

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 Software designing
4	EXPERIMENTAL INVESTIGATIONS
5	FLOWCHART
6	RESULT
7	ADVANTAGES & DISADVANTAGES
8	APPLICATIONS
9	CONCLUSION
10	FUTURE SCOPE
11	BIBILOGRAPHY

1. INTRODUCTION:

A **chatbot** is a [software](#) application used to conduct an on-line chat [conversation](#) via text or text-to-speech, in lieu of providing direct contact with a live human agent. Designed to convincingly simulate the way a human would behave as a conversational partner, chatbot systems typically require continuous tuning and testing, and many in production remain unable to adequately converse or pass the industry standard [Turing test](#). The term "ChatterBot" was originally coined by [Michael Mauldin](#) in 1994 to describe these conversational programs.

Chatbots are used in [dialog systems](#) for various purposes including customer service, request routing, or for information gathering. While some chatbot applications use extensive word-classification processes, [natural language processors](#), and sophisticated [AI](#), others

simply scan for general keywords and generate responses using common phrases obtained from an associated library or [database](#).

Most chatbots are accessed on-line via website popups or through [virtual assistants](#). They can be classified into usage categories that include: [commerce](#) ([e-commerce](#) via chat), [education](#), [entertainment](#), [finance](#), [health](#), [news](#), and [productivity](#).

1.1 Over view:

Movie Ticketing bot is a project that has been developed under Faculty Development Program(FDP) by name Gurucool, a unique initiative by IBM and SmartBridge to enable faculty on emerging technologies via project-based learning. The program is planned for a month with 6 days of instructor-led training on AI, ML, IoT, Cloud Native followed by Project Build-A-Thon.

1.2 Purpose:

Movie Ticketing bot project was developed based on observations at a Cine multiplex by name Gorantla Cinemas, Ongole, Andhra Pradesh, India. While developing Movie Ticket Bot, the developer gets familiar with developing a chatbot application using IBM Watson Assistant, Node-Red web integration. The project being developed can be extended further by integrating with a database. The project is capable of addressing the issues of Consumer pertaining to a multiplex or the needs of a Small town with few movie halls.

2. LITERATURE SURVEY:

In 1950, [Alan Turing](#)'s famous article "[Computing Machinery and Intelligence](#)" was published, which proposed what is now called the [Turing test](#) as a criterion of intelligence. This criterion depends on the ability of a [computer program](#) to impersonate a human in a real-time written conversation with a human judge to the extent that the judge is unable to distinguish reliably—on the basis of the conversational content alone—between the program and a real human. The notoriety of Turing's proposed test stimulated great interest in [Joseph Weizenbaum](#)'s program [ELIZA](#), published in 1966, which seemed to be able to fool users into believing that they were conversing with a real human. However Weizenbaum himself did not claim that ELIZA was genuinely intelligent, and the introduction to his paper presented it more as a debunking exercise:

[In] artificial intelligence ... machines are made to behave in wondrous ways, often sufficient to dazzle even the most experienced observer. But once a particular program is unmasked, once its inner workings are explained ... its magic crumbles away; it stands revealed as a mere collection of procedures ... The observer says to himself "I could have written that". With that thought, he moves the program in question from the shelf marked "intelligent", to that reserved for curios ... The object of this paper is to cause just such a re-evaluation of the program about to be "explained". Few programs ever needed it more.

ELIZA's key method of operation (copied by chatbot designers ever since) involves the recognition of clue words or phrases in the input, and the output of corresponding pre-prepared or pre-programmed responses that can move the conversation forward in an apparently meaningful way (e.g. by responding to any input that contains the word 'MOTHER' with 'TELL ME MORE ABOUT YOUR FAMILY'). Thus an illusion of understanding is generated, even though the processing involved has been merely superficial.

ELIZA showed that such an illusion is surprisingly easy to generate because human judges are so ready to give the benefit of the doubt when conversational responses are *capable of being interpreted* as "intelligent".

Interface designers have come to appreciate that humans' readiness to interpret computer output as genuinely conversational—even when it is actually based on rather simple pattern-matching—can be exploited for useful purposes. Most people prefer to engage with programs that are human-like, and this gives chatbot-style techniques a potentially useful role in interactive systems that need to elicit information from users, as long as that information is relatively straightforward and falls into predictable categories. Thus, for example, online help systems can usefully employ chatbot techniques to identify the area of help that users require, potentially providing a "friendlier" interface than a more formal search or menu system. This sort of usage holds the prospect of moving chatbot technology from Weizenbaum's "shelf ... reserved for curios" to that marked "genuinely useful computational methods".

Among the most notable early chatbots are [ELIZA](#) (1966) and [PARRY](#) (1972). More recent notable programs include [A.L.I.C.E.](#), [Jabberwacky](#) and D.U.D.E ([Agence Nationale de la Recherche](#) and [CNRS](#) 2006). While ELIZA and PARRY were used exclusively to simulate typed conversation, many chatbots now include other functional features, such as games and web searching abilities. In 1984, a book called *The Policeman's Beard is Half Constructed* was published, allegedly written by the chatbot [Racter](#) (though the program as released would not have been capable of doing so).

One pertinent field of AI research is [natural language processing](#). Usually, [weak AI](#) fields employ specialized software or programming languages created specifically for the narrow function required. For example, A.L.I.C.E. uses a [markup language](#) called [AIML](#), which is specific to its function as a [conversational agent](#), and has since been adopted by various other developers of, so-called, [Alicebots](#). Nevertheless, A.L.I.C.E. is still purely based on [pattern matching](#) techniques without any reasoning capabilities, the same technique ELIZA was using back in 1966. This is not strong AI, which would require [sapience](#) and logical reasoning abilities.

Jabberwocky learns new responses and context based on real-time user interactions, rather than being driven from a static database. Some more recent chatbots also combine real-time learning with [evolutionary algorithms](#) that optimise their ability to communicate based on each conversation held. Still, there is currently no general purpose conversational artificial intelligence, and some software developers focus on the practical aspect, [information retrieval](#).

Chatbot competitions focus on the Turing test or more specific goals. Two such annual contests are the [Loebner Prize](#) and The Chatterbox Challenge (the latter has been offline since 2015, however, materials can still be found from web archives).

[DBpedia](#) created a chatbot during the GSoC of 2017. and can communicate through Facebook Messenger.

2.1 Existing System:

In the existing system people used to get information about availability of Movies through News Papers/ Posters/ Cable TV network/ Using search browser. After knowing

details of Movies Available they need to book tickets either calling the theatre/ through web form/ visit the theatre in advance. This manual process booking tickets is a tedious and time-consuming process. We needed an integration such as a chatbot for to cater the needs of consumer through online. For to develop such a process using various software requires purchasing licenses, Incurring more cost.

2.2 Proposed Solution:

The best available solution for developing a chatbot can be using a cloud environment such as IBM cloud. IBM cloud provides various resources for to develop custom software at a low cost. Using IBM cloud we can get Software as a Service(SaaS), Platform as a Service(PaaS) there by reducing the development costs.

The Movie ticket chatbot application has been developed using IBM Watson Assistant, with Node-Red as web access.

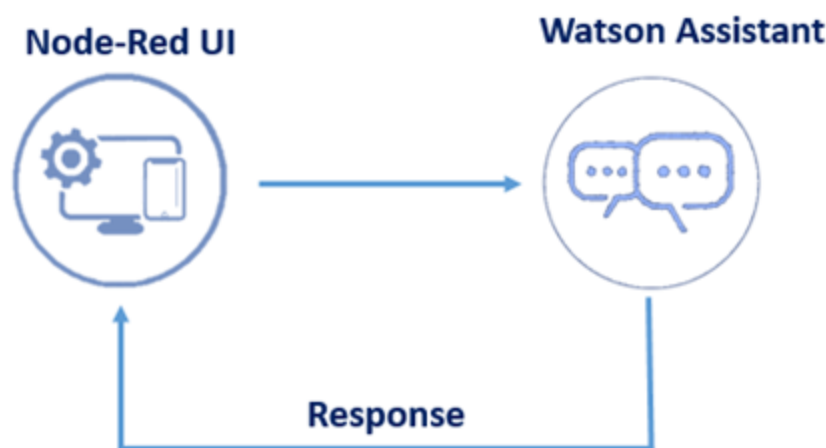
3. THEORITICAL ANALYSIS:

In this project, we will be building a chatbot using Watson assistant. This chatbot should have the following capabilities:

- a. Give the list of movies available.
- b. The Bot should be able to show different show timings
- c. When a movie is selected the bot should show the availability of tickets and their respective prices.
- d. The bot should be in a position to book tickets.

Using IBM cloud Services IBM Watson Assistant and Node-Red

3.1 Block Diagram:



3.2 Software Designing:

To complete this project you need to have an IBM cloud account. You can create this account using this [linkhttps://cloud.ibm.com/login](https://cloud.ibm.com/login).

We will be building a chatbot using Watson assistant. We create Watson assistant service by going to the catalog. If you have already an existing Watson assistant then according to the lite plan of the IBM account you can create 5 skills(chatbots).

Watson Assistant lets you build conversational interfaces into any application, device, or channel. <https://cloud.ibm.com/catalog/services/watson-assistant>

