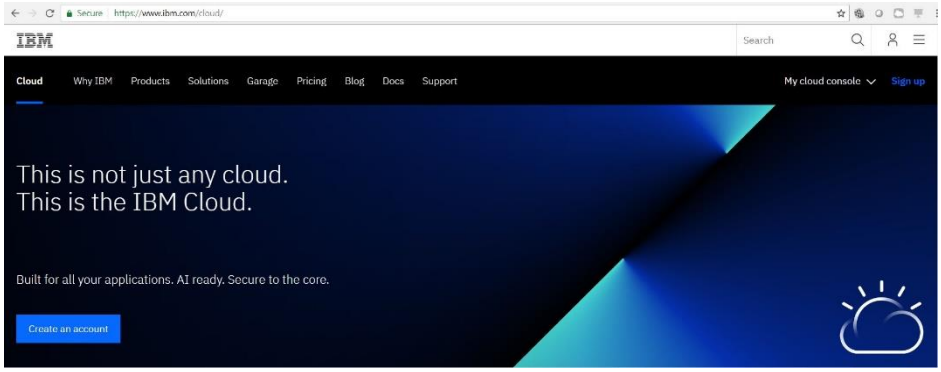
For this lab, we took the swagger files from TMF OPEN API TABLE and using the definitions and paths defined in the swagger files, we were able to generate persistent models using API Connect (apic command line). These models generated stores and retrieves information from MongoDB. Once the models are generated and linked to database, we tested the models and were able to push it to **IBM API Connect** on **IBM Cloud**. Thus, creating APIs which can be accessed anywhere. The deployment of TMF APIs is explained in the API Connect Hands-on section of appendix.

5. Environment Setup

5.1. Creating an IBM Cloud Account

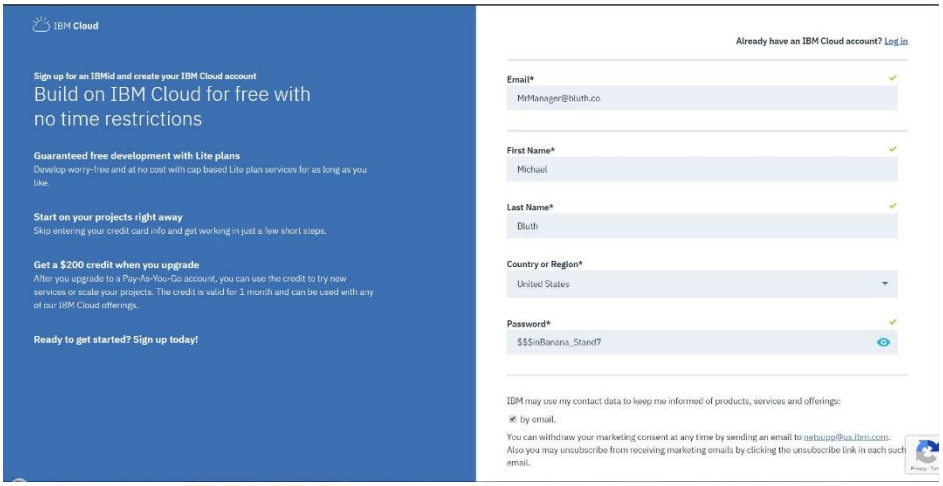
- Step (1)

Navigate to www.ibm.com/cloud, Click the Blue "Create Account" Button



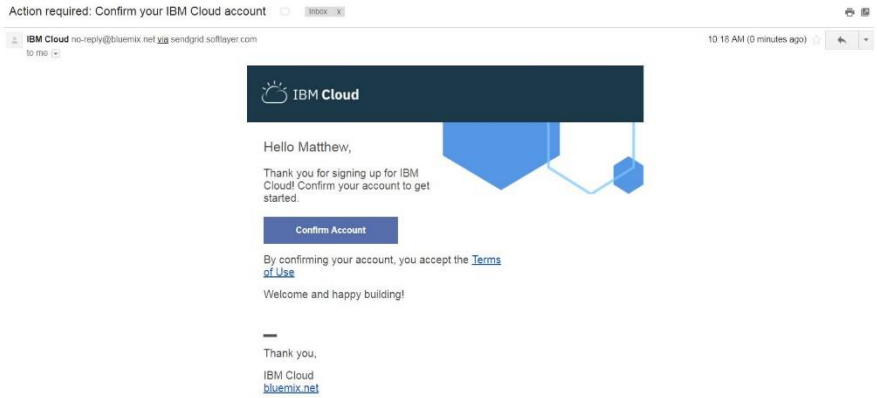
- Step (2)

Create your account details and password

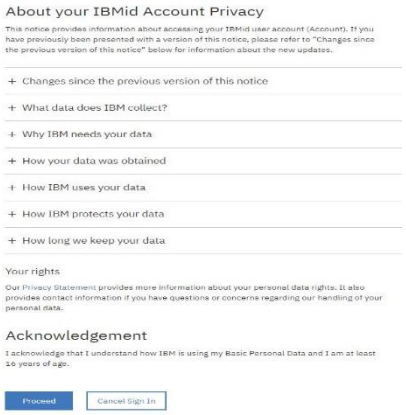


- Step (3)

You will receive an Account Confirmation Email at the address you provided. Click the blue “Confirm Account” button within the email



You will immediately be provided with a copy of the IBMid Privacy Policy. Read the policy, then click “Proceed”



Write down your organization information:

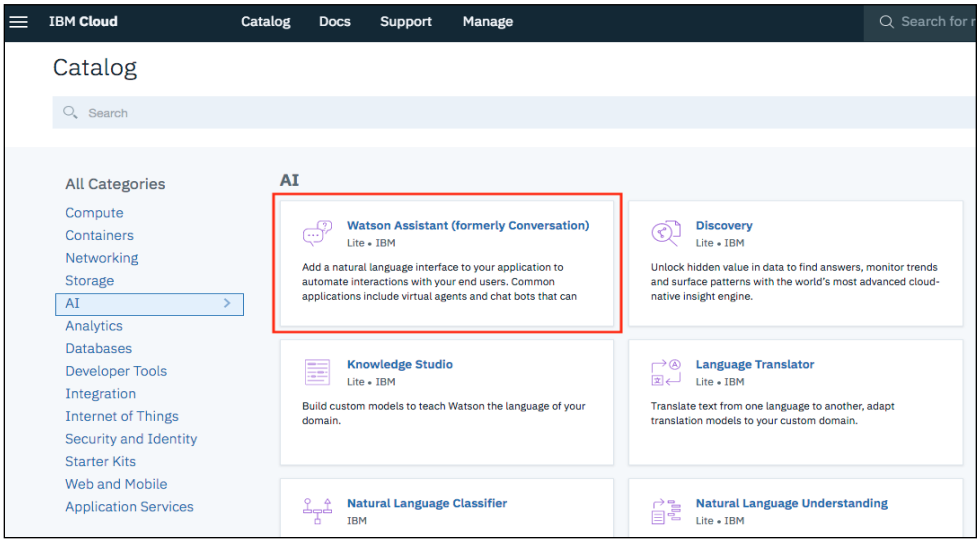
Organization:	
Space:	

5.2. Creating a Watson Assistant Service

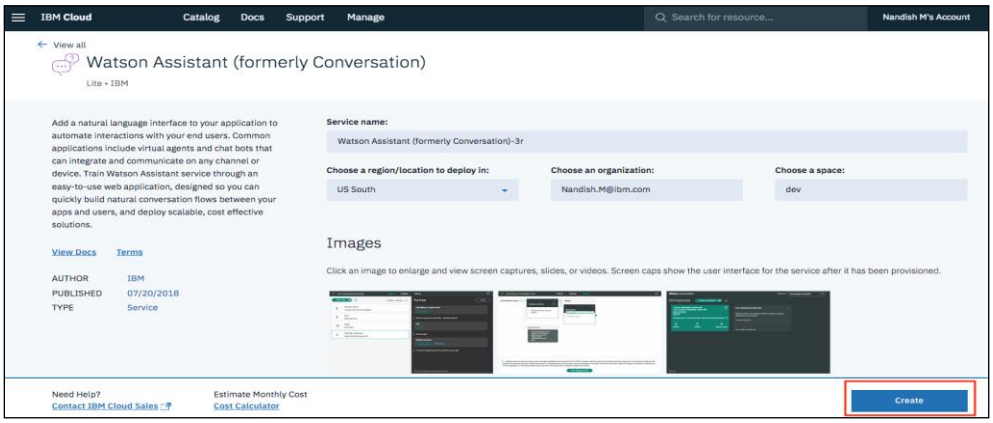
- Create Watson Assistant service

  Navigate to 'Catalog' section in IBM Cloud.

  Select 'Watson Assistant' in 'AI' Category (as shown in image below).

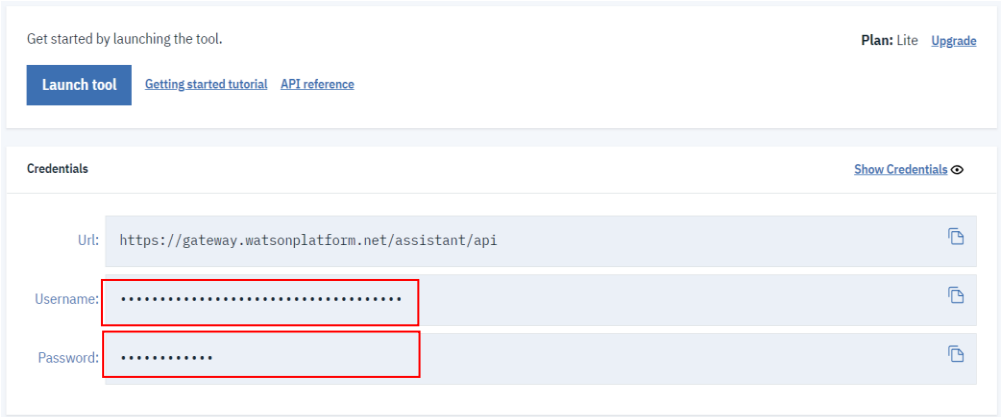


  Click 'Create'



- Create a workspace:

  Once Watson assistant service is created, you will be able to see this screen.

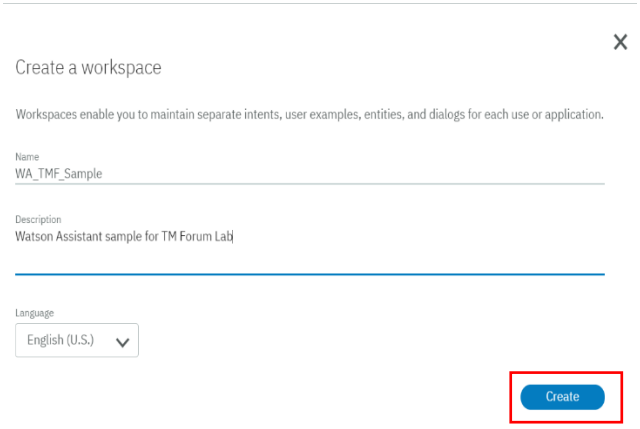


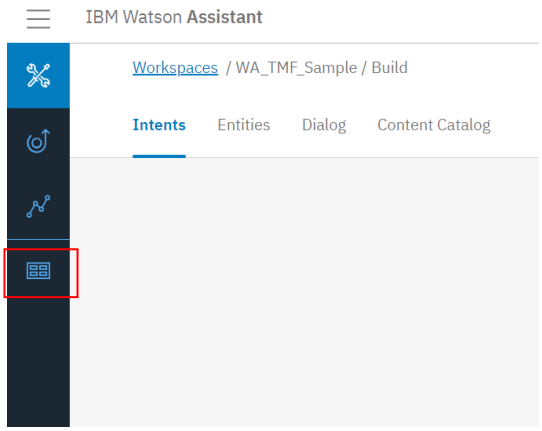
Click “Launch tool” and workspaces tab to create new workspaces. Click “Create” to create a workspace and provide name and description as below.

Name: WA_TMF_Sample

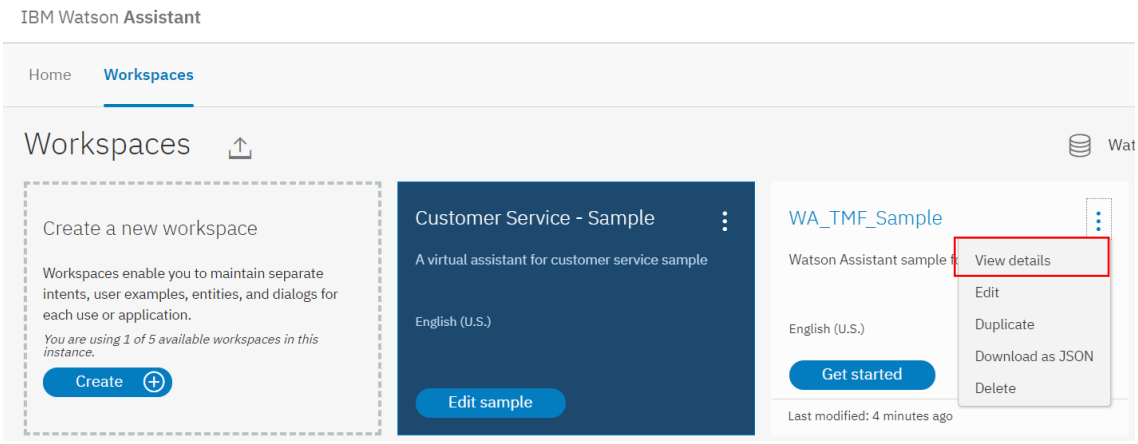
Description: Watson Assistant sample for TM Forum Lab

After the workspace is created, click “Back to workspaces” from the menu on left side.





Click “View details” of your workspace to see the “Workspace_ID”



- <u>Get the Service Credentials:</u>

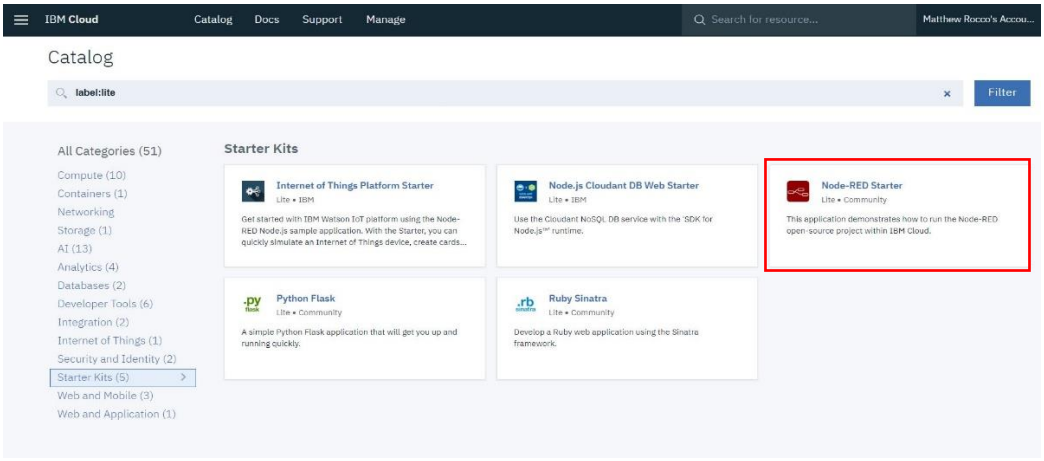
Please make a note of username and password of the Watson assistant service and workspace id of the workspace created.

