

# Build Your Own Chat Bot

Whenever we consume a product as a customer in any sector, such as telecom, banking services, etc., we are provided a platform where we can reach out to in case of any assistance regarding the product. This platform is generally known as a customer care service. We are provided with an email or a telephone number where we can reach out and the concerned person assists us with the common problems that a customer can face.

As automation is taking over the world, businesses are also looking to automate this customer care service so that there can be less human intervention. But the problem with automating this, is understanding the problem that the customer faces. We as a customer converse with the customer care executive, and may be with a series of question we explain our problem.

In this modern era of AI, we can see that whenever we visit a commercial website, we are greeted by a chat bot popup. This chat bot does exactly that a customer care executive does. We can converse with the bot and explain our problem with the product. The chat bot can even suggest us an appropriate product based on our needs after a conversation.

In this chapter we are going to build one such chat bot which can suggest us an appropriate product based on a conversation.

IBM Watson has a number of cognitive solutions which can cater to a huge plethora of use cases. One such solution is the IBM Watson Assistant.

IBM Watson Assistant provides us with a wonderful toolset to build a chat bot based on any business need. We can even take the chat bot and share it or deploy it in a web application.

Before jumping in right into building the bot, let us first understand the building blocks that we will be using to create a complete chat bot.

## What is Watson Assistant?

Watson Assistant is one of the several services provided by IBM Watson to create a Chat bot specific to any range of business requirements. This tool does not require any sort of programming, rather we can set up logic using its building blocks so as to suit our needs. The chat bot we will create is known as an **assistant**.

IBM Watson Assistant also allows our assistant to publish in social media such as Facebook, publish as an independent web application, or integrate with our own web resources using the wonderful API it provides.

An assistant is built of the following building blocks:

- Intents
- Entities
- Dialogs

Let us look at each of them one by one.

## Intents

Intents are the objectives or goals that your users will want to accomplish at any given time. In other words, intents are what the chat bot can do or address. You can make a list of all the goals that you anticipate your user will want the chat to address for them and they become the intents of the assistant. For example, if one of the usefulness of a weather assistant is to answer what is the humidity percentage, then *ask\_humidity* can be kept as an intent and the intent should contain some sample utterances. Utterances are some of the ways in which the user should ask the assistant, like “What is the humidity percentage value today?” or “Can you please tell me how much the humidity percentage is today?”

## Entities

Entities are terms which are used in context of a particular intent to differentiate the flow of an intent. For example, a town can be an entity which can be used to identify which town’s humidity does the user enquires.

## Dialogs

A dialog can be thought of as a container which captures the intents and entities and directs them to another dialog based on the condition defined in them. A set of dialogs are connected in the form of a tree and forms a **Dialog Skill**. A **dialog skill** contains the training data and logic that enables an assistant to help your customers.

Now Let us get our hands on into building a nice chat bot.

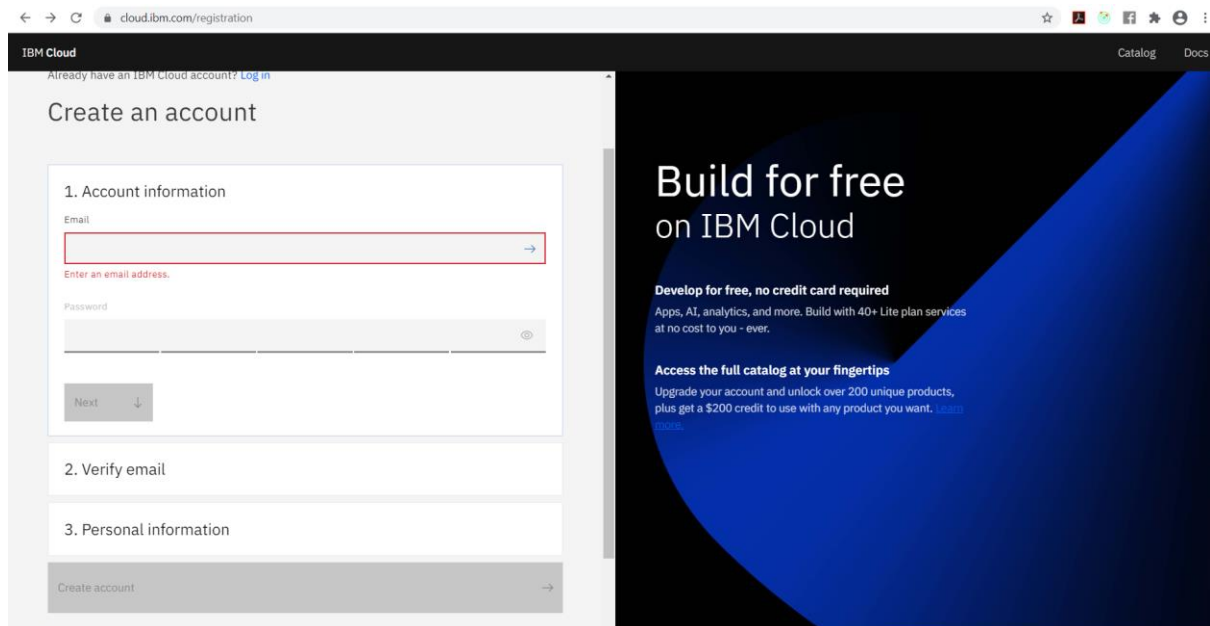
## First Ever Project - The Chat bot

As this is the first time you will be doing something in IBM Watson, let us first prepare ourselves to get into the jungle of IBM Watson.