Before you begin

Step 1: Open Watson Assistant

Step 2: Create an assistant

Step 3: Create a dialog skill

Step 4: Add intents from a content catalog

Step 5: Build a dialog

Adding a start node

Testing the start node

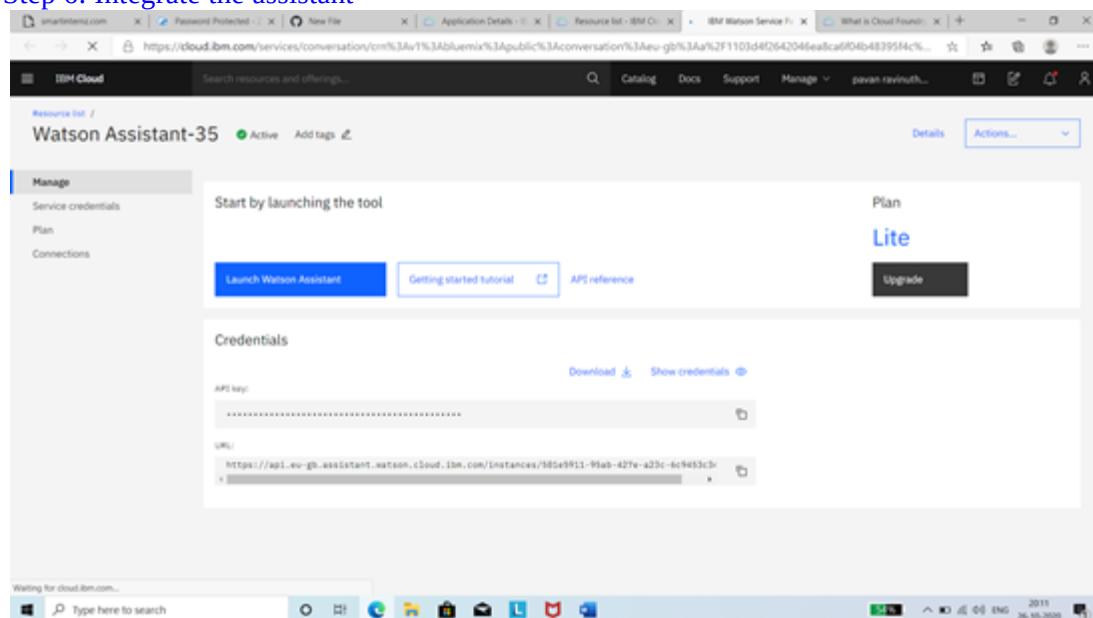
Adding nodes to handle

Intents Testing

Intent recognition

Result of building a dialog

Step 6: Integrate the assistant



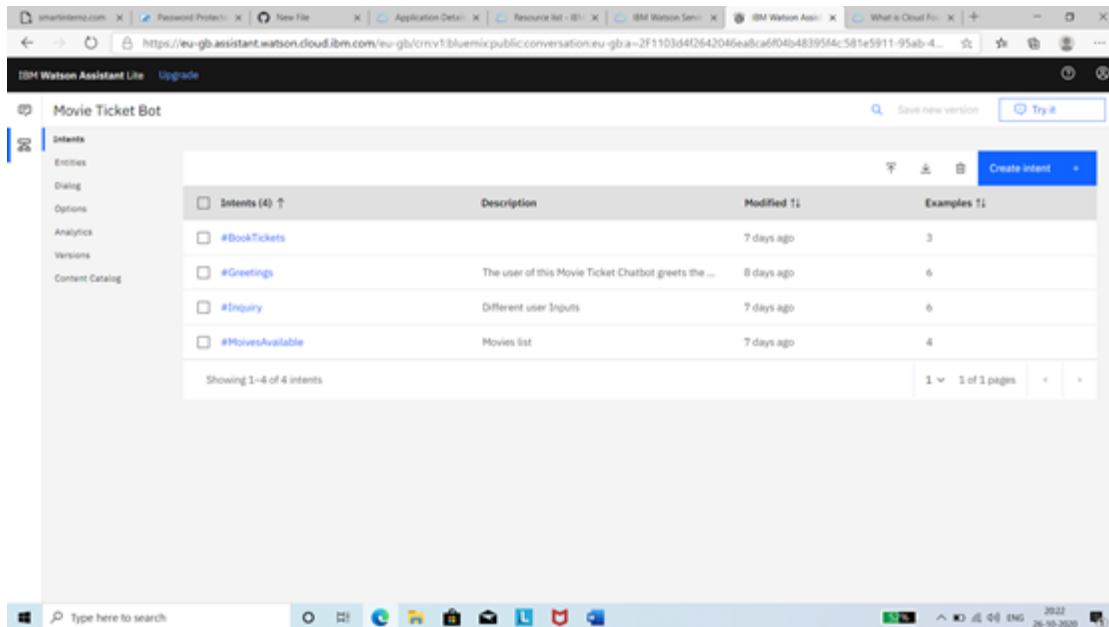
After you create a Watson Assistant service instance, you land on the **Manage** page.

1. Click **Launch Watson Assistant**. If you're prompted to log in, provide your IBM Cloud credentials.

A new browser tab or window opens and Watson Assistant is displayed.

- An assistant named **Movie Ticket Bot** is created for you automatically. An *assistant* is a chatbot. You add skills to your assistant so it can interact with your customers in useful ways.
- A dialog skill named **Movie Ticket Assistant** is added to the assistant for you automatically. A *dialog skill* is a container for the artifacts that define the flow of conversations that your assistant has with your customers.

The dialog skill is opened and the *Intents* page is displayed.



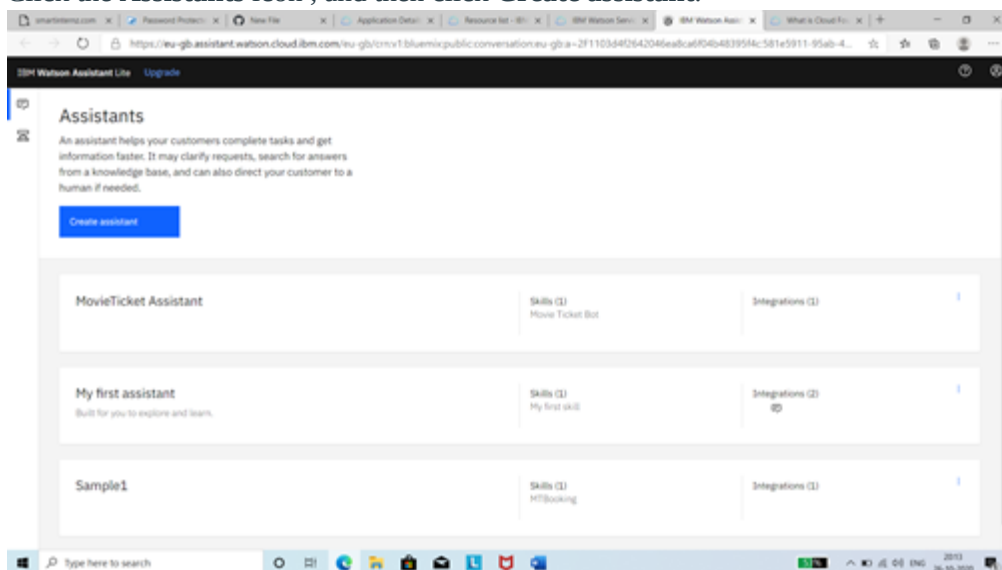
If available in your location, a tour begins that you can step through to learn about the product. Follow the tour; it provides a great overview of the product.

If an assistant and skill are not created automatically, complete Steps 2 and 3. Otherwise, [skip to Step 4: Add intents from a content catalog](#).

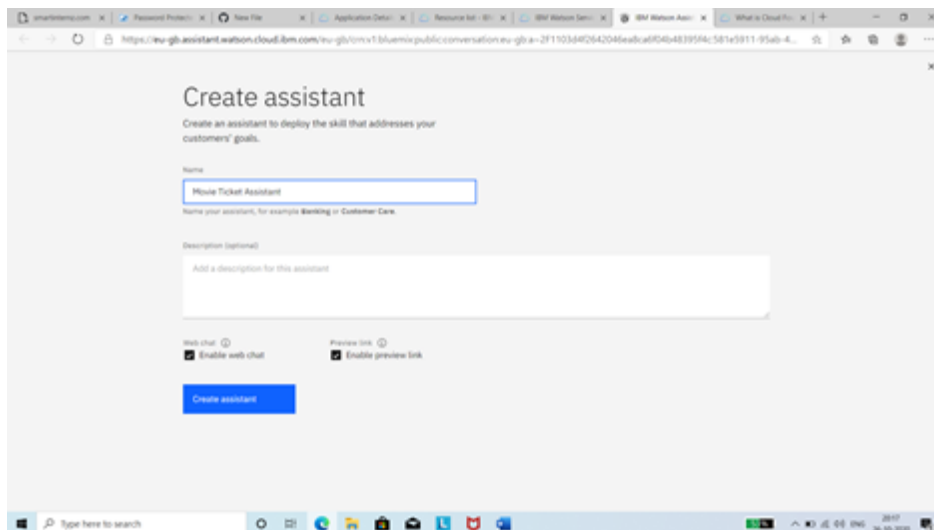
Step 2: Create an assistant

An *assistant* is a cognitive bot to which you add skills that enable it to interact with your customers in useful ways.

1. Click the **Assistants** icon , and then click **Create assistant**.



2. Name the assistant **Movie Ticket Bot**.

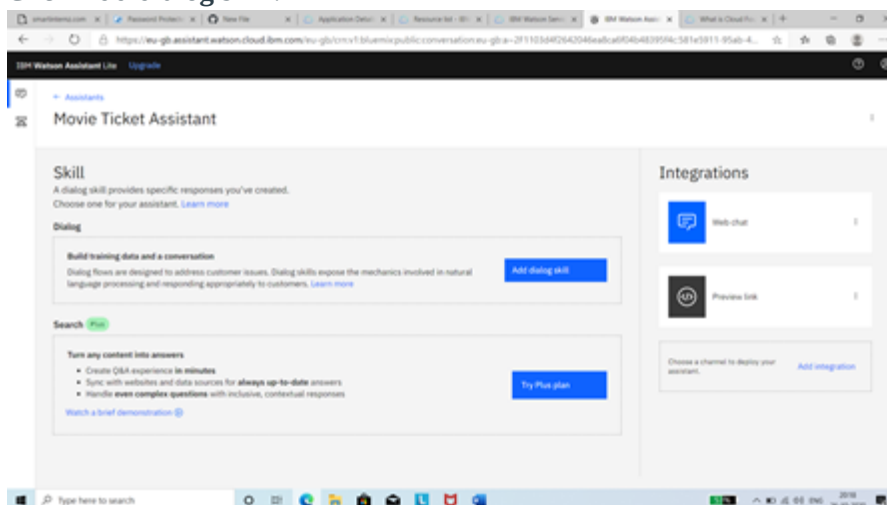


3. Click **Create assistant**.

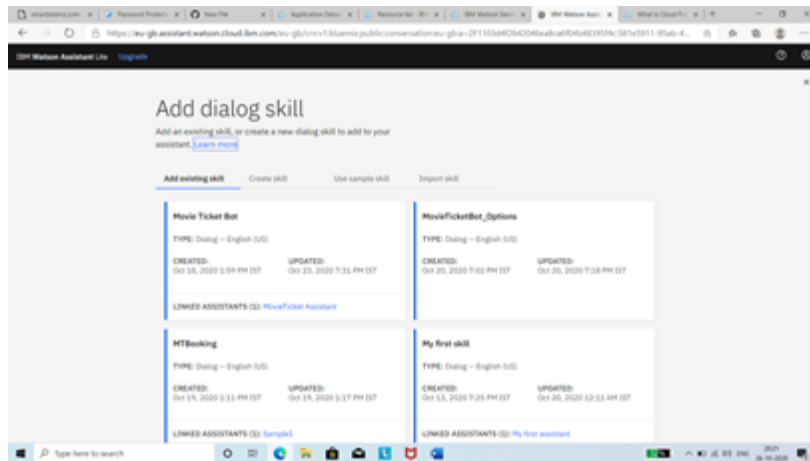
Step 3: Create a dialog skill

A *dialog skill* is a container for the artifacts that define the flow of a conversation that your assistant can have with your customers.

1. Click the *Movie Ticket Bot* tile to open the assistant.
2. Click **Add dialog skill**.



3. Give your skill the name **Movie Ticket Bot**.
4. **Optional.** If the dialog you plan to build will use a language other than English, then choose the appropriate language from the list.
5. Click **Create dialog skill**.

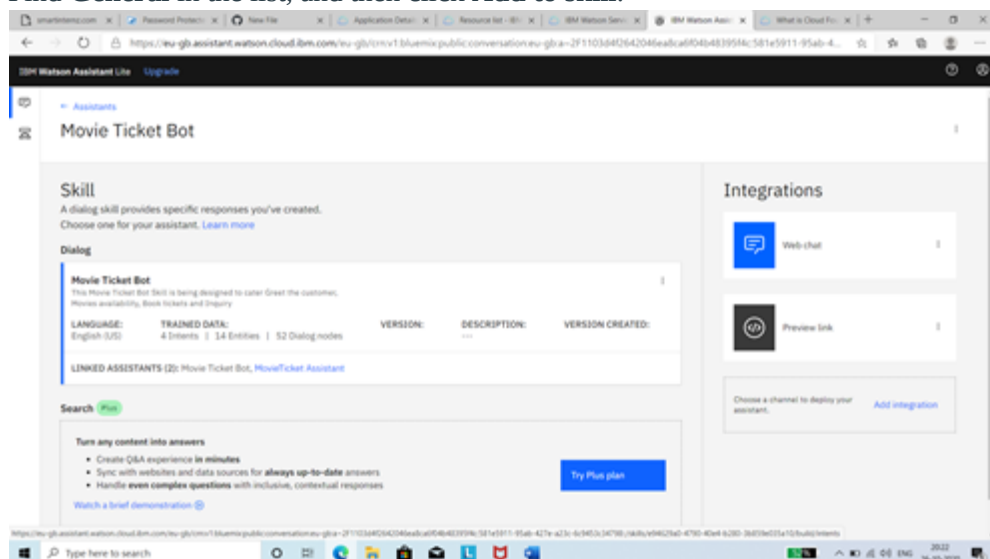


The skill is created and opens to the *Intents* page.