Username:	
Password:	
Workspace ID:	

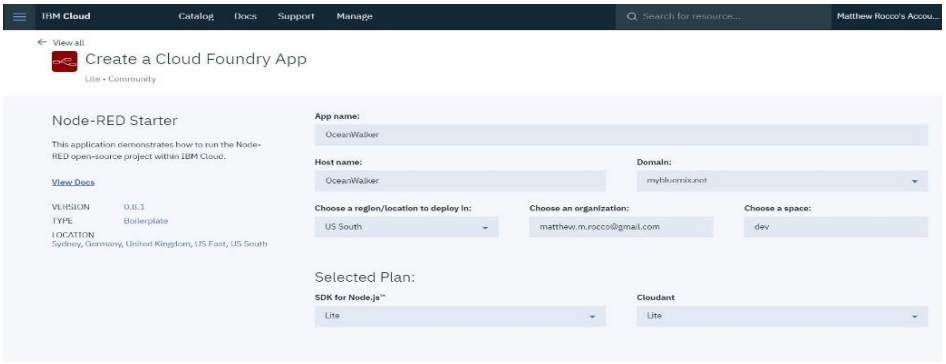
5.3. Instantiate a Node-red starter

- Step (1)
  Within the IBM Cloud Dashboard, navigate to the Catalog using the “Catalog” Button in the upper toolbar

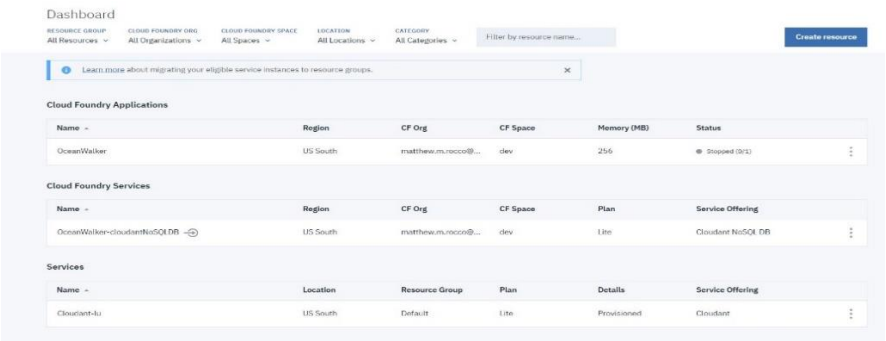
- Step (2)
  Once in the Catalog, Search for “**Node-RED Starter**” and click the Description Card. Alternatively, click the “**Starter Kits**” Category in the Left-hand filter menu as shown.



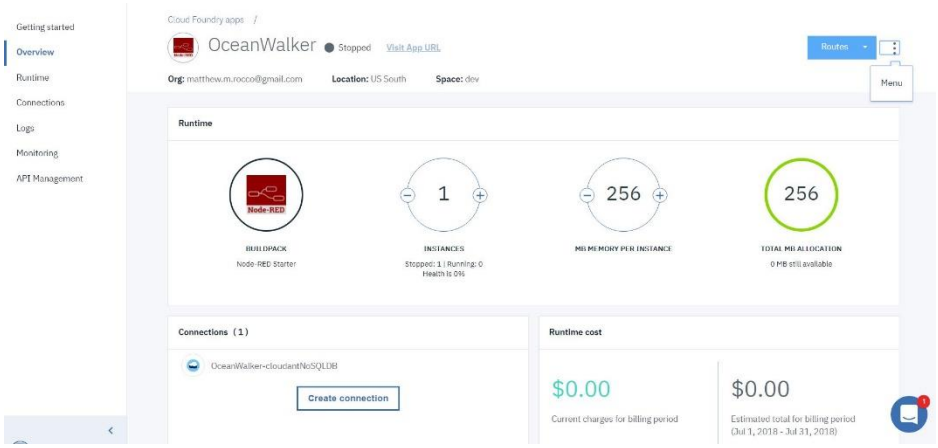
- Step (3)
  Give your Node-RED Service a unique App Name and then Create the Service.



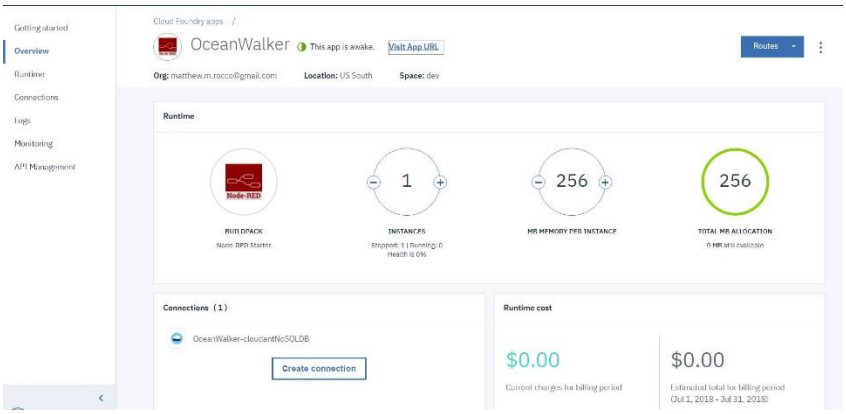
You will then be redirected to the IBM Cloud Dashboard. Your Node-RED Service will appear under the “Cloud Foundry Applications” Header. Click It.



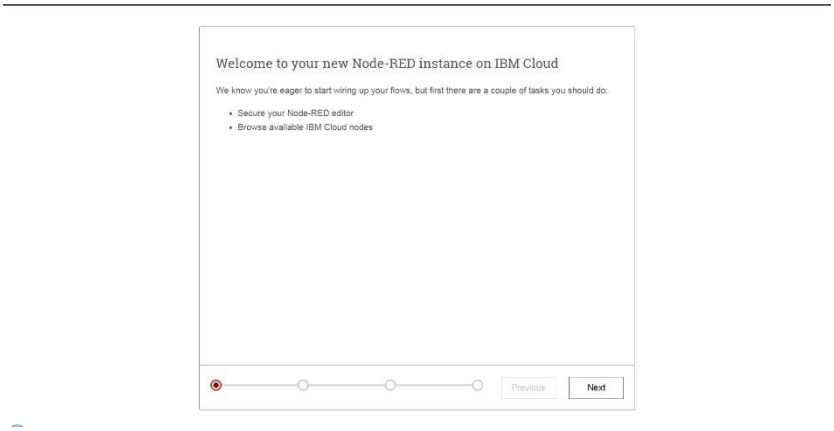
- Step (4)
  In the Node-RED Service Menu, navigate to the “Overview” Tab in the Left-hand menu, then select the “Menu” in the upper-right, as shown. Start the Service.



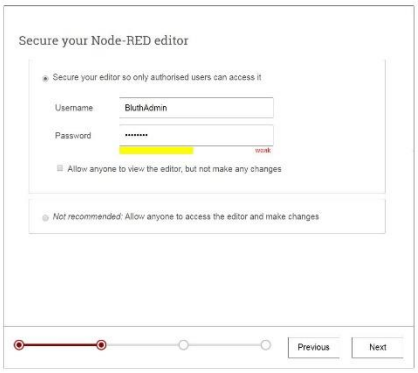
Wait roughly One Minute. The page should refresh and show a “This app is awake” status. Refresh the page manually, if necessary. Once the Status message is displayed, click the “Visit App URL” link, as shown



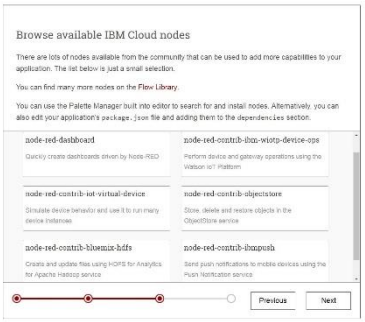
- Step (5)
  You will be taken to a Node-RED Startup Wizard. Click “Next”



Choose to “Secure your Node-RED editor” using the Radio Button. Create a Username and Password. Leave “Allow anyone to view the editor” unchecked. Then Click “Next.”



Click “Next” again.



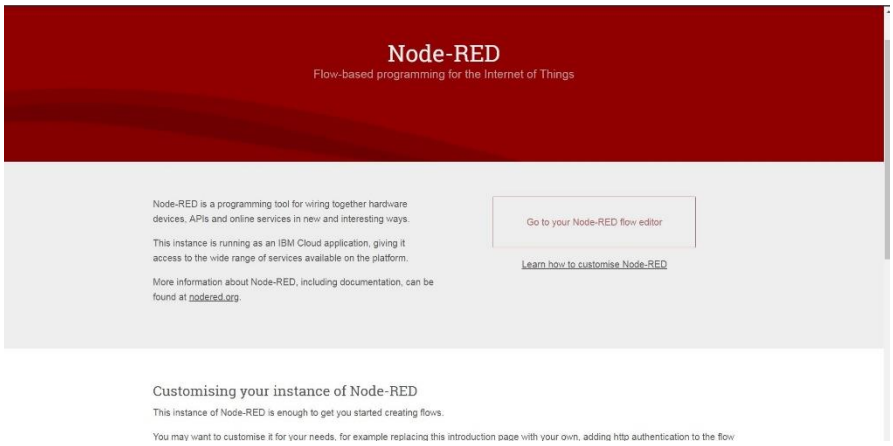
Finish the Install by Clicking “Finish”

Write down your credentials:

USERNAME:	
PASSWORD:	

- <u>Step (6)</u>

  Your Node-RED Instance is now live. You can begin working with the Flow Editor at any time by clicking the “Go to Your Node-RED flow editor” button in the center of the page.



You will then be brought to the Flow Editor, as shown below

6. Self-Care App Lab

6.1. Creating and configuring the IBM Functions Service

6.1.1. What is IBM Functions

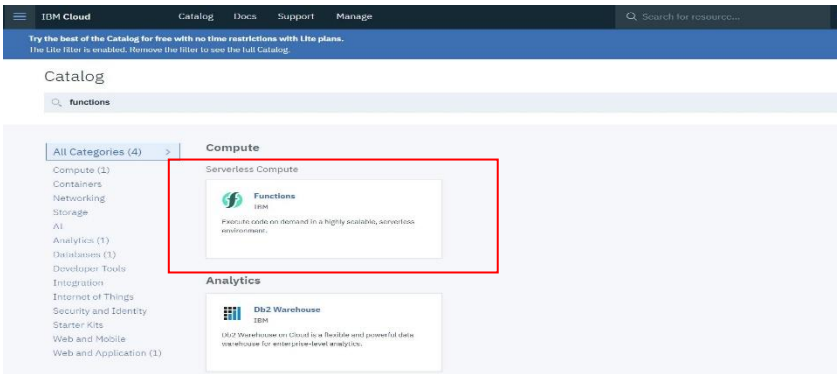
IBM Cloud Functions (based on Apache OpenWhisk) is a Function-as-a-Service (FaaS) platform which executes functions in response to incoming events. Given Cloud Functions' event-driven nature, it offers several benefits for user-facing applications, whereas the HTTP requests coming from the user's browser serve as the events. Cloud Functions applications use compute capacity and billed only when they are serving user requests. Idle standby or waiting mode is nonexistent. This feature makes Cloud Functions considerably less expensive when compared to traditional containers or Cloudfoundry applications. Both of which can spend most of their time idle, waiting for inbound user requests, and being billed for all that "sleeping" time.

IBM has a growing list of runtime execution options, which now includes, Java, Swift, Node, Python, PHP and other languages of your choice using Docker.

6.1.2. Creating IBM Functions:

- Step (1)
  Navigate to catalog.mybluemix.net to reach the IBM Cloud Catalog.
  Search "**Functions**" to find IBM Functions Service.

  ****Note: Please make sure that the actions being created on IBM Functions as a part of this lab are on the same cloudfoundry organization and same space as the Watson assistant service created above. This way Watson assistant and IBM Functions can call each other.**



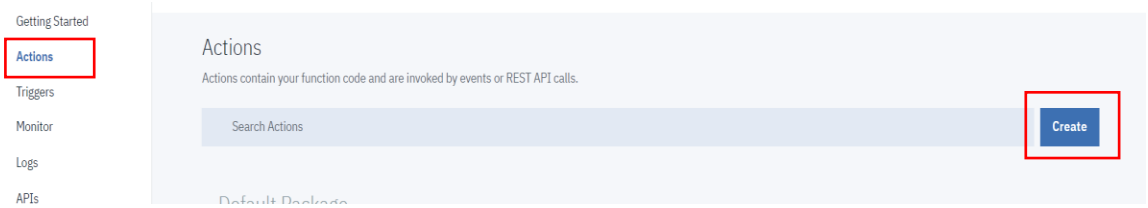
- Step (2)
  Navigate to "API Key" of Getting started menu, look for the API Key section on the right having Current namespace, host and key. The hidden key (click the show key to make it visible) here has the credentials of

IBM Functions in the form of "**username:password**". Write down these credentials as these are required in Watson Assistant to call IBM Functions.

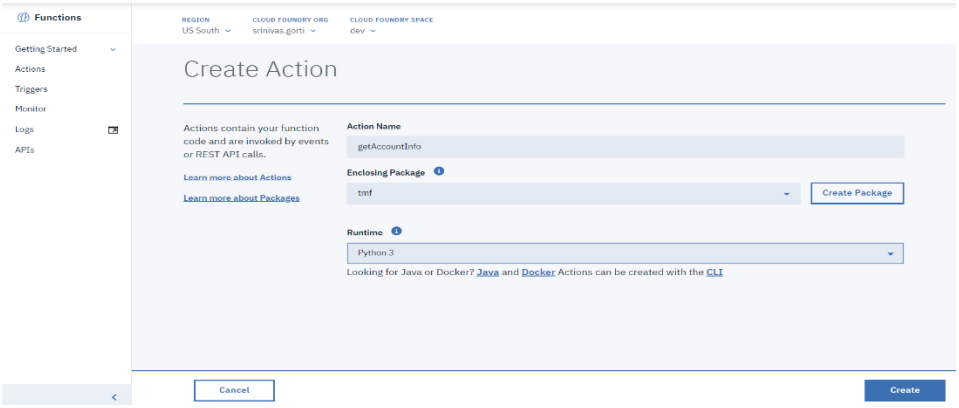
NAMESPACE:	
USERNAME:	
PASSWORD:	

- Step (3)

Navigate to "Actions" in the Left-hand menu. Click the Blue "**Create**" Button on the right-hand side and click "Create Action" to create a new action.

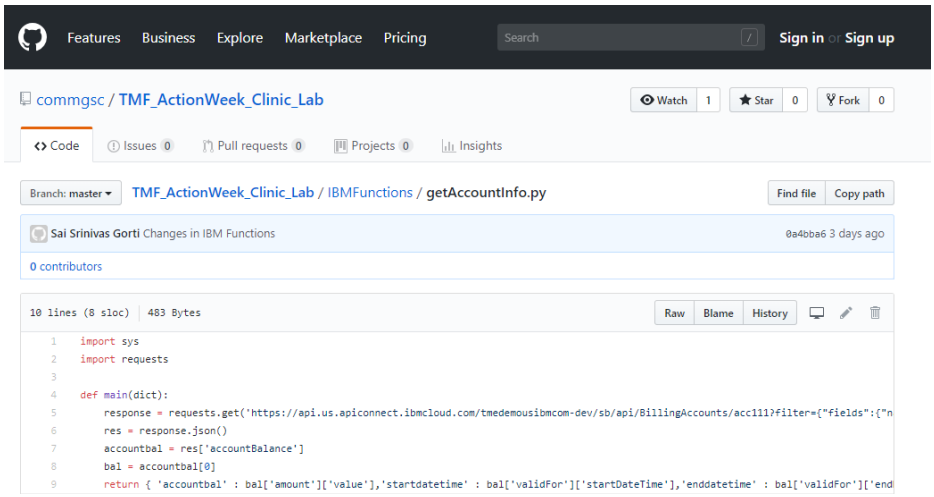


Enter "**getAccountInfo**" as the Action Name. Use the "**Create Package**" button to create a Package. Name it "**tmf**". Select Python 3 as your Runtime in the Dropdown Menu. Then click the blue "**Create**" Button in the lower right corner.