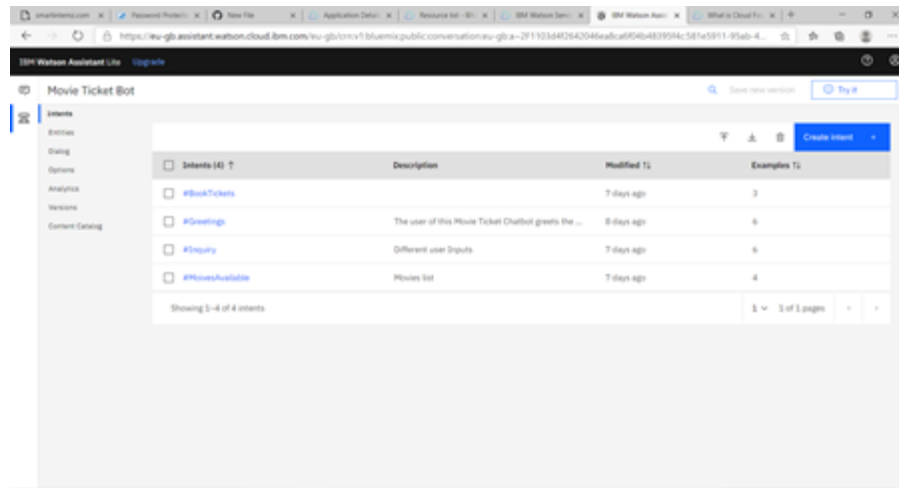
Step 4: Add intents from a content catalog

The Intents page is where you start to train your assistant. In this tutorial, you will add training data that was built by IBM to your skill. Prebuilt intents are available from the content catalog. You will give your assistant access to the **General** content catalog so your dialog can greet users, and end conversations with them.

1. Click **Content Catalog** from the Skills menu.
2. Find **General** in the list, and then click **Add to skill**.



3. Open the **Intents** tab to review the intents and associated example utterances that were added to your training data. You can recognize them because each intent name begins with the prefix `#Greetings`. You will add the `#MoviesAvailable`, `#BookTickets` and `#Inquiry` intents to your dialog in the next step.



You successfully start to build your training data by adding prebuilt content from IBM.

Step 5: Build a dialog

A [dialog](#) defines the flow of your conversation in the form of a logic tree. It matches intents (what users say) to responses (what your virtual assistant says back). Each node of the tree has a condition that triggers it, based on user input.

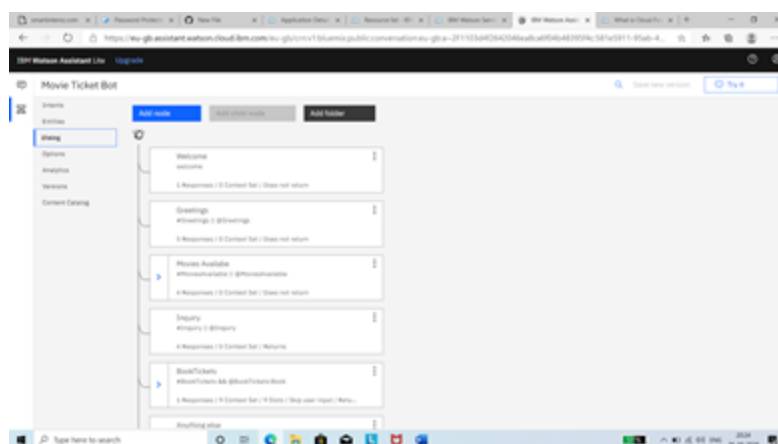
We'll create a simple dialog that handles greeting and ending intents, each with a single node.

Adding a start node

1. From the Skills menu, click **Dialog**.

The following two dialog nodes are created for you automatically:

- **Welcome:** Contains a greeting that is displayed to your users when they first engage with the assistant.
- **Anything else:** Contains phrases that are used to reply to users when their input is not recognized.



2. Click the **Welcome** node to open it in the edit view.

3. Replace the default response with the text, `Welcome! I am a MovieTicket Bot capable of doing the tasks 1.MoviesAvailable 2. Book Tickets 3. Enquiry Please provide an Input.`

The screenshot shows the Dialogflow console interface for editing a dialog node named 'Welcome'. At the top, there's a header with the node name 'Welcome', a 'Customize' button with a gear icon, and a close button 'X'. Below the header, a note states: 'Node name will be shown to customers for disambiguation so use something descriptive.' with a 'Settings' link. The main configuration area is divided into two sections: 'If assistant recognizes' and 'Assistant responds'. In the 'If assistant recognizes' section, the condition 'welcome' is selected. The 'Assistant responds' section shows a 'Text' response type. The response text is: 'welcome! I am a Movie Ticket Bot Capable of helping you to do tasks: 1. Movies Available 2. Book Tickets 3. Inquiry Please Provide an Input:'. Below the response text is a field for 'Enter response variation'. At the bottom, it says 'Response variations are set to sequential. Set to random | multiline' with a link to 'Learn more'.

4. Click to close the edit view.

You created a dialog node that is triggered by the `welcome` condition. (`welcome` is a special condition that functions like an intent, but does not begin with a `#`.) It is triggered when a new conversation starts. Your node specifies that when a new conversation starts, the system should respond with the welcome message that you add to the response section of this first node.

Testing the start node

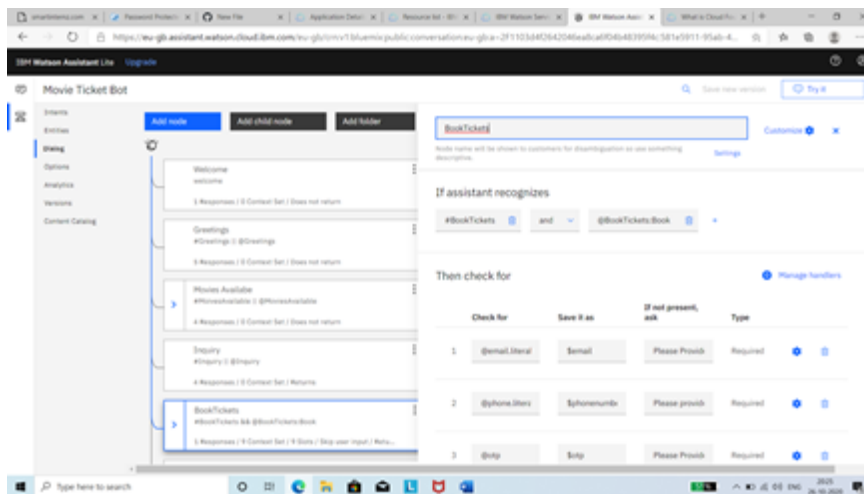
You can test your dialog at any time to verify the dialog. Let's test it now.


- Click the icon to open the "Try it out" pane. You should see your welcome message.

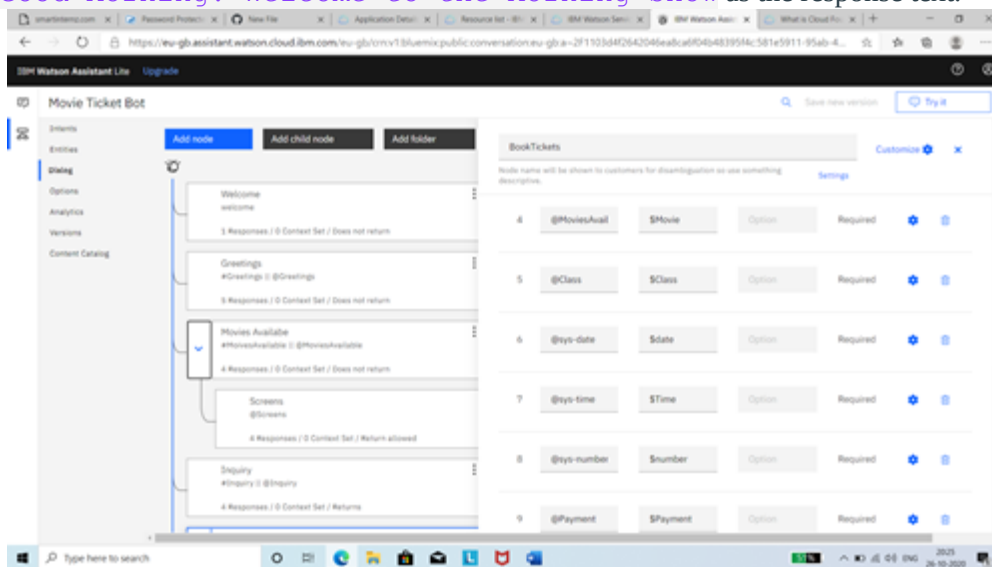
Adding nodes to handle intents

Now let's add nodes between the `Welcome` node and the `Anything else` node that handle our intents.

1. Click **Add node**.
2. In the node name field, type `Greetings`.
3. In the **If assistant recognizes** field of this node, start to type `#Greetings`. Then, select the `#Greetings` option.
4. Add the response text, `Good Morning`




5. Click  to close the edit view.
6. Click **Add node** to create a peer node.
7. Name the peer node **Good Morning** and specify **#Greetings** in the **If assistant recognizes** field.
8. Add **Good Morning! Welcome to the Morning Show** as the response text.



1. Click  to close the edit view.

Testing intent recognition

You built a simple dialog to recognize and respond to both greeting and ending inputs. Let's see how well it works.

1. Click the  to open the "Try it out" pane. There's that reassuring welcome message.
2. In the text field, type **Hi** and then press Enter. The output indicates that the **#Greetings** intent was recognized, and the appropriate response (**Hi! Welcome to the Morning Show**) is displayed.
 - Try the following input: **Good Morning**

- . [Good Afternoon](#)
- . [Good Evening](#)
- . [Good Night](#)
- . [sayonara](#)

Watson can recognize your intents even when your input doesn't exactly match the examples that you included. The dialog uses intents to identify the purpose of the user's input regardless of the precise wording used, and then responds in the way you specify.

Result of building a dialog

That's it. You created a simple conversation with two intents and a dialog to recognize them.

Step 6: Integrate the assistant

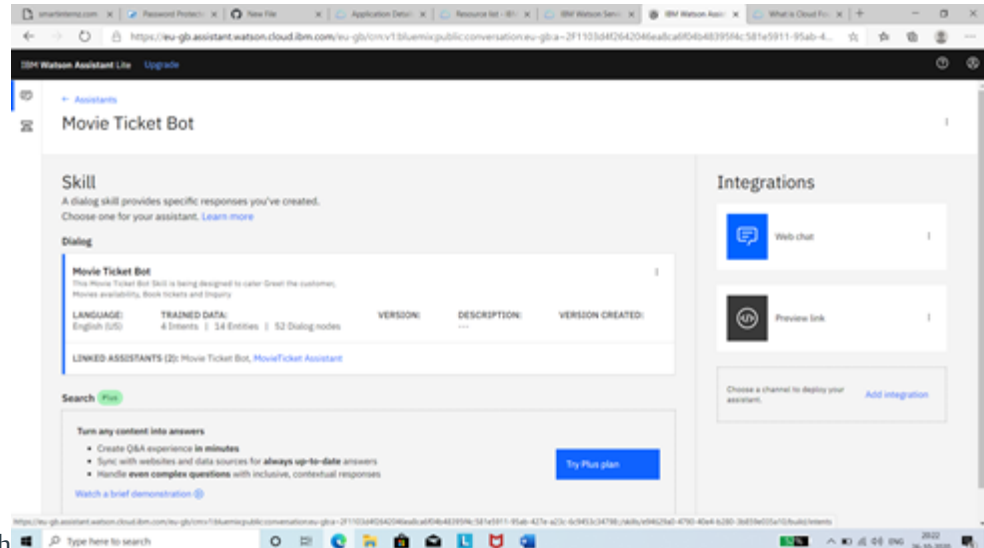
Now that you have an assistant that can participate in a simple conversational exchange, test it.

1. Click the **Assistants** icon to open a list of your assistants.
2. Find the *Movie Ticket Bot* assistant, and open it.

1. Test your assistant with a *Preview link* integration.

The *Preview link* integration is created for you automatically. It builds your assistant into a chat widget that is hosted by an IBM-branded web page. You can open the web page and chat with your assistant to test it out.

2. From the Integrations section, click the **Preview link** tile.

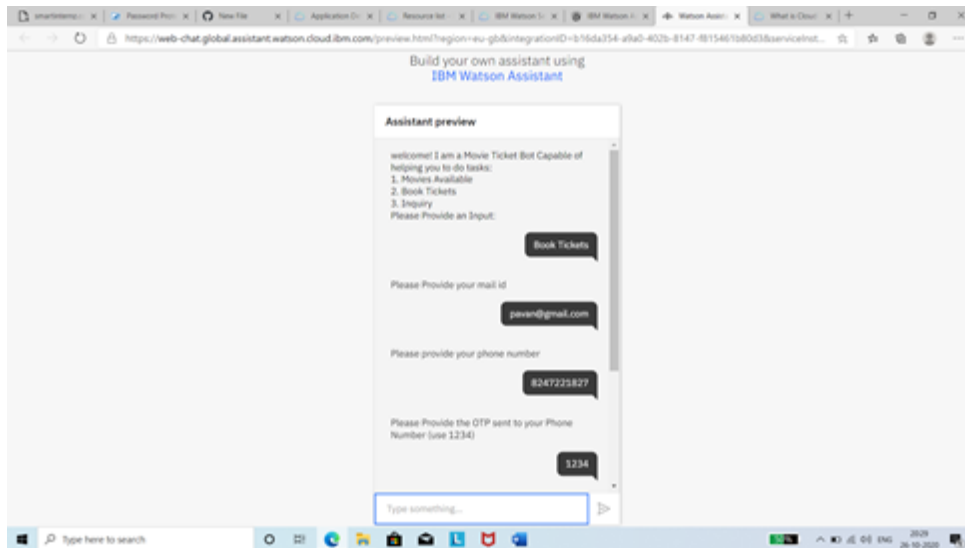


3. Click the URL that is displayed on the page.

The test web page opens in a new tab. You can start submitting message to see how your assistant responds.

With a Lite plan, you can use the service for free. With other plans, you are charged for messages that you submit from the preview link integration. You can review metrics about the test user conversations from the Analytics page. You are not charged for messages that you submit from the "Try it out" pane, and the exchanges you have there are not logged.

4. Type [hello](#) into the text field, and watch your assistant respond.



You can share the URL with others who might want to try out your assistant.

5. After testing, close the web page. Click the X to close the preview link integration page.

CLOUD FOUNDRY:

[Cloud Foundry](#) is the premier industry standard Platform-as-a-Service (PaaS), that ensures the fastest, easiest, and most reliable deployment of cloud-native apps. Cloud Foundry ensures that the build and deploy aspects of coding remain carefully coordinated with any attached services — resulting in quick, consistent and reliable iterating of apps.

Check out the following [video](#) to learn more about all the compute options available on IBM Cloud.

Benefits of Cloud Foundry

Choose your own language - IBM Cloud® Foundry includes runtimes for Java, Node.js, PHP, Python, Ruby, Swift and Go; plus, Cloud Foundry community build packs are also available. Combined with DevOps services, the app runtimes enable a delivery pipeline that automates much of the iterative development process.

Fault tolerant - Runtimes facilitate developing apps as stateless processes that quickly: start and stop, replicate if an instance fails, and duplicate if sustained or increased performance requires.

Extend apps with services - Runtimes link IBM Cloud services to apps as endpoints, giving any instance of an app embedded knowledge of how to manage relevant calls and data. In fact, runtimes manage all linked resources this way: SDKs, APIs (whether made available as cloud services or exposed from within a traditional enterprise as custom services), and also apps themselves when used as resources by other apps.

Access control - Fine grain assignment/dispensing of compute capacity to development teams.

Automatic placement - Apps are automatically placed across multiple data-center PODs for maximum reliability.

Automatic health management - Crashing apps will restart automatically.

Automatic routing - Internet reachable routes are automatically created for your apps.

High availability - Supports full high availability for extremely high app availability.

Automatic deployment scaling - The Auto-Scaling for IBM Cloud service enables you to automatically increase or decrease the compute capacity of your app, to rapidly adjust to dynamic loading needs.

NODE-RED: Flow based programming for Web Hosting

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.

Customizing 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.

To start customising your instance of Node-RED, you can either download the application locally or use IBM DevOps Services to edit and deploy your changes directly.

● **Securing the editor**

When you first ran this application you were presented with some options to secure the editor. To change those options, you can set some environment variables from either the Bluemix console or the `cf` command-line

The environment variables you can set are:

- `NODE_RED_USERNAME` - the username to secure the editor with
- `NODE_RED_PASSWORD` - the password to secure the editor with

- `NODE_RED_GUEST_USER` - set to `true` to allow anonymous users to have read-only access to the editor

Bluemix console

- On the Bluemix console page for this application, go to the 'Runtime' page and then the 'Environment Variables' section
- Add the required user-defined variables
- Click Save and restart your application

cf command-line

- Run the command:

`cf set-env [APPLICATION_NAME] [ENV_VAR_NAME] [ENV_VAR_VALUE]`

● **Enabling Application Metrics for Node.js monitoring**

When you first ran this application you were presented with an option to enable monitoring of your Node-RED flows using the [Application Metrics for Node.js](#) dashboard. To change those options, you can set an environment variable from either the Bluemix console or the `cf` command-line.

When enabled, the [Application Metrics for Node.js](#) dashboard will be available at <https://node-red-mphjj-2020-10-15.mybluemix.net/appmetrics-dash>

The environment variable you can set is:

- `NODE_RED_USE_APPMETRICS` - set to `true` to enable [Application Metrics for Node.js](#) monitoring. Set to `false` to disable.

Bluemix console

- On the Bluemix console page for this application, go to the 'Runtime' page and then the 'Environment Variables' section
- Add the required user-defined variable
- Click Save and restart your application

cf command-line

- Run the command:

`cf set-env [APPLICATION_NAME] [ENV_VAR_NAME] [ENV_VAR_VALUE]`

● **+ Adding new nodes to the palette**

There is a growing collection of additional nodes that can be added to the Node-RED editor. You can search for available nodes on the [Node-RED library](#).

To add a node to the editor you can either use the Palette Manager feature within the editor itself or manually edit the `package.json` file.

Palette Manager

- Within the editor, select the Manage Palette option from the drop-down menu
- Go to the Install tab
- Search for the module you're interested in and click install

Edit package.json

- Edit the file `package.json` and add the required node package to the dependencies section. The format is:

```
"node-red-node-package-name":"x.x.x"
```

Where x.x.x is the desired version number.

● + Upgrading the version of Node-RED

The application's `package.json` is setup to grab the latest stable release of Node-RED.

To trigger an upgrade following a new release being made available:

- Set the `NODE_MODULES_CACHE` environment variable to `false`. You can either do this on your application's Bluemix console page (Runtime -> Environment Variables), or by using the `cf` command-line:

```
cf set-env [APPLICATION_NAME] NODE_MODULES_CACHE false
```

- Trigger a restage of your application. This cannot be done using the Bluemix console, so the `cf` command-line should be used:

```
cf restage [APPLICATION_NAME]
```

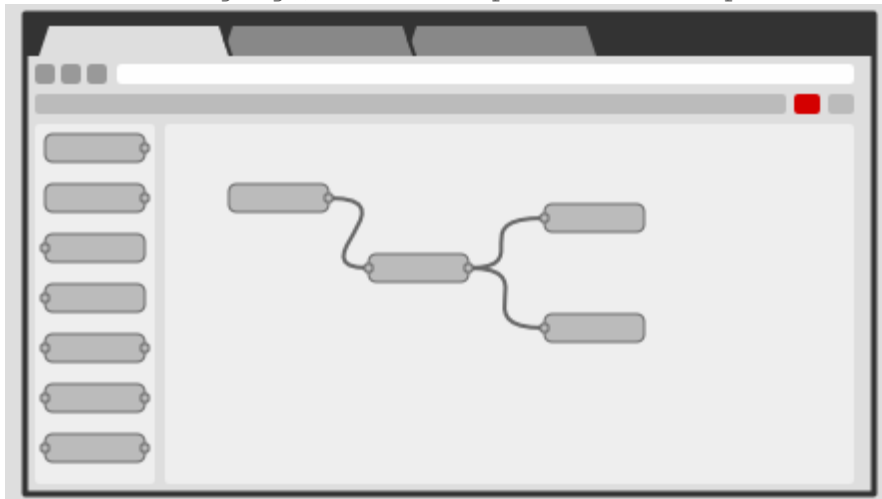
●

+ Changing the static web content

The page you are reading now is served as static content from the application. This can be replaced with whatever content you want in the `public` directory.

● + Remove static web content and serve the flow editor from the root path

In the file `bluemix-settings.js`, delete the `httpStatic` and `httpAdminRoot` entries.



Browser-based flow editing

Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.

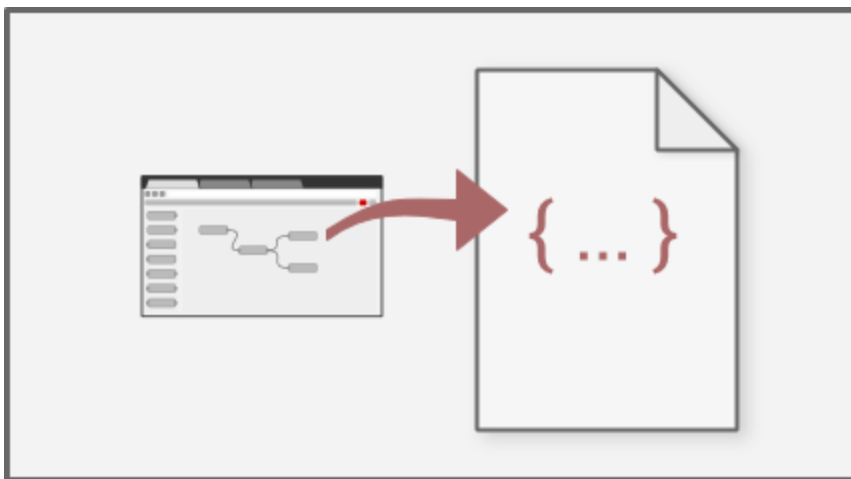
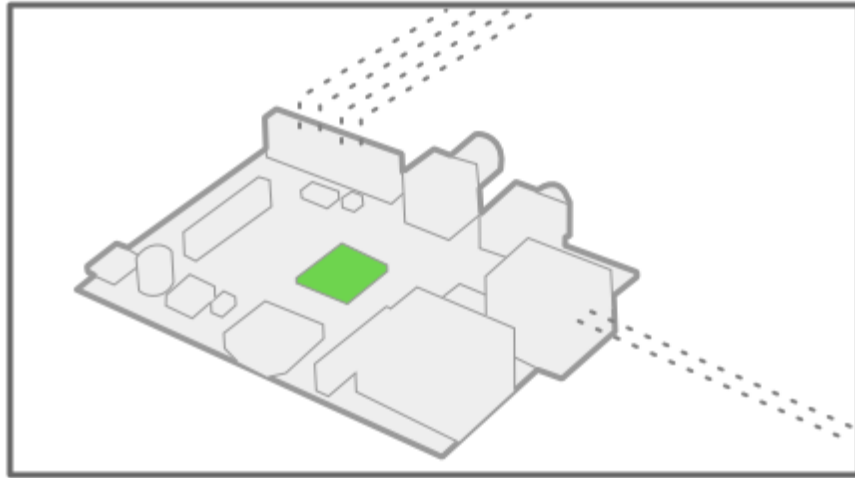
JavaScript functions can be created within the editor using a rich text editor.

A built-in library allows you to save useful functions, templates or flows for re-use.

Built on Node.js

The light-weight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.

With over 225,000 modules in Node's package repository, it is easy to extend the range of palette nodes to add new capabilities.



Social Development

The flows created in Node-RED are stored using JSON which can be easily imported and exported for sharing with others.