- Step (4)

In a separate browser tab, navigate to: Within the Git repository, navigate to TMF_ActionWeek_Clinic_Lab/IBMFunctions/getAccountInfo.py. Copy the code from "getAccountInfo.py"



Back in your IBM Functions Tab, paste the code from the Git Repository in the “code” editor in the center of the page.

- Step (5)

  Validating created IBM Function:
  Back on the main page, Click the “Invoke” Button above the code editor in the center of the page. You can see the results of your test in the “Activations” Window shown at right in the image below:

We will be able to see results as shown above If the function is executed properly without any errors.

- Step (6)

  Please create and validate for other functions available in the github folder using above steps. The other functions available are **getBillingEvents.py, getUsageReport.py** which can be obtained by navigating within the git repository to the following functions.

  - TMF_ActionWeek_Clinic_Lab/IBMFunctions/getBillingEvents.py
  - TMF_ActionWeek_Clinic_Lab/IBMFunctions/getUsageReport.py

6.2. Create a sample conversation using Watson Assistant

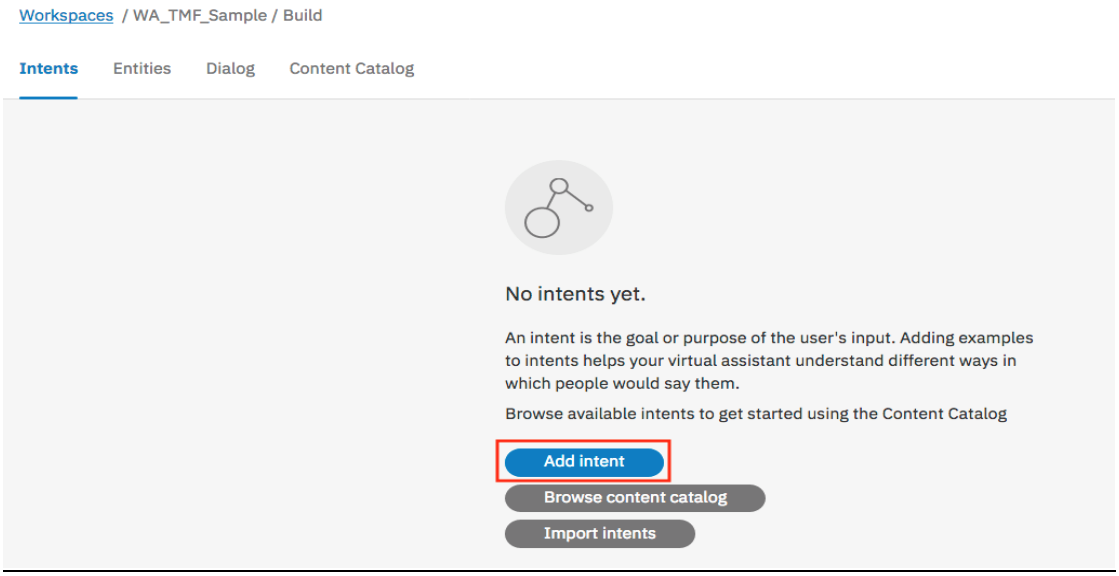
6.2.1. What is IBM Watson Assistant?

Watson Assistant is a robust platform that allows developers and non-technical users to collaborate on building conversational AI solutions. Its graphical UI, powerful NLP and familiar developer features allow the rapid creation of anything from simple chatbots to complex enterprise grade solutions for customer service and more. With the IBM Watson Assistant service, you can build a solution that understands natural-language input and uses machine learning to respond to customers in a way that simulates a conversation between humans.

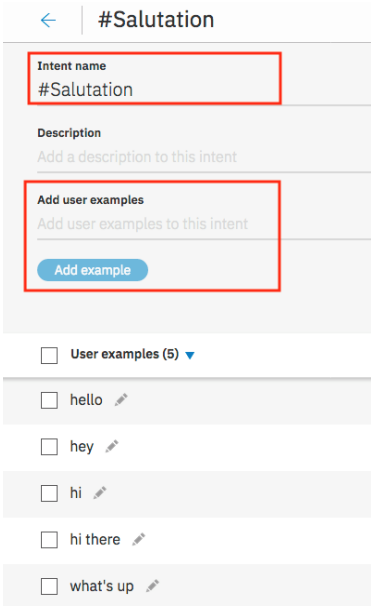
6.2.2. Building an Assistant

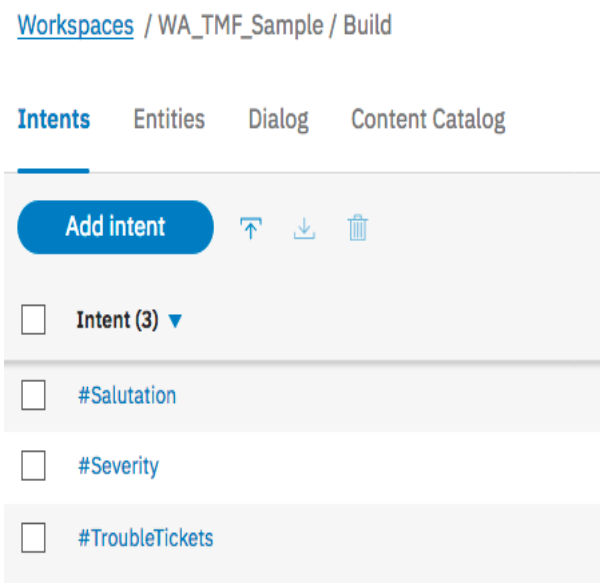
6.2.2.1. Creation of 'Intents'

Choose 'Intents' then select 'Add intent'



Provide 'Intent name' and 'Add example'.



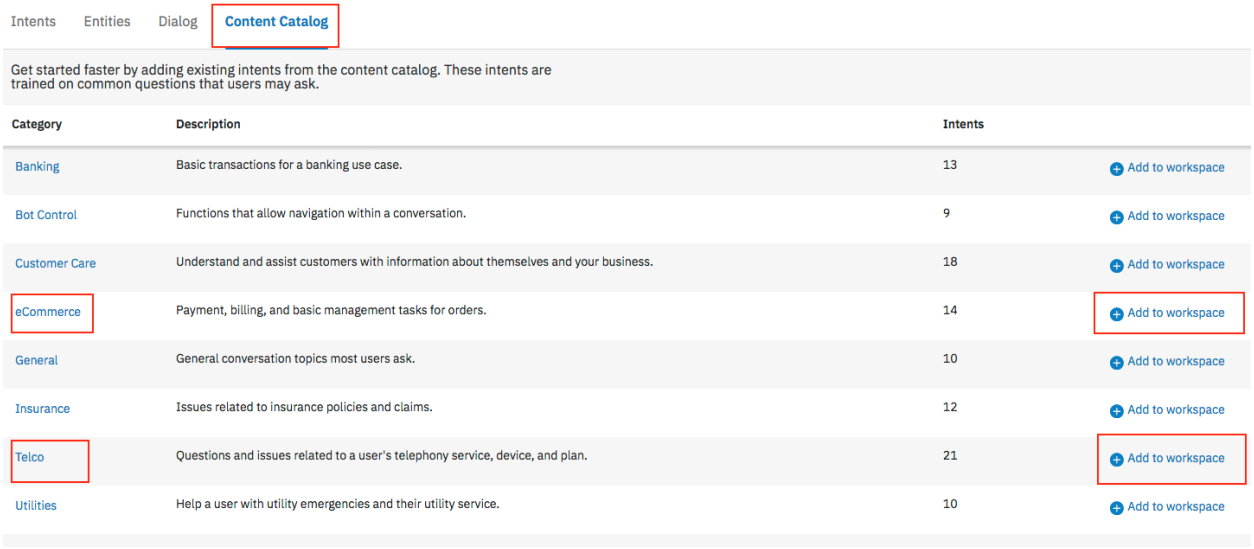


Please follow similar steps for ‘Usage’ intent using examples provided below.

In the appendix there are samples to create ‘Severity’ and ‘TroubleTickets’ Intents as well.

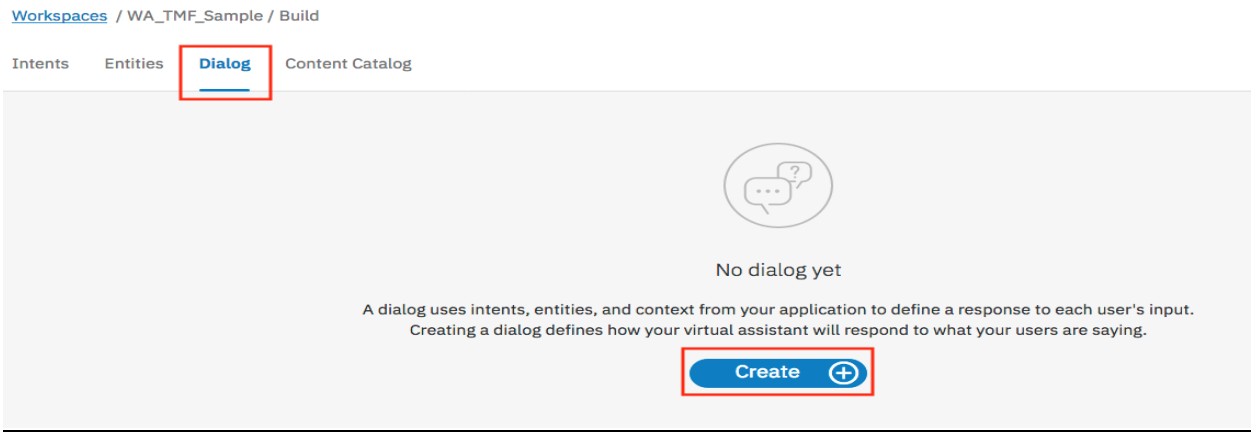
#Usage
can you just show me my overall usage?
Can you show me my overall usage?
It’s more than I expected, can you just show me my overall usage?
Why is it so much, show me my usage?

Navigate to ‘Content Catalog’. Add to workspace for categories: ‘ecommerce’ & ‘Telco’



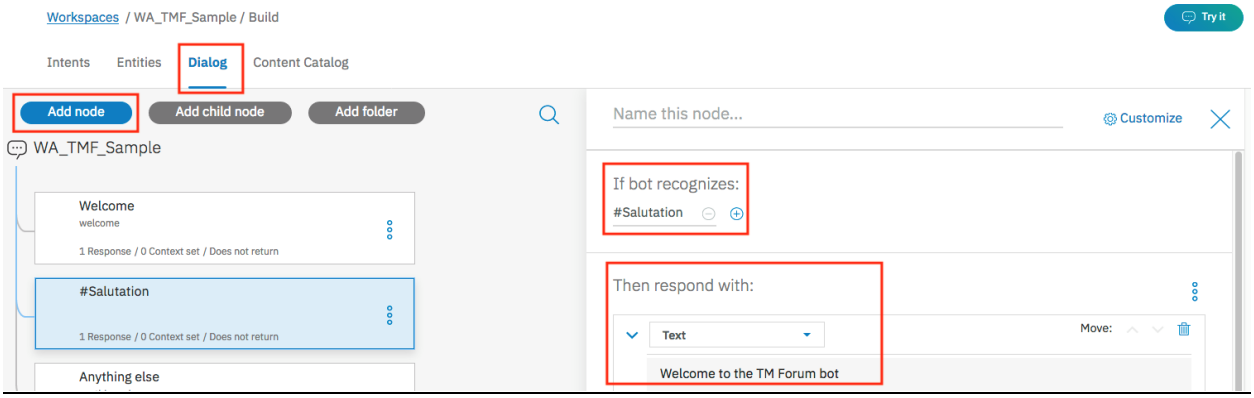
7.2.2.2 Creation of ‘Dialog’

Choose Dialog -> Create



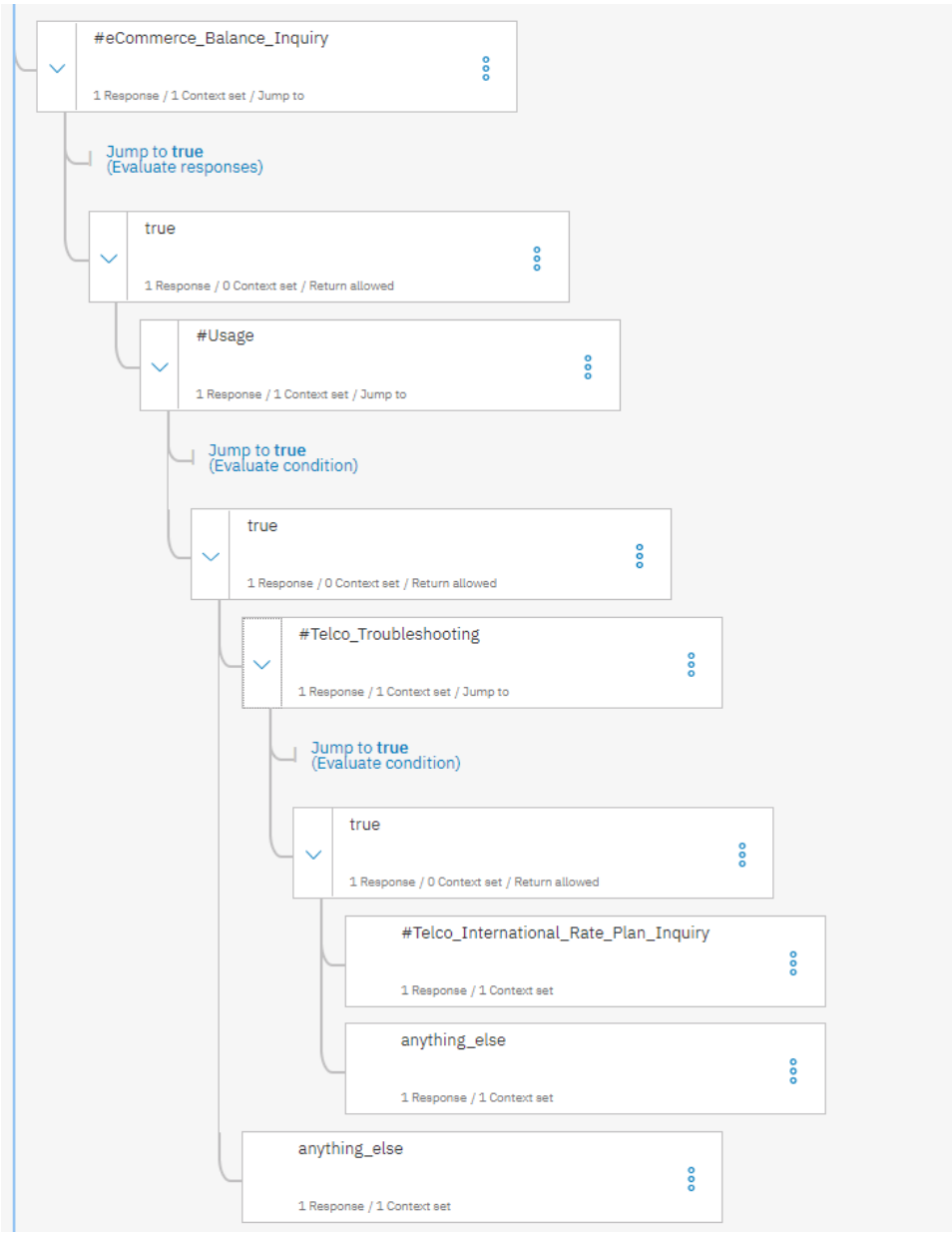
Creating a condition in conversation is done by adding a new node. Every new node is recognized by bot using either an intent or entity or context variable. Whenever bot recognizes a specific intent or entity, bot calls that specific node and then respond with the response type of the node.