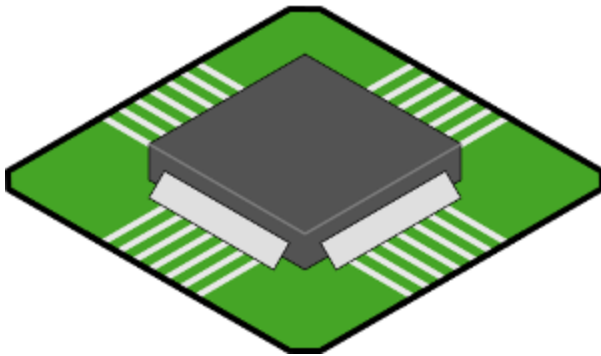
An online flow library allows you to share your best flows with the world.

Get Started: Node-RED is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.



Run locally

- [Getting started](#)
- [Docker](#)



On a device

- [Raspberry Pi](#)
- [BeagleBone Black](#)
- [Interacting with Arduino](#)
- [Android](#)



In the cloud

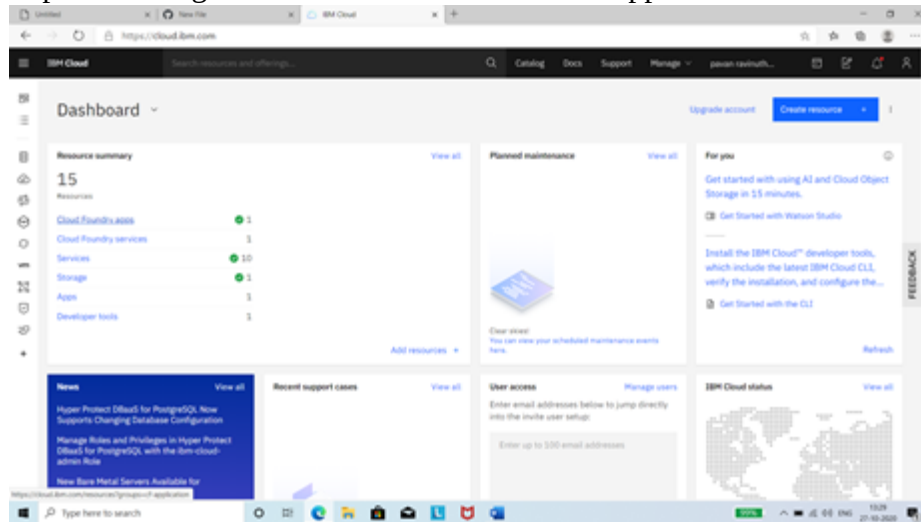
- [IBM Cloud](#)

- [SenseTecnica FRED](#)
- [Amazon Web Services](#)
- [Microsoft Azure](#)

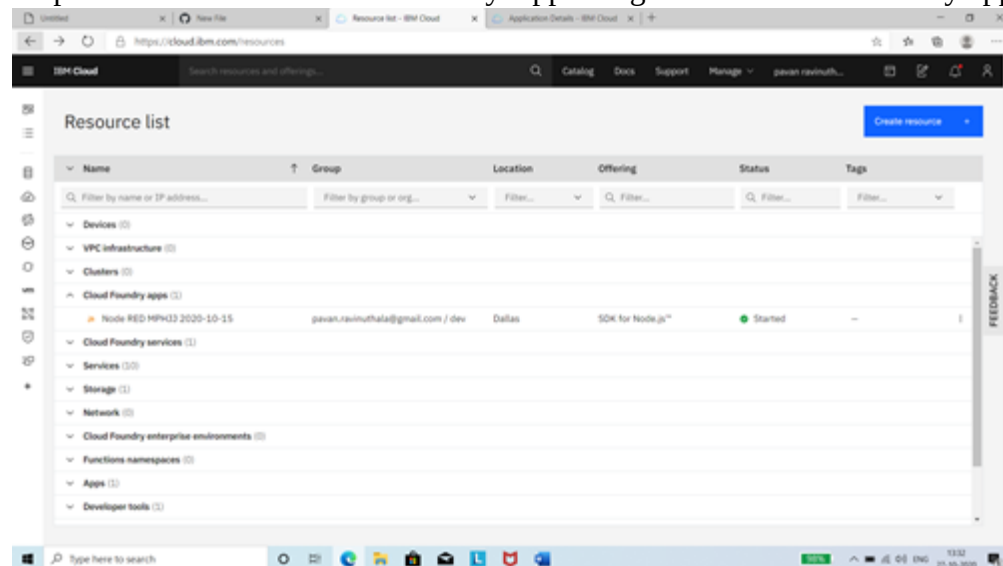
Overview:

For to integrate the Movie Ticket Bot with Web using node-RED we need to follow the below steps.

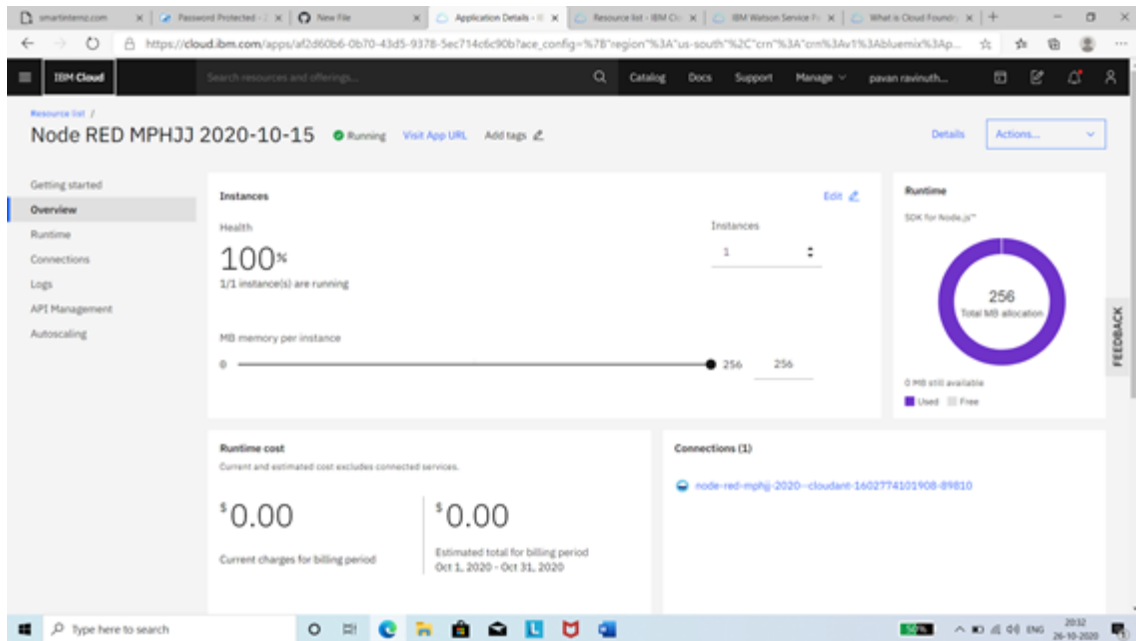
Step1: After login to IBM cloud the dash board appears



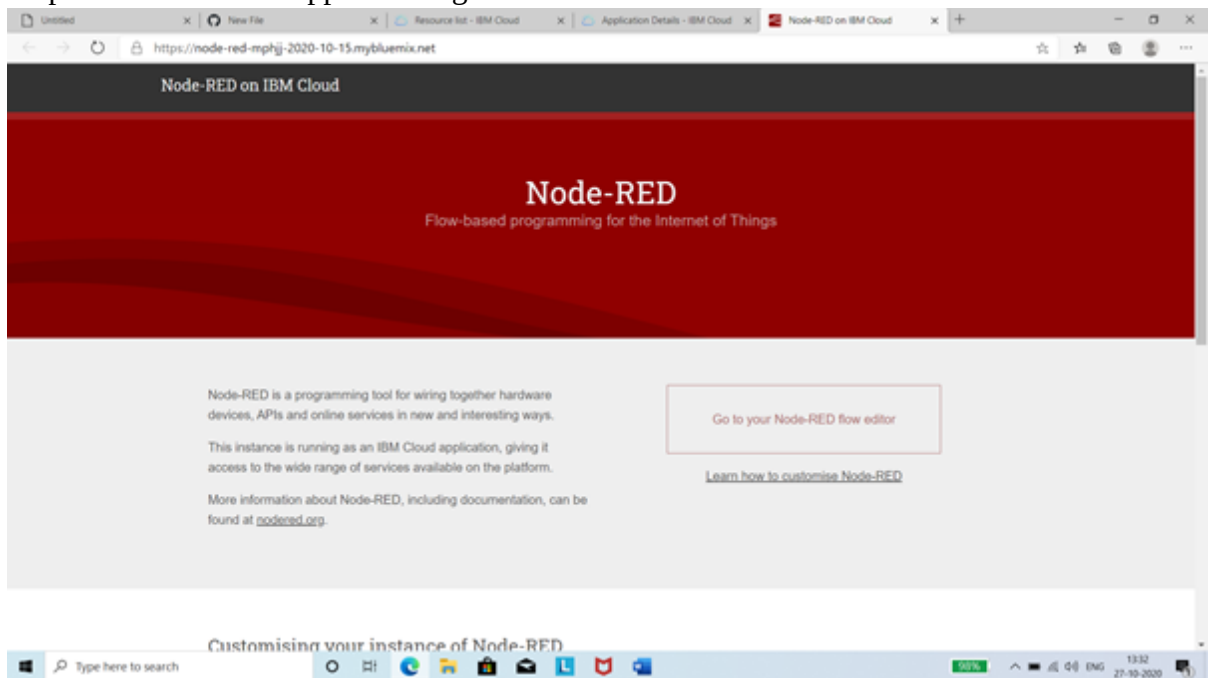
Step2: Double click on Cloud foundry Apps we get a list of cloud foundry apps available.



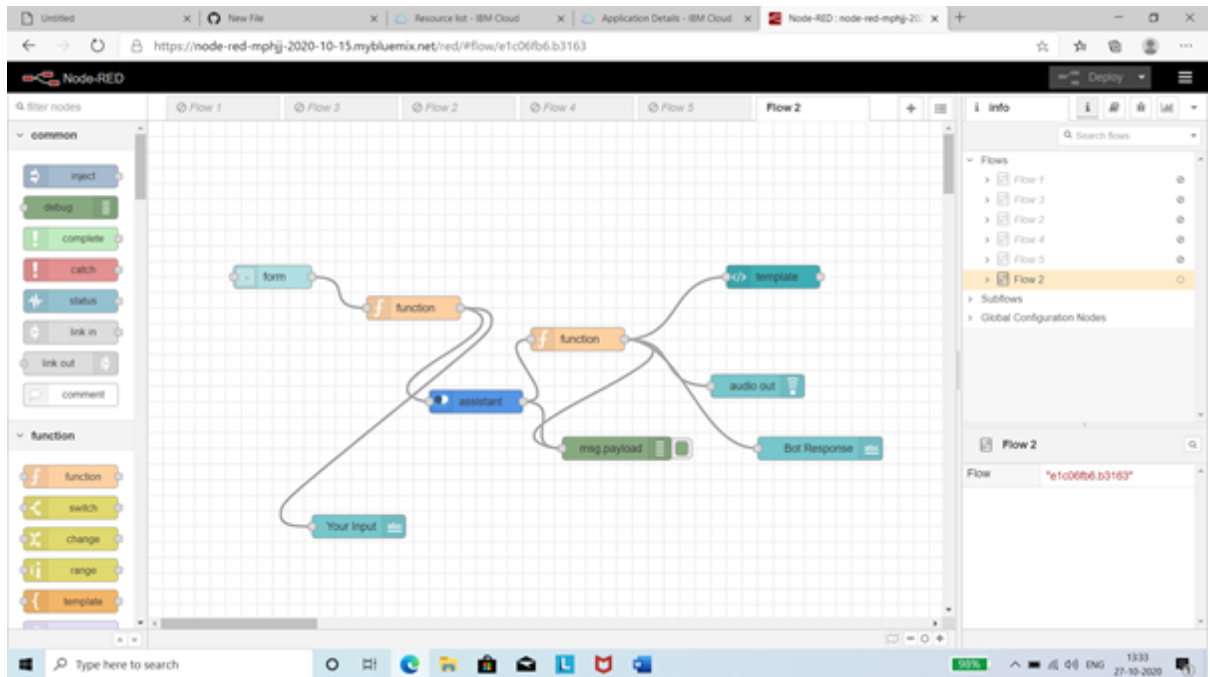
Step3: Double click on Node-RED link we will be displayed with a pre-configured node-RED



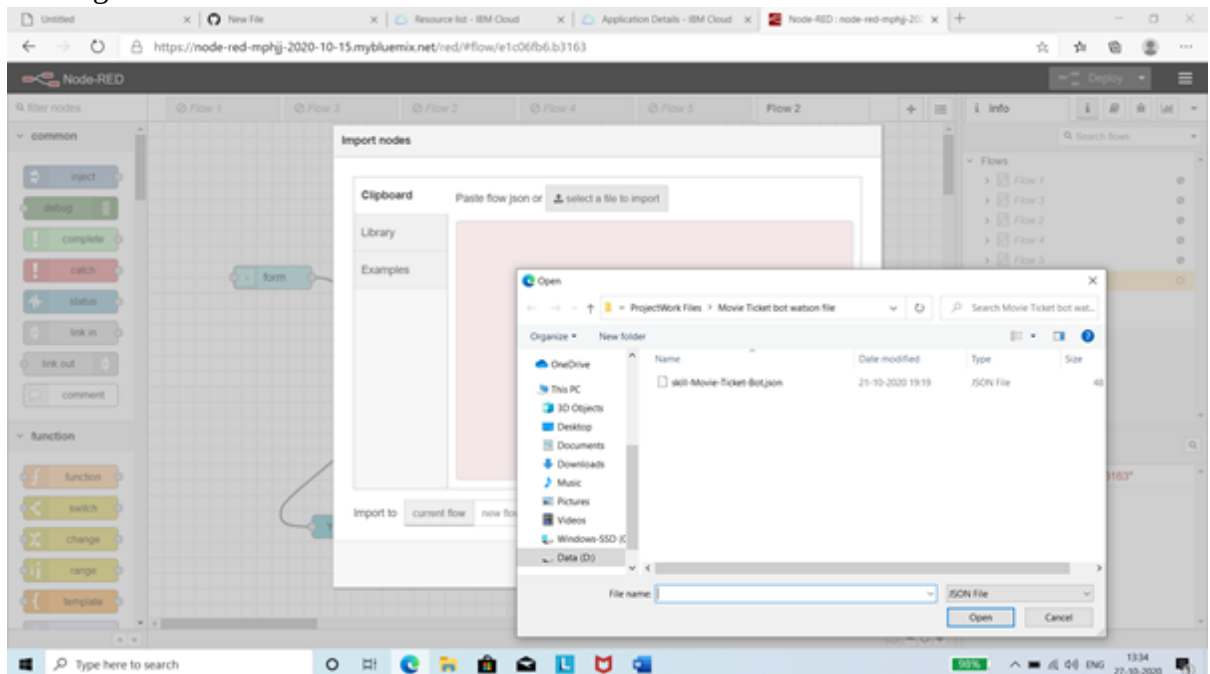
Step4: Click on view App URL to get the below Screen



Step 5: Upon page load we will be displayed with the following screen



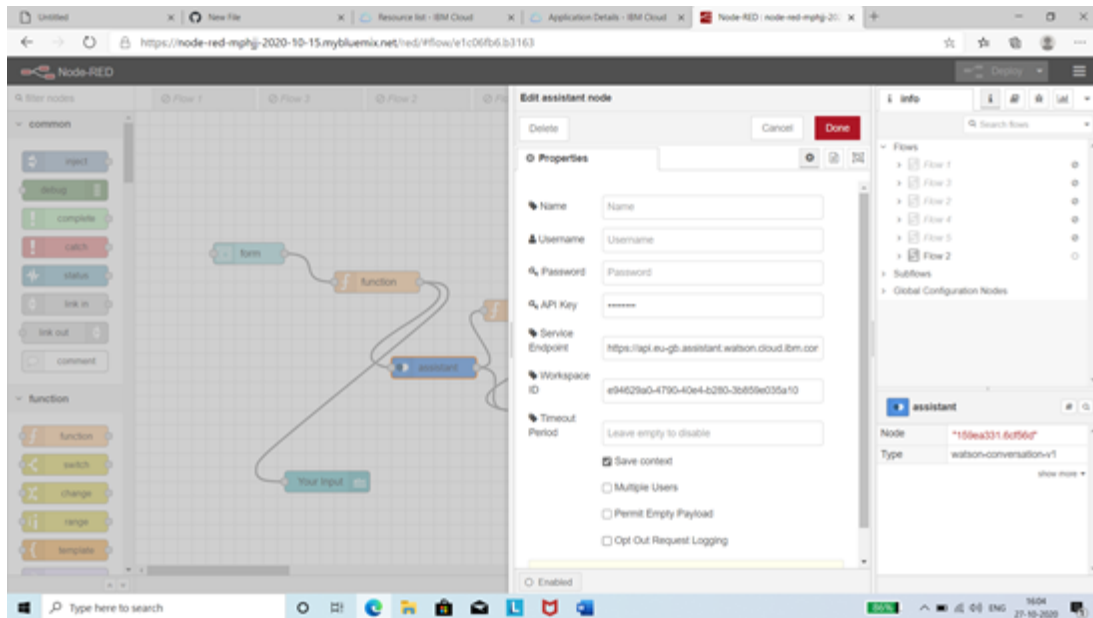
Step6: We will be navigated with the flow diagram that we are working on. Otherwise we can also upload the json file by click on Navigation Pane--> Importà we will be displayed with the following screen. Select the desired json file or copy paste the already existing code format.



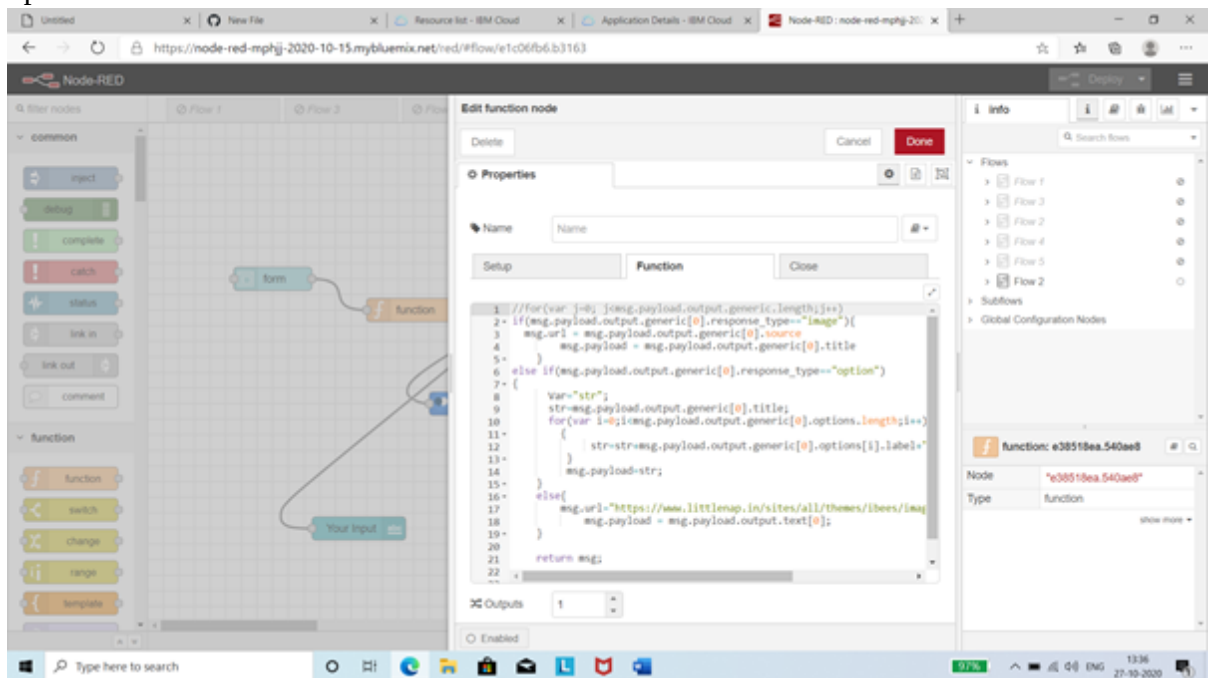
Step7: After designing or loading the required form with controls such as Form, User Input, Function towards input, Assistant, msg.payload, function toward the chatbot side, Template for image, Audio control, bot control and giving links as shown In the figure

Step8: Double click on the Assistant Control and change the API key ,Service end point to the Watson assistant's API key , URL repectively.

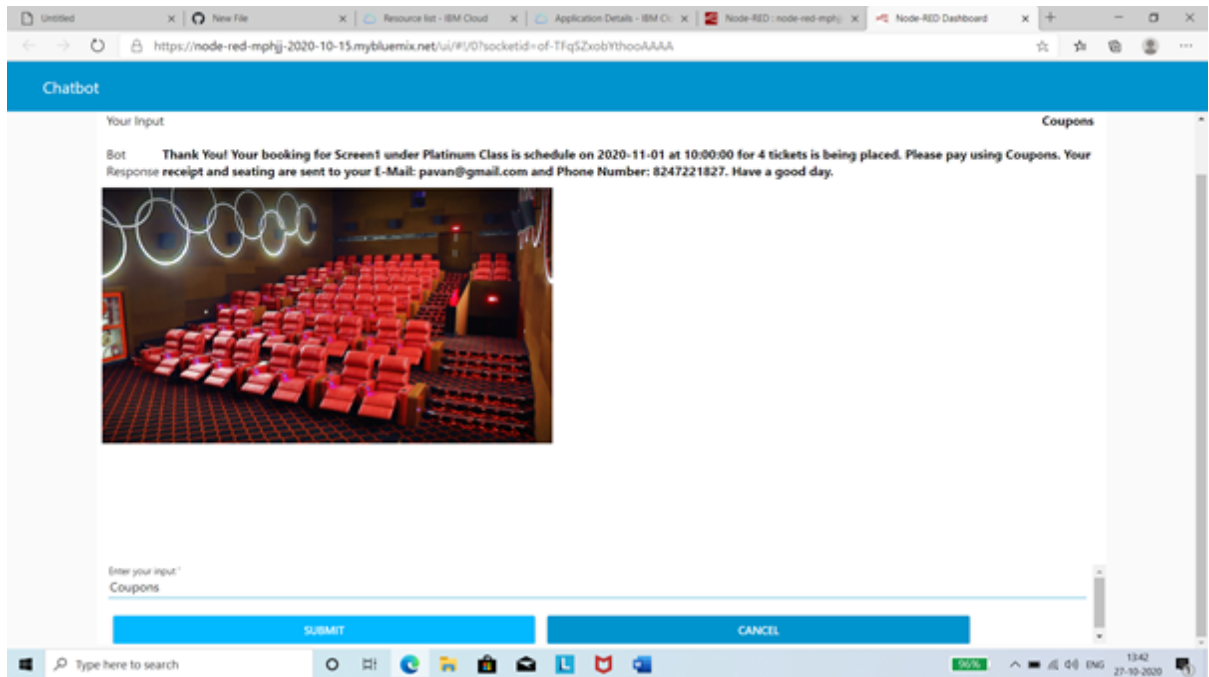
Step9: Change the workspace id to the API key of Movie ticket skill.



Step10: If we are having input type as options we need to write a code for to read the options as shown below:



Step11: click on deploy. We will be getting the output as shown in the figure:



4. EXPERIMENTAL INVESTIGATIONS: While developing the Movie Ticket Bot you come across keen observation of each and every feature of the Watson Assistant right from the creation of Service to Preview of the developed chatbot application.