Click “Add node” to add a new node and provide the following details for the bot to recognize this node.
If bot recognizes: Provide Intent/entity name, **then respond with**: Provide Watson response

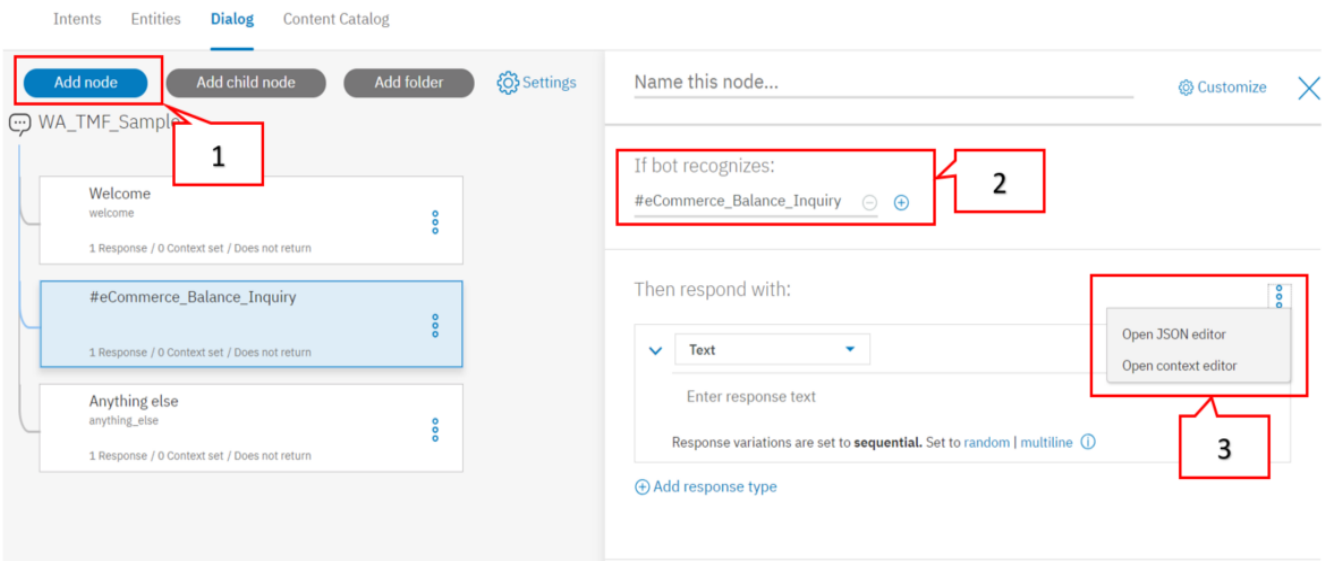


"#eCommerce_Balance_Inquiry:

Below image shows node "#eCommerce_Balance_Inquiry" and its child nodes.



Click "Add Node" and enter **"#eCommerce_Balance_Inquiry**" as intent in "If bot recognizes".



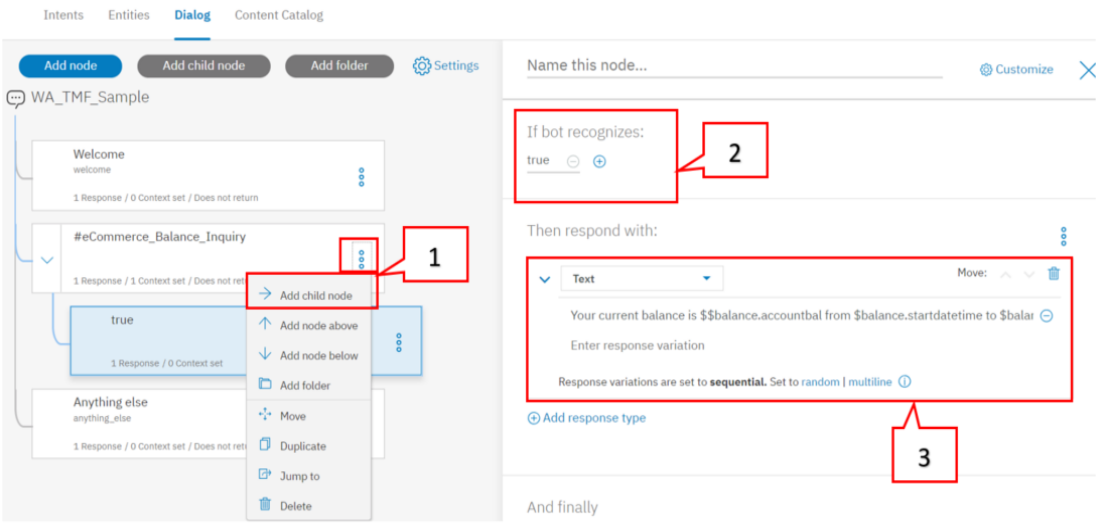
Then respond with: (Open JSON editor)

```
{
 "context": {
  "summary": "from-watson"
 },
 "output": {
  "text": {
   "values": [
    ""
   ],
   "selection_policy": "sequential"
  }
 },
 "actions": [
  {
   "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/getAccountInfo",
   "type": "server",
   "credentials": "$private.my_credentials",
   "result_variable": "balance"
  }
 ]
}
```

This name specifies the location of IBM Functions. The format of it would be NameSpace/Package/Action
Change it to match your IBM Functions.
Note: Watson Assistant and IBM functions should be on same cloudfoundry organization and space for them to call each other.
NameSpace is noted down in step (2) of section 7.1.2

The above node responds using a call to serverless service hosted on **IBM Functions** within the **same organization** as its response. Change the above address of IBM Functions to match your own organization's IBM Functions address created above in step(4) of section 7.1.2 (leaving it as such would not work).

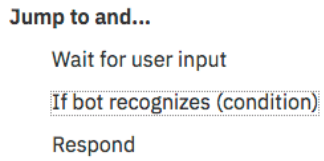
Once the processing is done by IBM Functions, the return value or result cannot be handled directly by this node. Hence, we create a child node "true" to handle the result.



For the child node "true" add the details as shown in the above image.

If bot recognizes: true,
Then respond with: "Your current balance is $$balance.accountbal from $balance.startdatetime to $balance.enddatetime."

Link these two nodes such that when ""**#eCommerce_Balance_Inquiry**" is recognized response is provided by "**true**" node. Navigate to "**#eCommerce_Balance_Inquiry**" and click on options ⋮ and "Jump to". Select "**true**" node and choose "Respond". This creates the link between the 2 nodes.



#Usage:

Add a node to recognize "#Usage" intent and then call IBM Functions using JSON editor. Response would be provided by its child node "true".

Click "Add node" (as a child of true node created above) and provide following details

If bot recognizes: "#Usage", then respond with: (Open JSON editor) (by clicking ⋮)

```
{
  "context": {
    "summary": "from-watson-table"
  },
  "output": {
    "text": {
      "values": [
        "Your Overall Usage for Voice, Data and SMS is as below:"
      ],
      "selection_policy": "sequential"
    }
  },
  "actions": [
    {
```

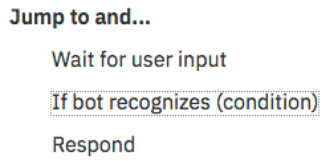
```
      "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/getUsageReport",
      "type": "server",
      "credentials": "$private.my_credentials",
      "result_variable": "result"
     }
    ]
}
```

Needs to be changed to match your IBM Functions address.

Add a child node “true” and add

If bot recognizes: true, then respond with: $result.

Link these two nodes such that when “**#Usage**” is recognized response is provided by “**true**” node. Navigate to “**#Usage**” and click on options ⋮ and “Jump to”. Select its child node “**true**” and choose “Respond”. This creates the link between the 2 nodes.



#Telco_Troubleshooting:

Add a node to recognize “**#Telco_Troubleshooting**” intent and then call IBM Functions using JSON editor. Response would be provided by its child node “true”.

Click “Add node” (as a child of true node of Usage created above) and provide below details

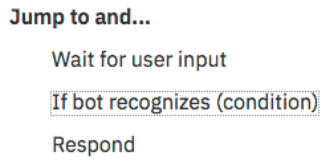
If bot recognizes: #Telco_Troubleshooting, then respond with: (Open JSON editor) (by clicking ⋮)

```
{
 "context": {
  "summary": "from-watson-table"
 },
 "output": {
  "text": {
   "values": [
    "Your top voice charges for this month are as below:"
   ],
   "selection_policy": "sequential"
  }
 },
 "actions": [
  {
   "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/getBillingEvents",
   "type": "server",
   "credentials": "$private.my_credentials",
   "result_variable": "result"
  }
 ]
}
```

Add a child node “true” and add

If bot recognizes: true, then respond with: $result.

Link these two nodes such that when "**#Telco_Troubleshooting**" is recognized response is provided by "**true**" node. Navigate to "**#Telco_Troubleshooting**" and click on options ⋮ and "Jump to". Select its child node "**true**" and choose "If bot recognizes (Condition)". This creates the link between the 2 nodes.



Anything_else:

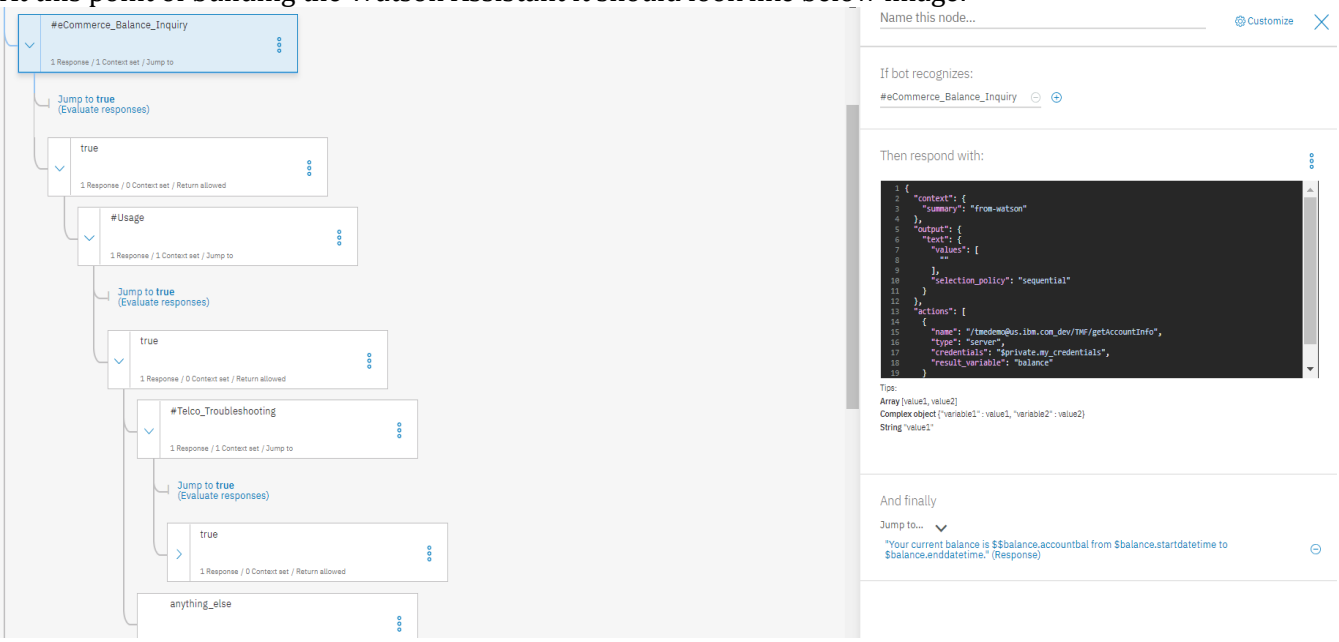
Add a child node to true node of Usage. Click "Add node" (as a child of true node of Usage created above) and set the below details.

If bot recognizes: "anything_else", then respond with: "Sure, let me connect to a human agent".

Set Context by clicking ⋮ and "Open context editor" and set the values below:

Variable: $summary, Value: "from-watson".

At this point of building the Watson Assistant it should look like below image:



#Telco_International_Rate_Plan_Inquiry:

Add a node to recognize "'#Telco_International_Rate_Plan_Inquiry".

Click "Add node" (as a child of true node of Telco_Troubleshooting created above) and provide below details:

If bot recognizes '#Telco_International_Rate_Plan_Inquiry', Then respond: "Sure, let me connect to a human agent"

Using options ⋮ and click "Open context editor", Set context for this node using

Variable: $summary, Value: "from-watson"

Anything_else:

Add a child node to true node of Usage. Click “Add node” (as a child of true node of Telco_troubleshooting created above) and set the below details.

If bot recognizes: “anything_else”, then respond with: “Sure, let me connect to a human agent”.

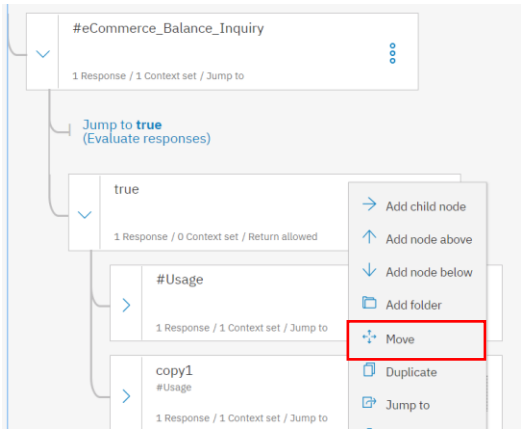
Set Context by clicking ⋮ and “Open context editor” and set the values below:

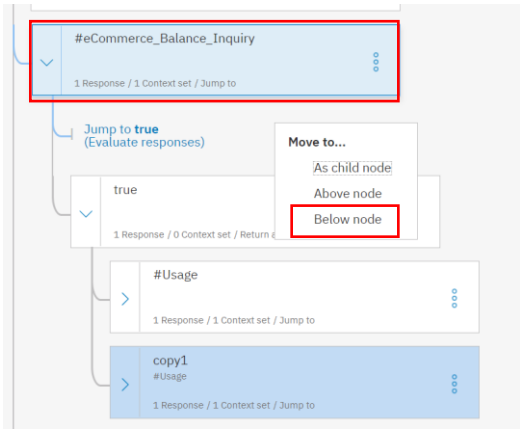
Variable: $summary, Value: “from-watson”.