The main things I observed are as follows:

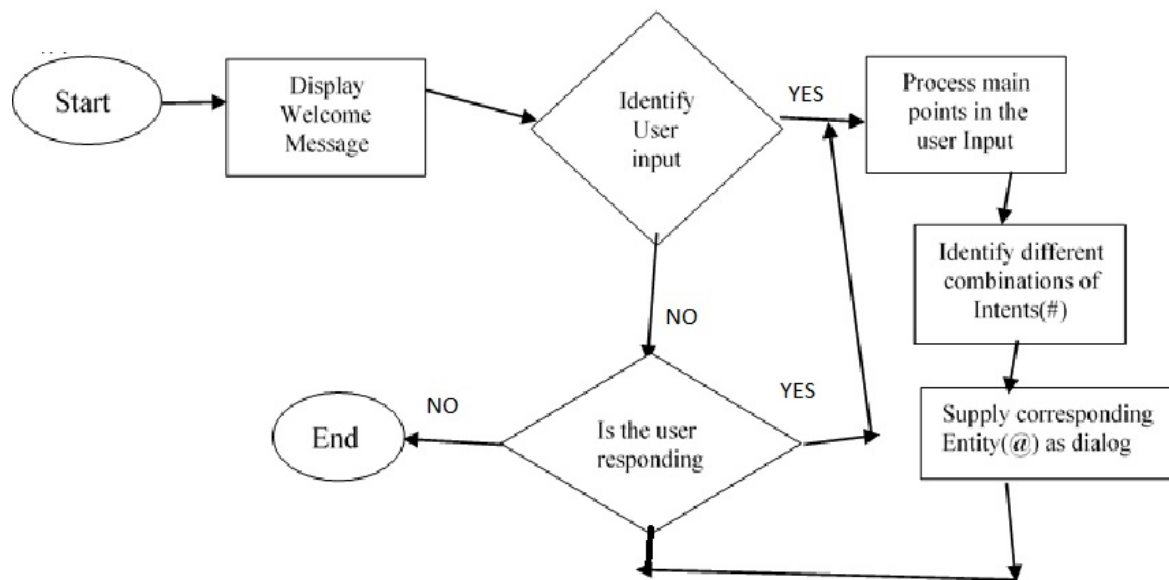
- In slots giving @email.literal for to extract the user email id
- In slots giving @phone.literal for to extract the user password
- Working with sys_date, Sys_time, Sys_number
- Testing the Watson Movie Ticket bot application rigorously

While integration the developed application with the Node-RED web access we come across various controls available to serve our purpose.

Writing code for reading the options is very good experience. The code snippet is as follows:

```
if(msg.payload.output.generic[0].response_type=="option")
{
    Var="str";
    str=msg.payload.output.generic[0].title;
    for(var i=0;i<msg.payload.output.generic[0].options.length;i++)
    {
        str=str+msg.payload.output.generic[0].options[i].label+"    "
    }
    msg.payload=str;
}
```

5. FLOW CHART: The following is the flow chart of Watson Assistant

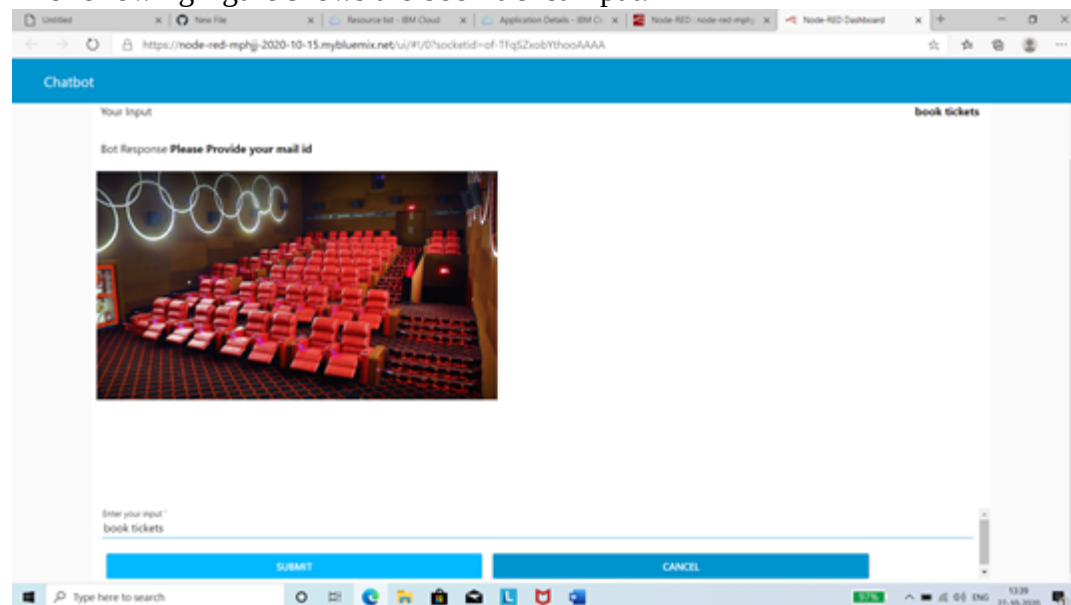


6. RESULT:

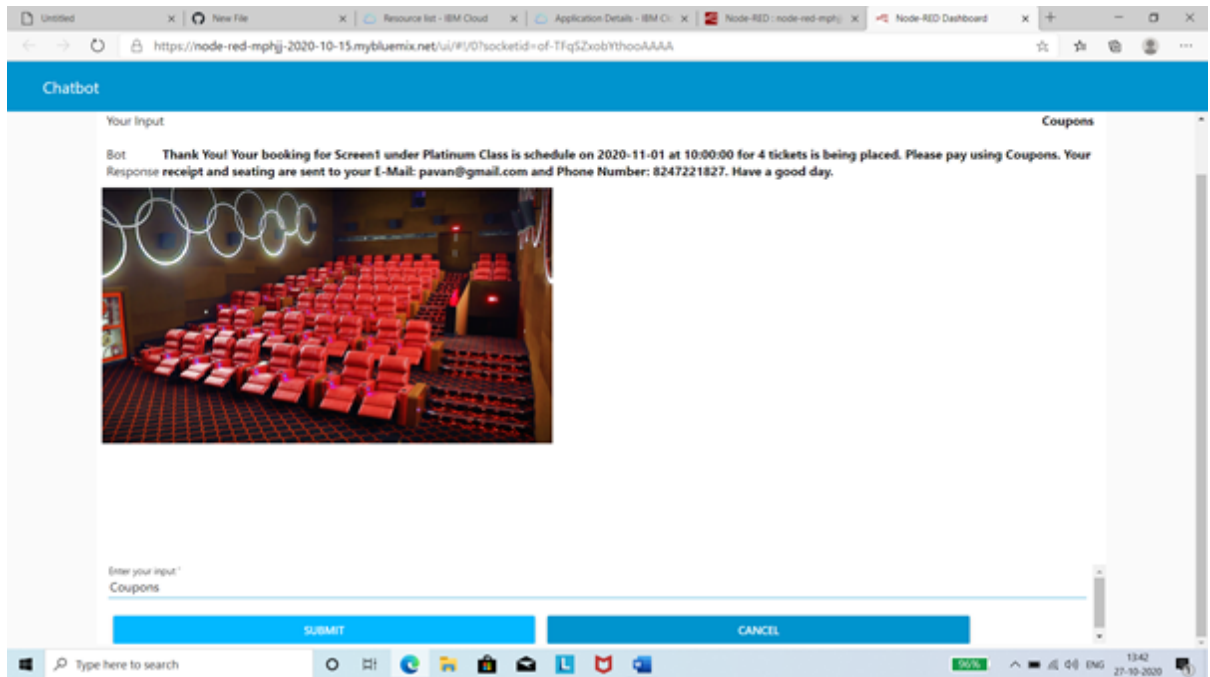
The final expected result is that of web hosting the Movie Ticket Bot and testing the outcome. For web access using node-red please copy the below url link and paste in on the browser:

<https://node-red-mphjj-2020-10-15.mybluemix.net/ui/#!/0?socketid=0pQgkiN8YTc5h-yFAABj>

The following figure shows the book-ticket input:



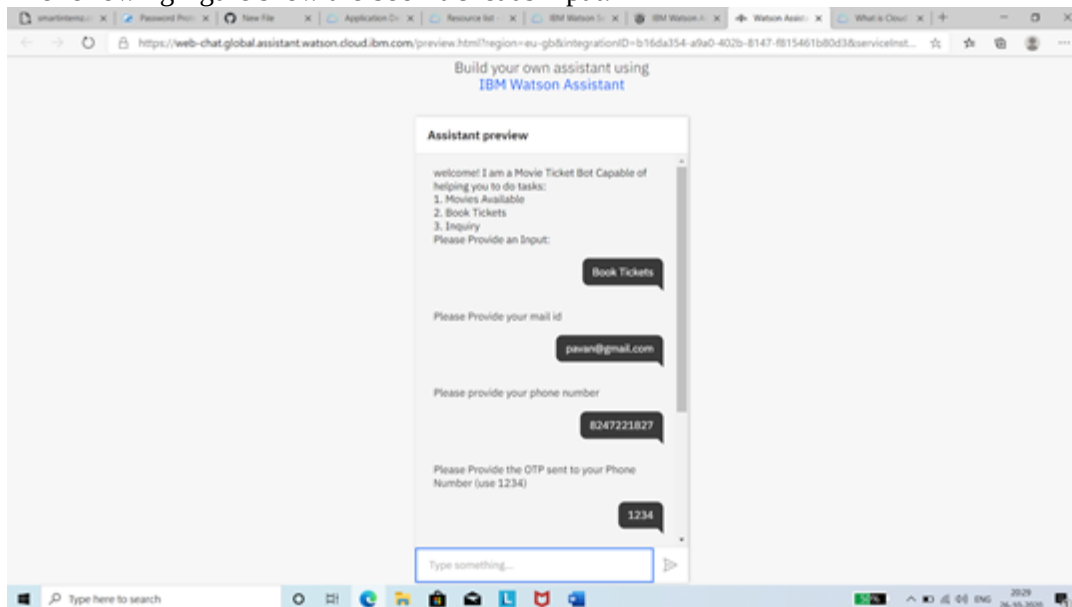
The following input shows the output of Book-Ticket Option:



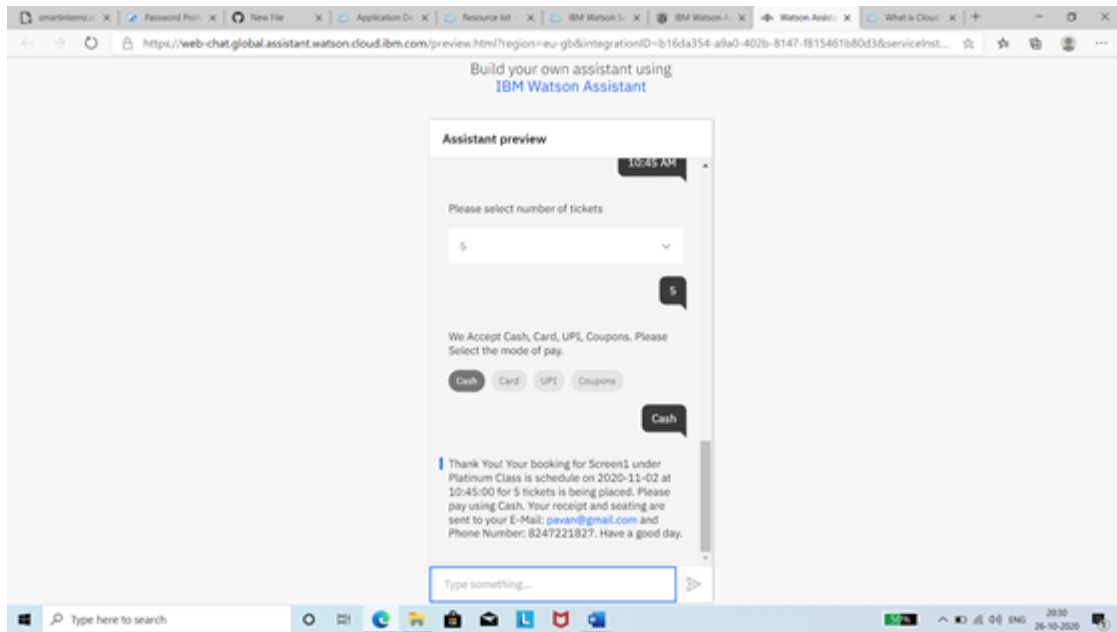
For to see the preview of the Chatbot please copy the below url link and paste in on the browser:

<https://web-chat.global.assistant.watson.cloud.ibm.com/preview.html?region=eu-gb&integrationID=b16da354-a9a0-402b-8147-f815461b80d3&serviceInstanceID=581e5911-95ab-427e-a23c-6c9453c34798>

The following figure show the book-ticket as input:



The following figure show the book-Ticket output:



7. ADVANTAGES OF CHATBOT:

The Movie Ticket Bot that is being developed using IBM cloud is a easy process provided uninterrupted internet access is available. Chatbots are used in [dialog systems](#) for various purposes including customer service, request routing, or for information gathering. While some chatbot applications use extensive word-classification processes, [natural language processors](#), and sophisticated [AI](#), others simply scan for general keywords and generate responses using common phrases obtained from an associated library or [database](#).