Duplicate the Usage node:

In this step we duplicate the Usage node and its child nodes created above as a new parent node.
Select options of usage node by clicking ⋮ and select “Duplicate”. This will create the duplicate node with name “copy1”.

Click options on the copy1 node as shown below and click “Move”. Then Select “eCommerce_Balance_Inquiry” node to add copy1 as a node below it by selecting “Below node”. Rename “copy1” as “Usage”.



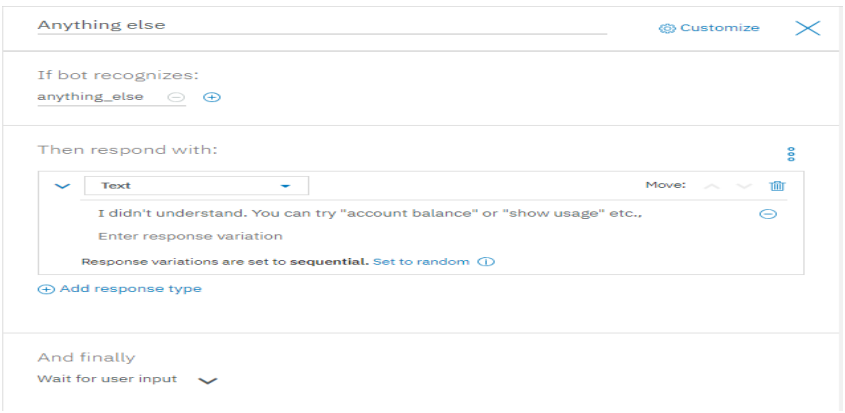


If bot recognizes: #thanks

Click “Add node” to add a new node (this is not a child node to any node) and then add the intent as “#thanks” by setting the value of
If bot recognizes as “#thanks”, then respond with: “It was nice helping you. Have a good day. Thank you!!!”

If bot recognizes: anything_else

Scroll down to find the “anything_else” node and set the response of it as shown below:

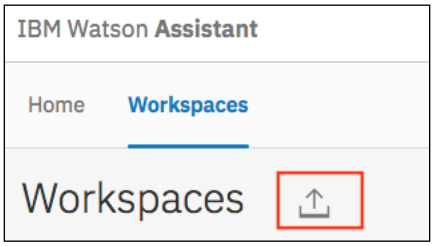


Step (*) (Alternate approach – Import using JSON file)

Create ‘Workspace’ with Name and Description

Import ‘WA_TMF.json’ file which is available within ‘Watson_Assistant’ folder.

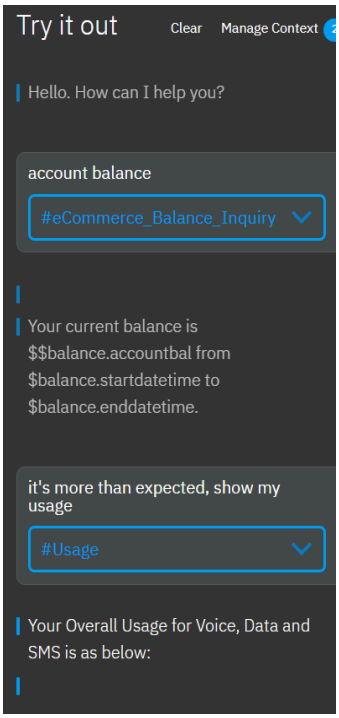
Github link: https://github.com/commgsc/TMF_ActionWeek_Clinic_Lab/tree/master/WatsonAssitant



Once json file has been imported, you could view ‘Intents’, ‘Entities’ and ‘Dialog’.

Step (3)

Try out the conversation, you just created!



6.3. Creating a Node-RED Flow

6.3.1. Introduction of the service

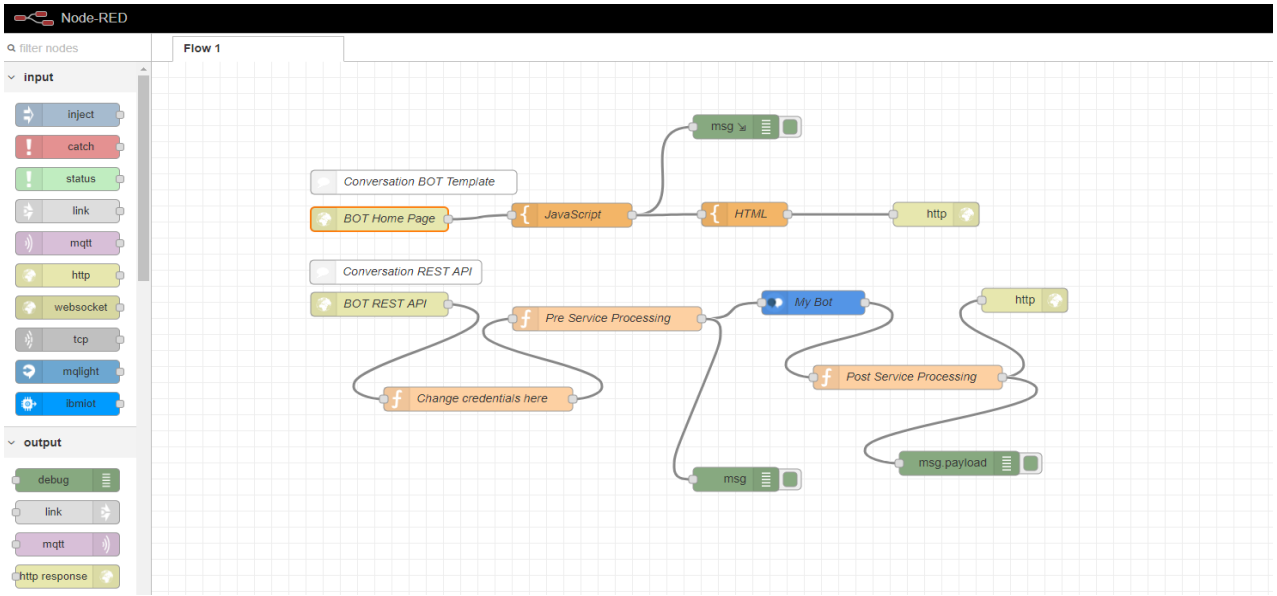
Node-RED is a powerful tool for building Internet of Things (IoT) applications with a focus on simplifying the 'wiring together' of code blocks to carry out tasks. It uses a visual programming approach that allows developers to connect predefined code blocks, known as 'nodes', together to perform a task. The connected nodes, usually a combination of input nodes, processing nodes and output nodes, when wired together, make up a 'flows'.

Node-RED provides a browser-based flow editor, which can be used to create JavaScript functions. Elements of applications can be saved or shared for re-use. The runtime is built on Node.js. The flows created in Node-RED are stored using JSON.

6.3.2. Importing the node-red flow:

So, the basic node-red flow would be to submit the user's input using a http request and then use the JavaScript of node-red to make an ajax call to Watson assistant. Watson assistant would send a response either as a text message or an object by calling IBM functions. This response is processed accordingly in the JavaScript and displayed on screen using HTML template.

In a new browser tab navigate to Github Repository. Navigate to node-red-flow.json in NodeRed folder. Copy the json file.
Open the node-red flow editor created in section 6.3 (Instantiate a node-red starter). Click the options button and then scroll over "Import" and click "Clipboard". Paste the node-red-flow.json copied above into the Clipboard and click "Import" to import the application flow.



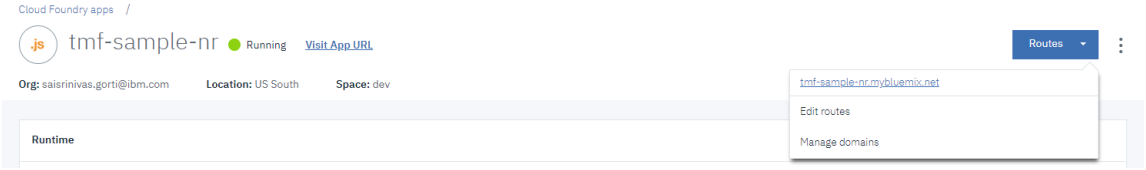
Node-red application flow would look like the above image after importing the json file. The flow requires IBM Functions and Watson Assistant credentials to be changed to run the application successfully. Change the IBM Functions credentials by double clicking on “Change credentials here” node to open the node and change the username and password to the credentials noted down in step (2) of section 7.1.2.

Change the Watson Assistant credentials by double clicking on “My Bot” node to open the node and change the username, password and Workspace_ID to the credentials noted down in step (2) of section 6.2.

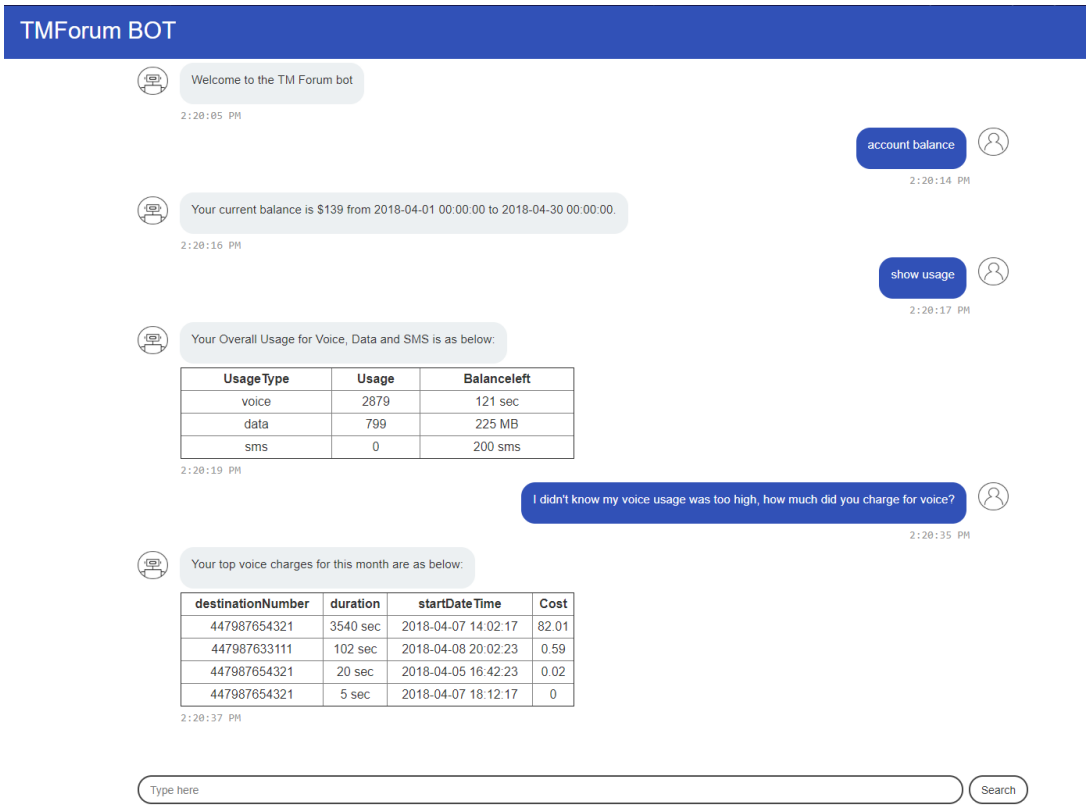
Click “Deploy” on the top right corner of the page once the credentials are changed. “Successfully deployed” alert would be displayed if the deployment is successful. At this point the application is successfully deployed on IBM Cloud.

6.3.3. Testing the node-red Application:

Navigate to IBM Cloud dashboard by logging into https://console.bluemix.net with respective credentials created in section 6.1. Click on the node-red application created above.



Click on the routes to note down the application url. This url opens up the node-red flow editor. Open a new browser tab and append “/bot” to the url and press enter to launch the node-red application.



6.4. *Creating and configuring the Node.js Application (Optional):*

A Node.js Application is also developed for this lab (Check it out if interested). Reactjs is used for developing front-end of the application.

Both the nodejs application and node-red application uses the same conversation, openwhisk functions and are developed using same use case. This application receives inputs from user (UI) and calls an openwhisk function to call the Watson assistant to get the intents and decide which action of IBM functions to be called next.

6.4.1. Setup the environment:

IBM Cloud CLI provides the command line interface to manage applications, containers, infrastructures, services and other resources in IBM Cloud. Please follow the instructions in the below link to install IBMCloud CLI and many other tools like IBM Cloud Functions plug-in, docker, kubectl (Kubernetes) etc.,

https://console-dal10.bluemix.net/docs/cli/reference/bluemix_cli/get_started.html#getting-started

6.4.2. Clone the Application from github:

Navigate to git repository and clone the repository using command
"git clone https://github.com/commgsc/TMF_ActionWeek_Clinic_Lab.git" (assuming that git is already installed).

Use this link to install git if it is not installed already.

6.4.3. Create the Open Whisk Functions:

IBM® Cloud Functions is a distributed, event-driven compute service also referred to as Serverless computing or as Function as a Service (FaaS). Cloud Functions runs application logic in response to events or direct invocations from web or mobile apps over HTTP.

IBM functions are based on openwhisk. Examples, resources, as well as the original openwhisk code can be found on these links:
https://console.bluemix.net/docs/openwhisk/openwhisk_about.html#platform-architecture
https://github.com/apache/incubator-openwhisk

To create the IBM Functions' actions, you should run the following commands:

```
ibmcloud login (to authenticate to your IBM Cloud Account)
ibmcloud wsk action create conversation1 actions/watson-assistant.js –web true
ibmcloud wsk action update conversation1 --param-file config/watson-assistant-config.json
```

6.4.4. Deploy the Application into IBM Cloud:

Navigate to the application folder in your command prompt and use "**npm start**" to run the application locally.

If it runs without any error, application running locally opens in a new browser window. If any changes are done to the application, after testing locally that it's working run "**npm run build**" from the command prompt to build the changes made. After successful build, log in to bluemix using "**bx login**" or "**ibmcloud login**".

```
C:\Srinivas\git\tmf_sample_app>bx login
API endpoint: https://api.us-east.bluemix.net

Email> srinivas11193@gmail.com

Password>
Authenticating...
OK

Select an account:
```

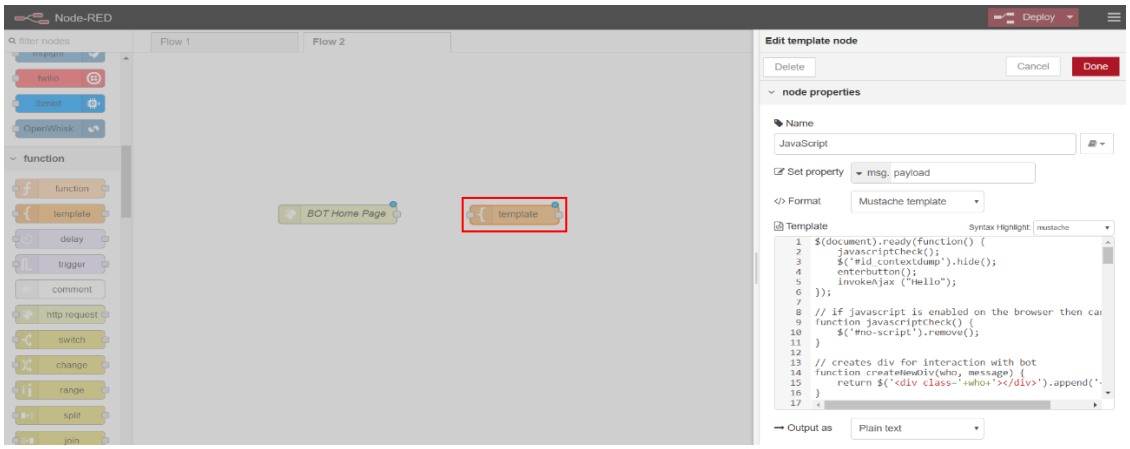
Target a specific organization and space using "**bx target –cf**" or "**ibmcloud target –cf**" commands. Once the bluemix credentials are pointing to proper organization and space. Use "**ibmcloud app push**" to push the application to IBM Bluemix. Test the application with that url returned to you in command line after successful publish.

6.5. Creating Node-Red flow (optional):

Step (1)

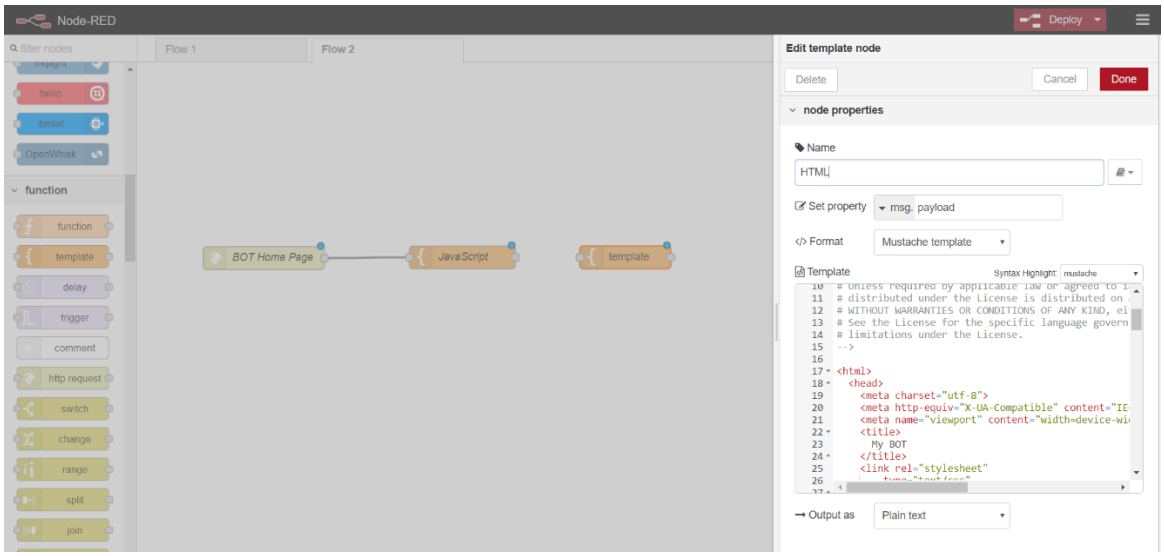
To create a node-red flow, open node-red editor as shown above. Drag "http" node from nodes on left side on to flow editor. Double click the http node to configure request url for http request. Type "/bot" for URL and "BOT Home Page" for name.

Drag "template" node from nodes on left side on to flow editor. Double click the template node to configure template name and its content. Type "JavaScript" for name and copy the code from github repo

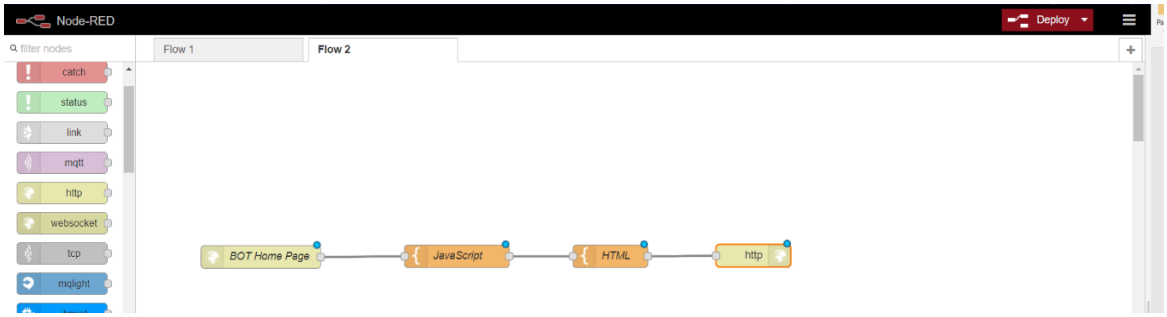


Link both the nodes by dragging output of the bot home page to the input of JavaScript node.

Drag “template” node from nodes on left side on to flow editor. Double click the template node to configure template name and its content. Type “HTML” for name and copy the HTML code from github repo.



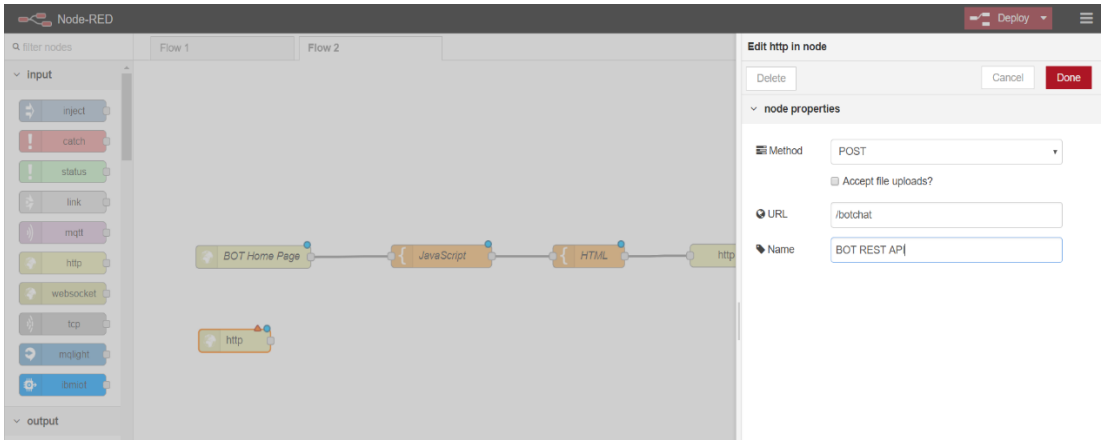
Link Javascript and HTML nodes. Add “http response” node to the flow by dragging from list of nodes. Link HTML and http response. Now we have completed a subflow of getting a request and submitting a response. At this stage, the flow should like the image shown below:



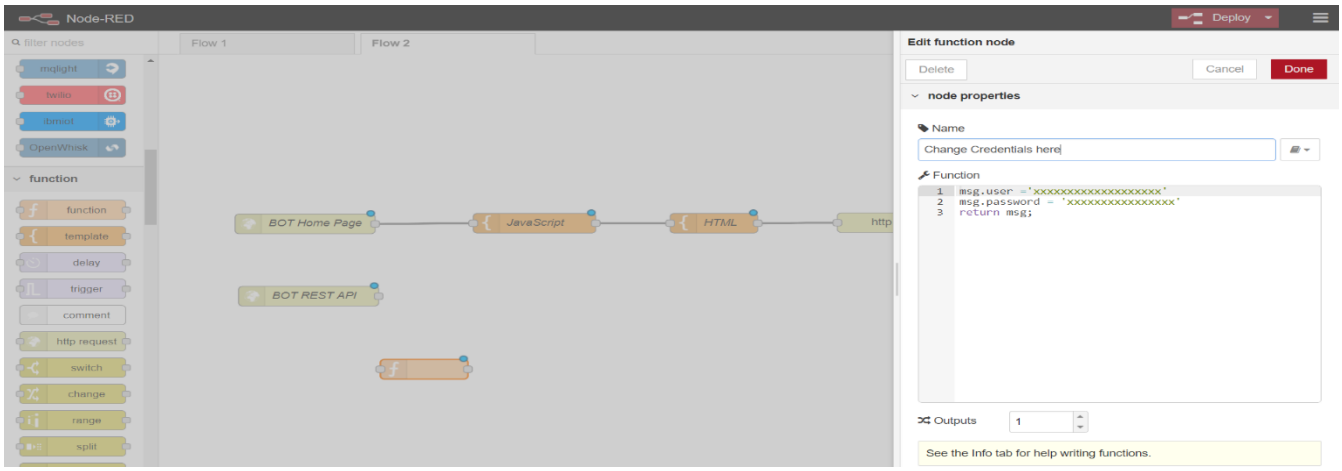
Step (2)

As we have created the basic application flow, now we would need the nodes which help us achieve the complete flow. We need to have a Watson assistant embedded in our flow, which would call IBM functions internally. So, the credentials for IBM functions has to be sent to Watson assistant using the flow.

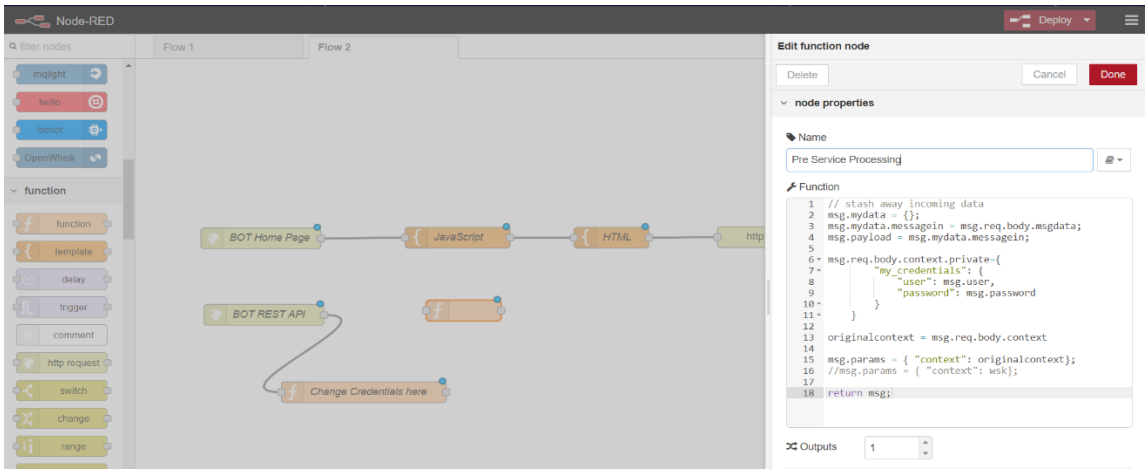
Starting with sub flow, we would be needing a “http request” node with method POST as we need to post the user input to Watson assistant to get the response. Add a http request and type “/botchat” for URL and “BOT REST API” for name.



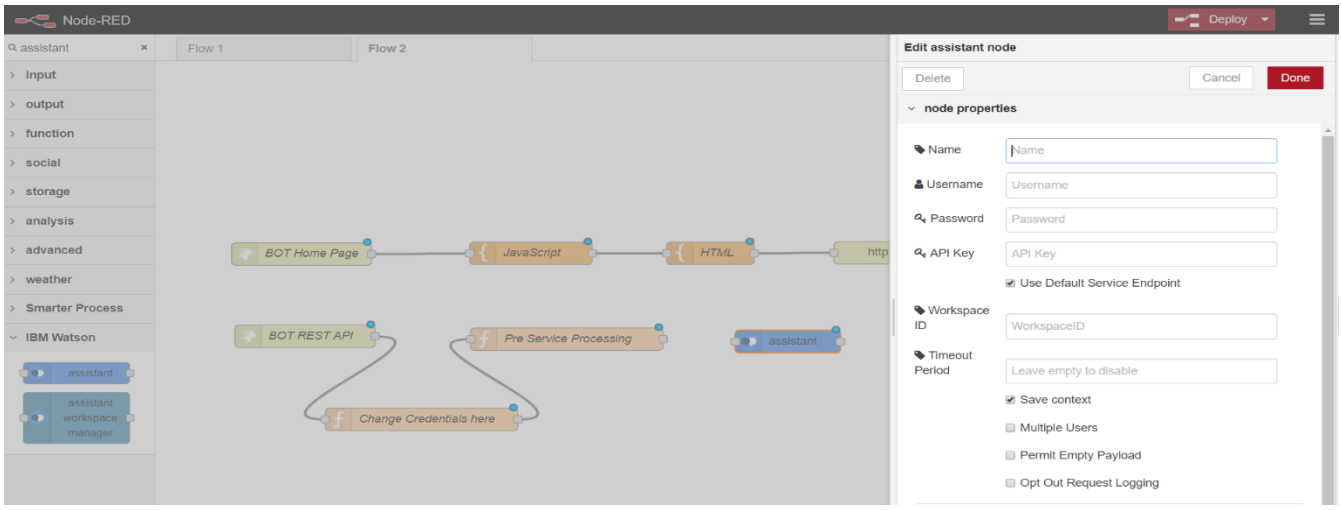
Add a “function” node to the flow editor and type “Change Credentials here” for name and copy text from “Change credentials here” file and paste it in the function text box. These credentials are created in step (2) in section 7.1.2. Link the http request node “BOT REST API” and “Change Credentials here” function node as discussed above.



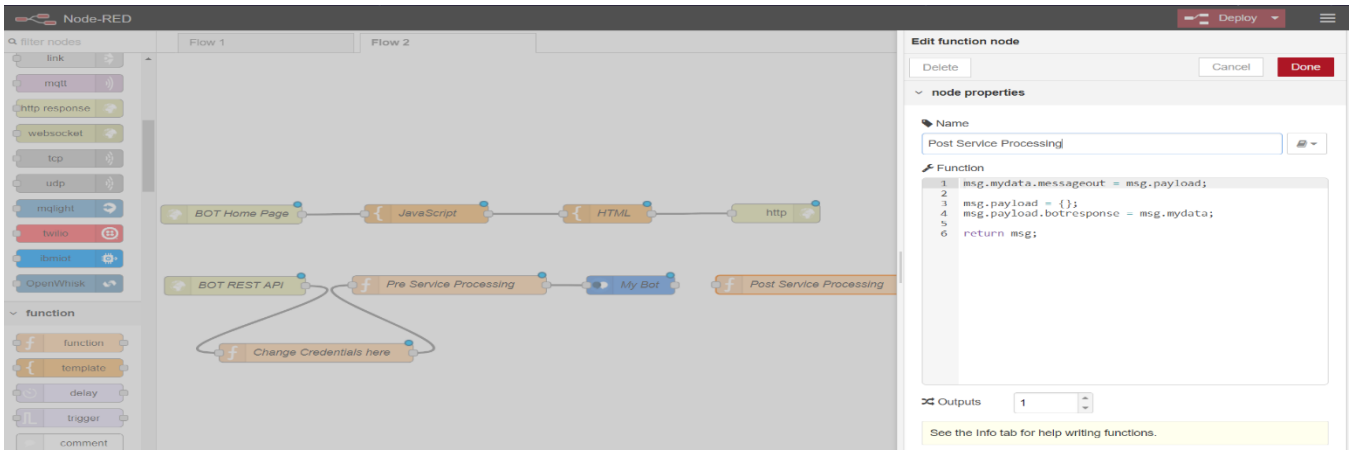
Add another function node for processing these credentials and sending as input to watson assistant. Copy the function code from node-red folder of github repo. Link “Change Credentials here” and “Pre Service Processing” nodes.