Most chatbots are accessed on-line via website popups or through [virtual assistants](#). They can be classified into usage categories that include: [commerce](#) ([e-commerce](#) via chat), education, entertainment, [finance](#), [health](#), [news](#), and [productivity](#).

LIMITATIONS OF CHATBOT:

The creation and implementation of chatbots is still a developing area, heavily related to [artificial intelligence](#) and [machine learning](#), so the provided solutions, while possessing obvious advantages, have some important limitations in terms of functionalities and use cases. However, this is changing over time.

The most common ones are listed below:

- As the database, used for output generation, is fixed and limited, chatbots can fail while dealing with an unsaved query.
- A chatbot's efficiency highly depends on language processing and is limited because of irregularities, such as accents and mistakes.
- Chatbots are unable to deal with multiple questions at the same time and so conversation opportunities are limited.
- Chatbots require a large amount of conversational data to train.
- Chatbots have difficulty managing non-linear conversations that must go back and forth on a topic with a user
- As it happens usually with technology-led changes in existing services, some consumers, more often than not from the old generation, are uncomfortable with

chatbots due to their limited understanding, making it obvious that their requests are being dealt with by machines.

8. APPLICATIONS:

Messaging apps:

Many companies' chatbots run on [messaging apps](#) or simply via SMS. They are used for [B2C](#) customer service, sales and marketing.

In 2016, Facebook Messenger allowed developers to place chatbots on their platform. There were 30,000 bots created for Messenger in the first six months, rising to 100,000 by September 2017.

Since September 2017, this has also been as part of a pilot program on WhatsApp. Airlines [KLM](#) and [Aeroméxico](#) both announced their participation in the testing; both airlines had previously launched customer services on the [Facebook Messenger](#) platform.

The bots usually appear as one of the user's contacts, but can sometimes act as participants in a group chat.

Many banks, insurers, media companies, e-commerce companies, airlines, hotel chains, retailers, health care providers, government entities and restaurant chains have used chatbots to answer simple questions, increase [customer engagement](#), for promotion, and to offer additional ways to order from them.

A 2017 study showed 4% of companies used chatbots. According to a 2016 study, 80% of businesses said they intended to have one by 2020.

As part of company apps and websites

Previous generations of chatbots were present on company websites, e.g. Ask Jenn from [Alaska Airlines](#) which debuted in 2008 or Expedia's virtual customer service agent which launched in 2011. The newer generation of chatbots includes IBM Watson-powered "Rocky", introduced in February 2017 by the [New York City](#)-based [e-commerce](#) company Rare Carat to provide information to prospective diamond buyers.

Chatbot sequences

Used by marketers to script sequences of messages, very similar to an Autoresponder sequence. Such sequences can be triggered by user opt-in or the use of keywords within user interactions. After a trigger occurs a sequence of messages is delivered until the next anticipated user response. Each user response is used in the decision tree to help the chatbot navigate the response sequences to deliver the correct response message.

Company internal platforms

Other companies explore ways they can use chatbots internally, for example for Customer Support, Human Resources, or even in [Internet-of-Things](#) (IoT) projects. [Overstock.com](#), for one, has reportedly launched a chatbot named Mila to automate certain simple yet time-consuming processes when requesting sick leave.^[32] Other large companies such as [Lloyds Banking Group](#), [Royal Bank of Scotland](#), [Renault](#) and [Citroën](#) are now using automated online assistants instead of [call centres](#) with humans to provide a first point of contact. A SaaS chatbot business ecosystem has been steadily growing since the [F8](#) Conference when Facebook's [Mark Zuckerberg](#) unveiled that Messenger would allow chatbots into the app.^[33] In large companies, like in hospitals and aviation organizations, IT architects are designing reference architectures for Intelligent Chatbots that are used to unlock and share knowledge and experience in the organization more efficiently, and reduce the errors in answers from expert service desks significantly. These Intelligent Chatbots make use of all kinds of artificial intelligence like image moderation and [natural language understanding](#) (NLU), [natural language generation](#) (NLG), machine learning and deep learning.

Customer Service

Many high-tech banking organizations are looking to integrate automated AI-based solutions such as chatbots into their customer service in order to provide faster and cheaper assistance to their clients who are becoming increasingly comfortable with technology. In particular, chatbots can efficiently conduct a dialogue, usually replacing other communication tools such as email, phone, or [SMS](#). In banking, their major application is related to quick customer service answering common requests, as well as transactional support.

Several studies report significant reduction in the cost of customer services, expected to lead to billions of dollars of economic savings in the next 10 years. In 2019, [Gartner](#) predicted that by 2021, 15% of all customer service interactions globally will be handled completely by AI. A study by Juniper Research in 2019 estimates retail sales resulting from chatbot-based interactions will reach \$112 billion by 2023.

Since 2016 when [Facebook](#) allowed businesses to deliver automated customer support, e-commerce guidance, content and interactive experiences through chatbots, a large variety of chatbots were developed for the [Facebook Messenger](#) platform.

In 2016, Russia-based Tochka Bank launched the world's first Facebook bot for a range of financial services, including a possibility of making payments.

In July 2016, [Barclays Africa](#) also launched a Facebook chatbot, making it the first bank to do so in Africa.

The France's third-largest bank by total assets [Société Générale](#) launched their chatbot called SoBot in March 2018. While 80% of users of the SoBot expressed their satisfaction after having tested it, Société Générale deputy director Bertrand Cozzarolo stated that it will never replace the expertise provided by a human advisor.

The advantages of using chatbots for customer interactions in banking include cost reduction, financial advice, and 24/7 support.

Healthcare

Chatbots are also appearing in the healthcare industry. A study suggested that physicians in the United States believed that chatbots would be most beneficial for scheduling doctor appointments, locating health clinics, or providing medication information.

[Whatsapp](#) has tied up with the [World Health Organisation \(WHO\)](#) to make a chatbot service that answers users' questions on [Covid-19](#).

[The Indian Government](#) recently launched a chatbot called MyGov Corona Helpdesk, that works through [Whatsapp](#) and helps people access information about the Coronavirus (Covid-19) pandemic.

Certain patient groups are still reluctant to use chatbots. A mixed-methods study showed that people are still hesitant to use chatbots for their healthcare due to poor understanding of the technological complexity, the lack of empathy and concerns about cyber-security. The analysis showed that while 6% had heard of a health chatbot and 3% had experience of using it, 67% perceived themselves as likely to use one within 12 months. The majority of participants would use a health chatbot for seeking general health information (78%), booking a medical appointment (78%) and looking for local health services (80%). However, a health chatbot was perceived as less suitable for seeking results of medical tests and seeking specialist advice such as sexual health. The analysis of attitudinal variables showed that most participants reported their preference for discussing their health with doctors (73%) and having access to reliable and accurate health information (93%). While 80% were curious about new technologies that could improve their health, 66% reported only seeking a doctor when experiencing a health problem and 65% thought that a chatbot was a good idea. Interestingly, 30% reported dislike about talking to computers, 41% felt it would be strange to discuss health matters with a chatbot and about half were unsure if they could trust the advice given by a

chatbot. Therefore, perceived trustworthiness, individual attitudes towards bots and dislike for talking to computers are the main barriers to health chatbots.

Politics

In New Zealand, the chatbot SAM – short for [Semantic Analysis Machine](#) (made by Nick Gerritsen of Touchtech) – has been developed. It is designed to share its political thoughts, for example on topics such as climate change, healthcare and education, etc. It talks to people through Facebook Messenger.

In [India](#), the state government has launched a chatbot for its Aaple Sarkar platform, which provides conversational access to information regarding public services managed.

Toys

Chatbots have also been incorporated into devices not primarily meant for computing, such as toys.

Hello [Barbie](#) is an Internet-connected version of the doll that uses a chatbot provided by the company ToyTalk, which previously used the chatbot for a range of smartphone-based characters for children. These characters' behaviors are constrained by a set of rules that in effect emulate a particular character and produce a storyline.

The [My Friend Cayla](#) doll was marketed as a line of 18-inch (46 cm) dolls which uses [speech recognition](#) technology in conjunction with an [Android](#) or [iOS](#) mobile app to recognize the child's speech and have a conversation. It, like the Hello Barbie doll, attracted controversy due to vulnerabilities with the doll's [Bluetooth](#) stack and its use of data collected from the child's speech.

IBM's [Watson computer](#) has been used as the basis for chatbot-based educational toys for companies such as *CogniToys* intended to interact with children for educational purposes.

Malicious use

Malicious chatbots are frequently used to fill [chat rooms](#) with spam and advertisements, by mimicking human behavior and conversations or to entice people into revealing personal information, such as bank account numbers. They are commonly found on [Yahoo! Messenger](#), [Windows Live Messenger](#), [AOL Instant Messenger](#) and other [instant messaging](#) protocols. There has also been a published report of a chatbot used in a fake personal ad on a dating service's website.

[Tay](#), an AI chatbot that learns from previous interaction, caused major controversy due to it being targeted by internet trolls on Twitter. The bot was exploited, and after 16 hours began to send extremely offensive Tweets to users. This suggests that although the bot learned effectively from experience, adequate protection was not put in place to prevent misuse.

If a text-sending [algorithm](#) can pass itself off as a human instead of a chatbot, its message would be more credible. Therefore, human-seeming chatbots with well-crafted online identities could start scattering fake news that seem plausible, for instance making false claims during a presidential election. With enough chatbots, it might be even possible to achieve artificial [social proof](#).

9. CONCLUSION:

Movie Ticket Bot using IBM Watson assistant Service, Node-RED service is a great experience in getting familiarize with various services provided by IBM cloud environment. The entire development process is successful provided continuous supply of internet connection available for to access the cloud resources. All the deliverables of the Movie Ticket Bot are verified and is working fine as per the project definition.

10.FUTURE SCOPE:

Movie Ticket Bot can be integrated with a Data Base for to store the values stored in the context variables. We need to go through the different services provided under the IBM cloud platform using their documentations.

1. **BIBLIOGRAPHY:**

The following online tutorials had been referred for to develop the project.

- a. Wikipedia on chatbot
<https://en.wikipedia.org/wiki/Chatbot>
- b. YouTube recording of Day2 training program conducted by smartinternz in association with IBM trainer Mahankali Suryatej
<https://www.youtube.com/watch?v=tUBJZfnxeTw>
- c. Online live session of IBM trainer Mahankali SuryaTej using webex
- d. IBM cloud docs of Watson Assistant and Node-RED.
- e. Browsing using Google and Bing