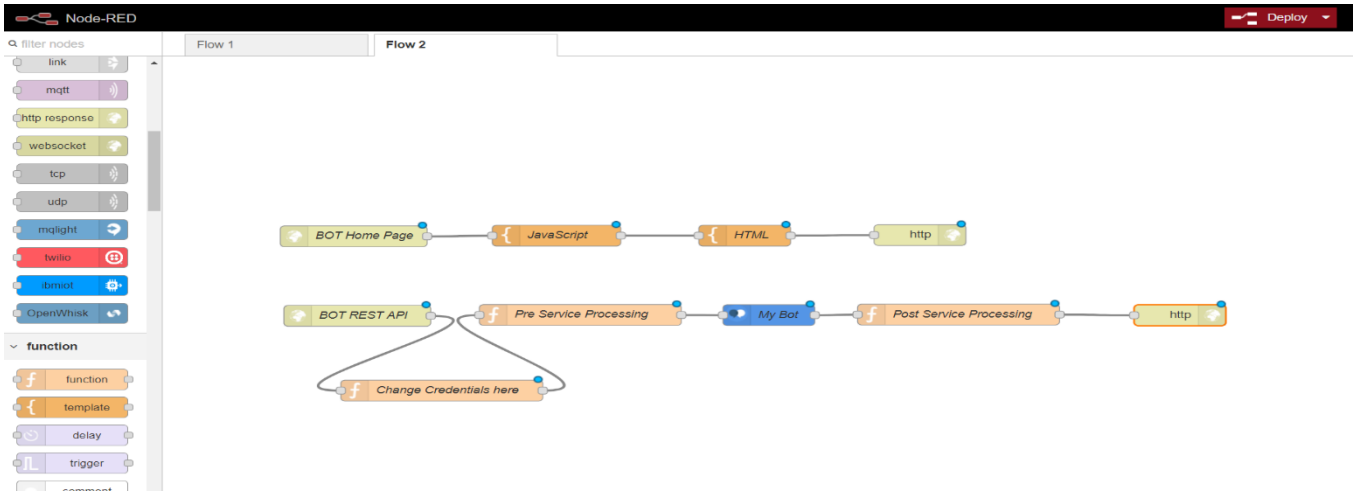
Search for assistant node using the search in node palette on left side. Type name as “My Bot” and add required credentials like username, password and workspace ID from your watson assistant created above. Link “Pre Service Processing” node and “My Bot” node.



Add a function node and http response node to the flow by dragging them from left onto flow editor. Name the function node as “Post Service Processing” and copy the content from github repo. This function receives the bot response and sends it to http response to submit it to JavaScript. Link “My Bot” to “Post Service Processing” and “Post Service Processing” to “http” response node.



This completes the node-red flow for the application. At this stage it should look like the flow shown below.



7. Appendix

7.1. API Connect Hands-on:

Account Management use case:

7.1.1. Install LoopBack Tools:

Loopback CLI tool: Install loopback command-line interface (CLI) tool using command "npm install -g loopback-cli".

Install IBM API Connect v5 developer toolkit: IBM API Connect is an end-to-end API management solution that uses Loopback to create APIs and provides integrated build and deployment tools. Install IBM API Connect v5 developer toolkit using "**npm install -g apiconnect**".

7.1.2. Create LoopBack Application:

Run the loopback application generator "**apic loopback**" to create a new application and follow the prompts to give name of application, directory of application, version of loopback, and kind of application (empty server in our case).

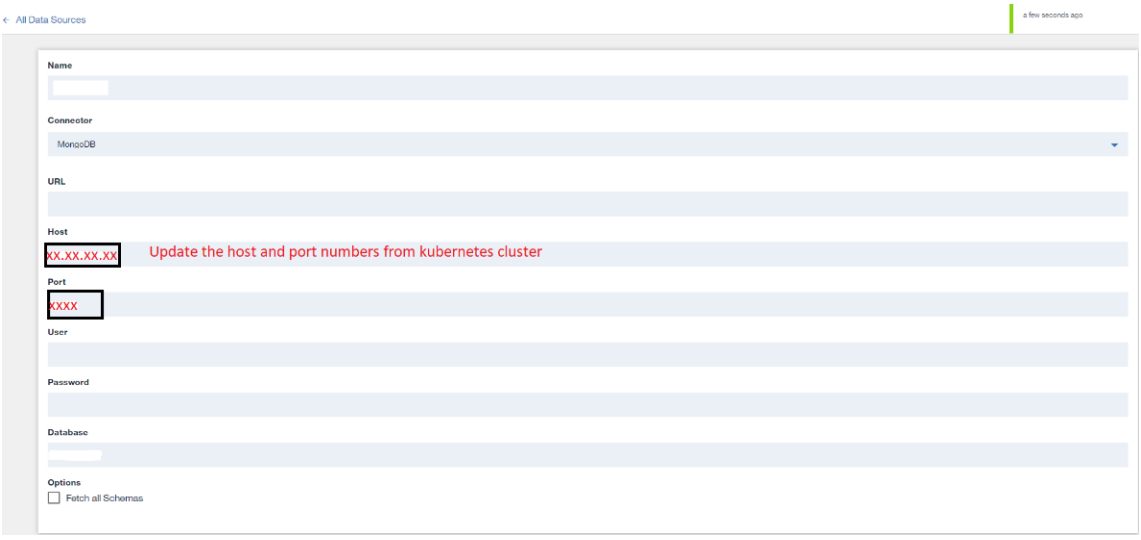
```
C:\Srinivas\IBM projects\TMF_Test>apic loopback
? What's the name of your application? selfcare-account
? Enter name of the directory to contain the project: selfcare-account
? Which version of LoopBack would you like to use? 3.x (current)
? What kind of application do you have in mind?
> empty-server (An empty LoopBack API, without any configured models or datasources)
  hello-world (A project containing a controller, including a single vanilla Message and a single remote method)
  notes (A project containing a basic working example, including a memory database)
```

7.1.3. Creating a Data Source:

This can be done using CLI as well as API designer.

Command Line Interface:
Create a Data Source using apic CLI using "**apic create –type datasource**" and follow the prompts to provide the database url, host, port, user, password and database.

API Designer:
Navigate to Data Sources section of API Designer which can be opened using command "**apic edit**". Provide details of the database to be used in the application.



7.1.4. Import TM Forum APIs:

Use “apic loopback:swagger” to import TM Forum API into the sample application you created. Provide the File path or url of the TM Forum swagger file to be imported. Follow the prompts to select models to be generated and data sources to be attached for those respective models.

```
C:\Srinivas\IBM projects\TMF_Test\selfcare-account>apic loopback:swagger
? Enter the swagger spec url or file path: ../AccountManagement.yaml
Loading ../AccountManagement.yaml...
? Select models to be generated: swagger_tmf-api_accountManagement_v2_, BillingAccount
? Select the data-source to attach models to: (Use arrow keys)
> selfcare-app-db (mongodb)
  (no data-source)
```

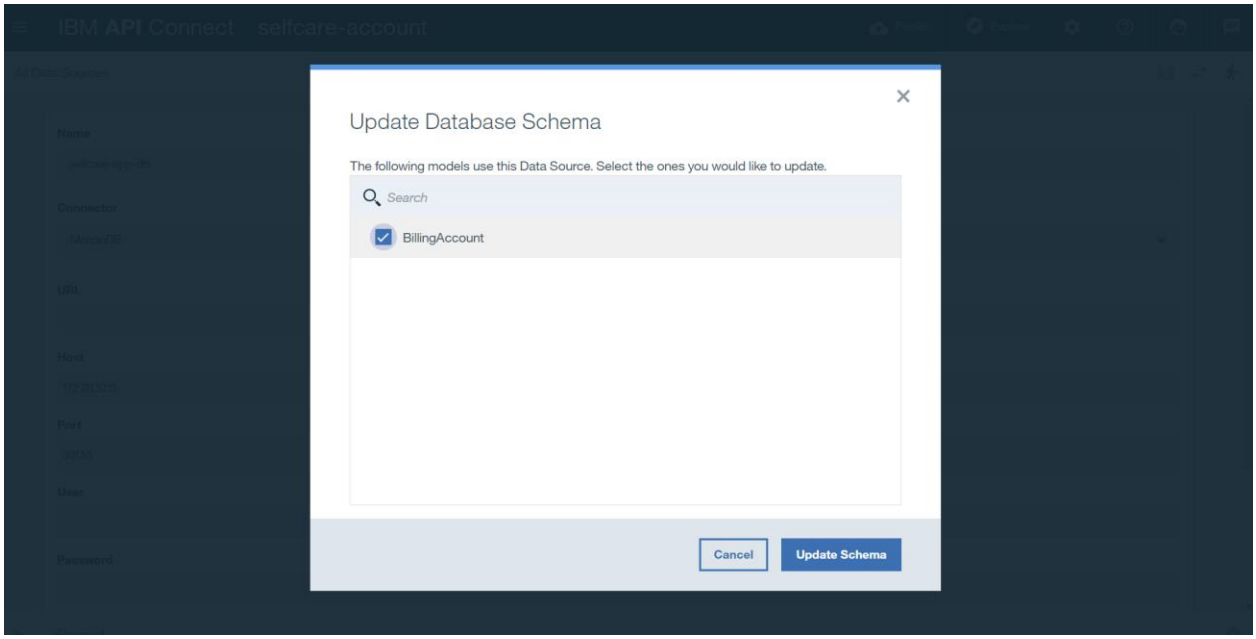
Execution of above command creates models and links them with the datasource.

```
C:\Srinivas\IBM projects\TMF_Test\selfcare-account>apic loopback:swagger
? Enter the swagger spec url or file path: ../AccountManagement.yaml
Loading ../AccountManagement.yaml...
? Select models to be generated: swagger_tmf-api_accountManagement_v2_, BillingAccount
? Select the data-source to attach models to: selfcare-app-db (mongodb)
Creating model definition for swagger_tmf-api_accountManagement_v2_...
Creating model definition for BillingAccount...
Model definition created/updated for swagger_tmf-api_accountManagement_v2_.
Model definition created/updated for BillingAccount.
Creating model config for swagger_tmf-api_accountManagement_v2_...
Creating model config for BillingAccount...
Model config created for swagger_tmf-api_accountManagement_v2_.
Model config created for BillingAccount.
Generating C:\Srinivas\IBM projects\TMF_Test\selfcare-account\server\models\swagger_tmf-api_accountmanagement_v2_.js
Models are successfully generated from swagger spec.
Done running LoopBack generator

Updating swagger and product definitions
Created C:\Srinivas\IBM projects\TMF_Test\selfcare-account\definitions\selfcare-account.yaml swagger description
```

7.1.5. Create Database Schemas:

After the models are generated, Open the API Designer using “**apic edit**” and navigate to the data sources page. Find this button ⇄ on the top right corner of the page. Click that to select the model schemas to be generated in the database. This schema generation uses the model definitions in the TM Forum swagger file.

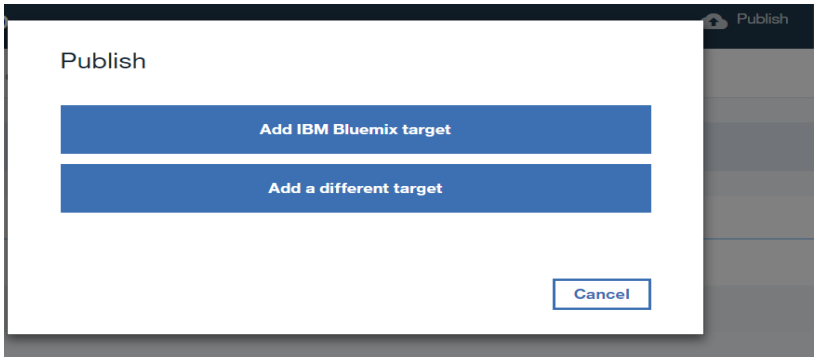


7.1.6. Test the Application:

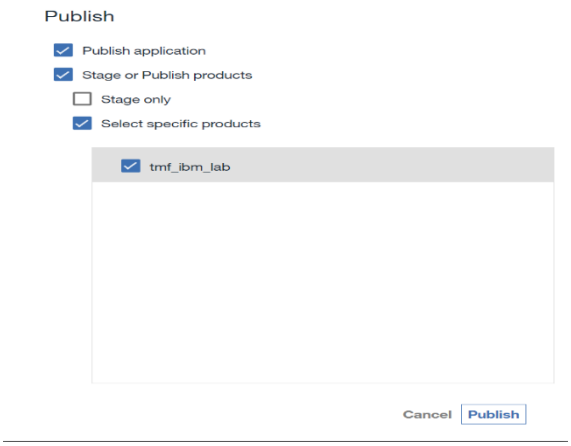
Test the application locally by clicking explore after starting the server using play button in bottom left corner. Click the API endpoint to be checked and fill the data accordingly to call the operation.

7.1.7. Push the application to cloud:

Publish the application using the publish button on top right corner. Add target according to your specification. Click “Add IBM Bluemix target” and select the organization, catalog and add an application name to publish the application.



Once the target is added, click publish and select the target you just added. Select the below options and click “**publish”**.

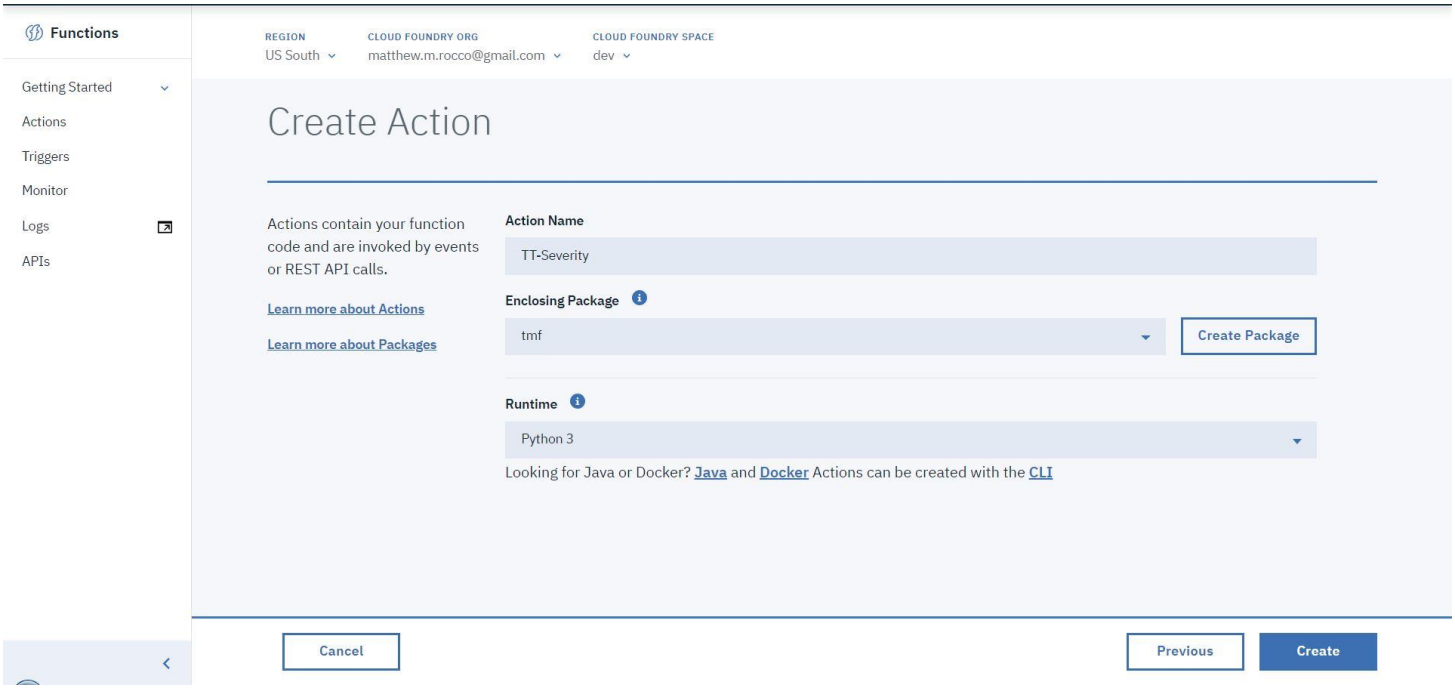


Once published and the application is started check by navigating to your api connect instance and catalog and by clicking “**explore”** to test the application.

7.2. Building a Trouble Ticket Use case with Watson Assistant.

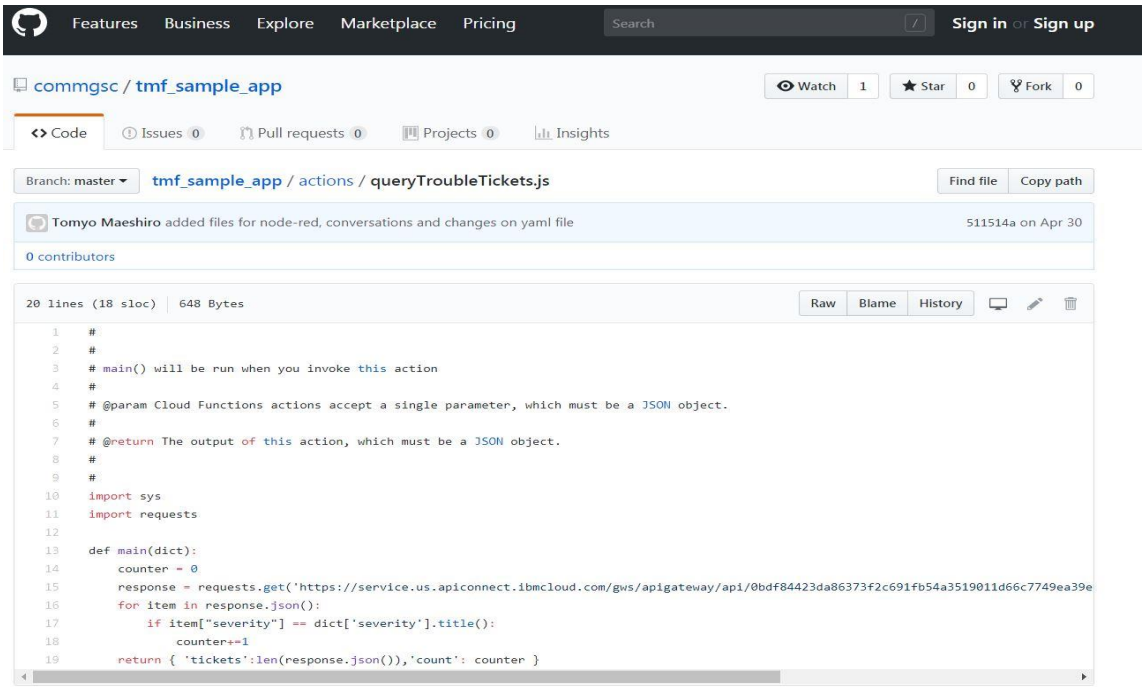
7.2.1 IBM Functions - TroubleTicket OpenWhisk Functions:

Enter “TT-Severity” as the Action Name. Use the “Create Package” button to create a Package. Name it tmf. Select Python 3 as your Runtime in the Dropdown Menu. Then click the blue “Create” Button in the lower right corner.

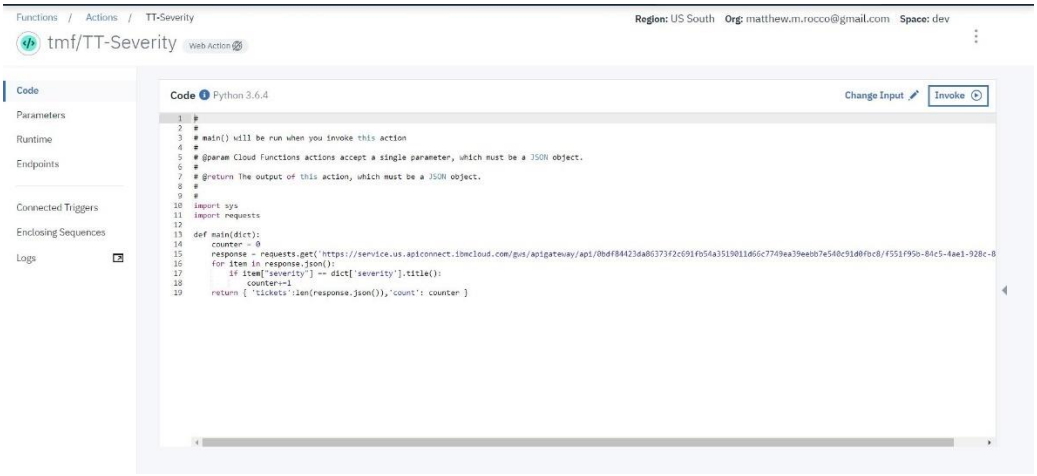


Step (3)

In a separate browser tab, navigate to: Within the Git repository, navigate to IBM-WatsonAssistant-SelfCare/actions/queryTroubleTickets.js. Copy the code that you find there.

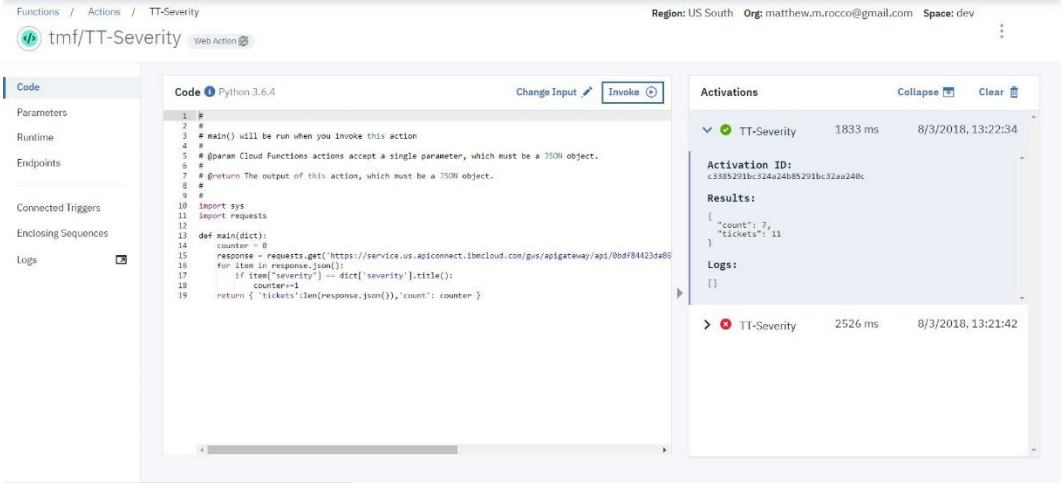


Back in your IBM Functions Tab, paste the code from the Git Repository in the “code” editor in the center of the page. Once pasted, Click the “Change Input” icon above the code editor.



Change the Action Input to {“severity”:”High”}, as shown. Then Click the Blue “Apply” button

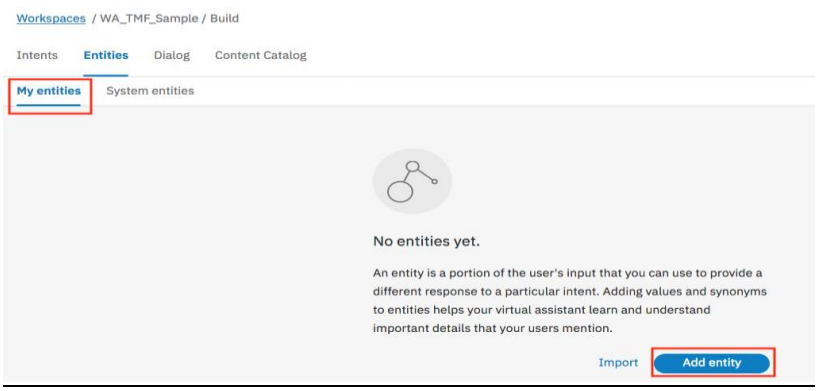
Back on the main page, Click the “Invoke” Button above the code editor in the center of the page. You can see the results of your test in the “Activations” Window shown at right in the image below:



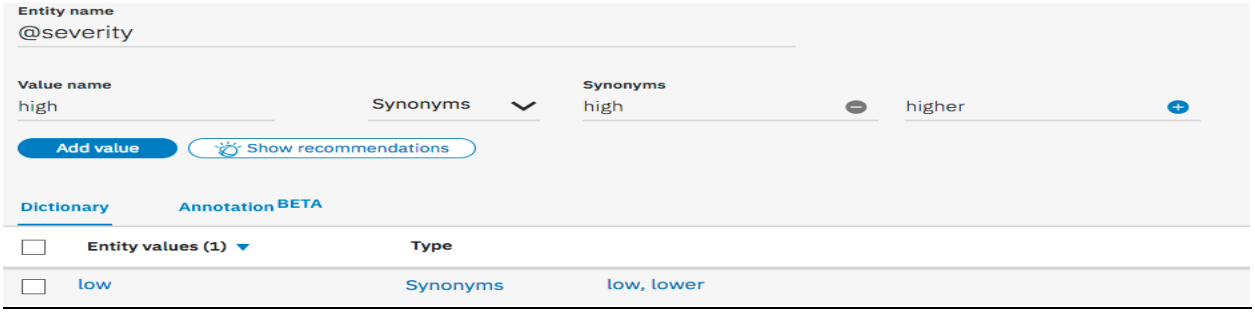
7.2.2. Watson Assistant - TroubleTicket in Watson Assistant:

7.2.2.1. Creation of 'Entities'

Choose Entities -> My entities -> Add entity



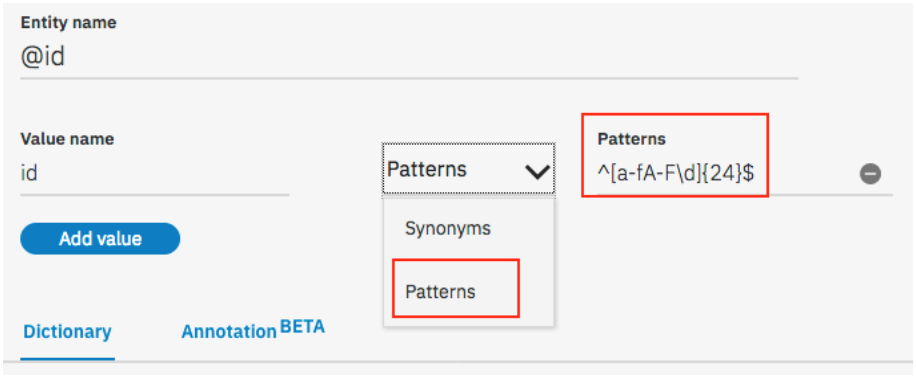
Provide Entity name, Value and Synonyms



Entity name: @id

Type: Choose Patterns

Patterns: ^[a-fA-F\d]{24}$



7.2.2.2. Creation of Intents:

#Severity
How many trouble tickets are of high severity
How many tickets are marked as low
How many high Severity tickets are there?
High Severity tickets

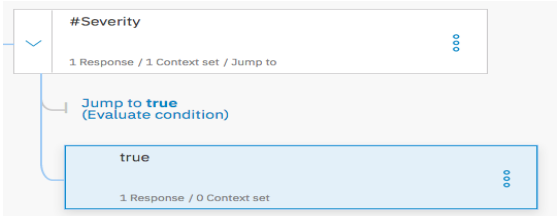
#TroubleTickets
Trouble Tickets
Tickets
need info about tt
I want to query about trouble tickets
I want to discuss a trouble ticket

#ticketdetails
can you give some details on those high severity tickets
can you give some details on those low severity tickets
Need details about high severity trouble tickets
Need details about low severity trouble tickets

#ticket-info
Can you give me some info on above tickets
Can you give me some more information on these tickets
Some info on these tickets

7.2.2.3. Creation of Dialog:

#Severity



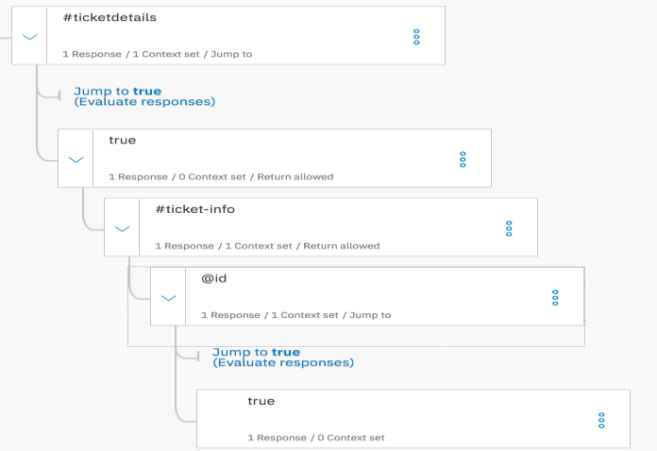
If bot recognizes: #Severity, then respond with: (Open JSON editor)

```
{
 "context": {
  "summary": "from-watson"
 },
 "output": {
  "text": {
   "values": [ "" ],
   "selection_policy": "sequential"
  }
 },
 "actions": [
  {
   "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/TT_Severity",
   "type": "server",
   "parameters": {
    "severity": "@severity"
   },
   "credentials": "$private.my_credentials",
   "result_variable": "count"
  }
 ]
}
```

Click on ‘Jump to’ and choose ‘If bot recognizes (condition)’

If bot recognizes: true, Then respond with: “The number of @severity severity tickets are $count.count”

#ticketdetails



If bot recognizes: #ticketdetails, Then respond with:

```
{ "context": {
   "summary": "from-watson-table"
 },
 "output": {
   "text": {
     "values": [      ""      ],
     "selection_policy": "sequential"    } },
 "actions": [   {
     "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/TT_Details",
     "type": "server",
     "parameters": {
       "severity": "@severity"    },
     "credentials": "$private.my_credentials",
     "result_variable": "tickets"
   } ] }
```

If bot recognizes: true

Then respond with: $tickets

If bot recognizes: #ticket-info

Then set context:

Variable: $summary

Value: “from-watson”

And respond with: Please provide id of the trouble ticket you are looking for?

If bot recognizes: @id

Then respond with: (Open JSON editor)

```
{
  "context": {
```

```
  "summary": "from-watson"
 },
 "output": {
  "text": {
   "values": [       ""      ],
   "selection_policy": "sequential"
  } },
 "actions": [    {
   "name": "/tmedemo@us.ibm.com_dev/TMForumLabs/GetTroubleTicket",
   "type": "server",
   "parameters": {
    "id": "@id.literal"
   },
   "credentials": "$private.my_credentials",
   "result_variable": "ticket"
  } ] }
```


If bot recognizes: true

Then respond with: $ticket