Let us first get comfortable with this awesome tool- IBM Watson Assistant. So open up your favourite browser and get started.

## Create IBM ID

As a first step, we need to set up an account in IBM Cloud. This one is pretty easy. Go to <https://cloud.ibm.com/registration> and set up your account by providing your email id and a password, verifying email and entering your personal account information.



The screenshot shows a web browser window with the URL [cloud.ibm.com/registration](https://cloud.ibm.com/registration). The page is titled "IBM Cloud" and has a navigation bar with "Catalog" and "Docs" links. The main content area is divided into two sections. On the left, there is a "Create an account" form with the following steps:

- 1. Account information**
  - Email: A text input field with a red border and a blue arrow icon.
  - Enter an email address.
  - Password: A text input field with a blue arrow icon.
  - Next: A button with a blue arrow icon.
- 2. Verify email**
- 3. Personal information**

At the bottom of the form, there is a "Create account" button with a blue arrow icon. On the right, there is a promotional banner with a blue background and white text:

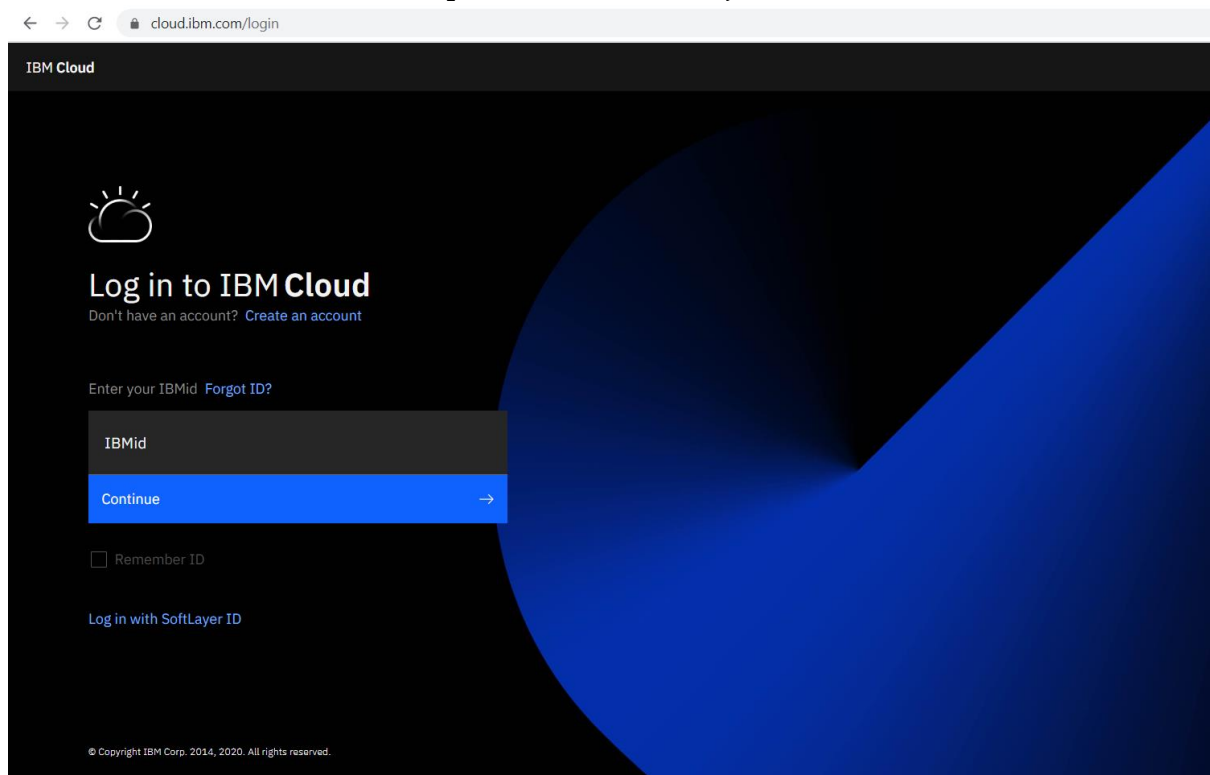
**Build for free**  
on IBM Cloud

**Develop for free, no credit card required**  
Apps, AI, analytics, and more. Build with 40+ Lite plan services at no cost to you - ever.

**Access the full catalog at your fingertips**  
Upgrade your account and unlock over 200 unique products, plus get a \$200 credit to use with any product you want. [Learn more.](#)

[Figure 2.1]

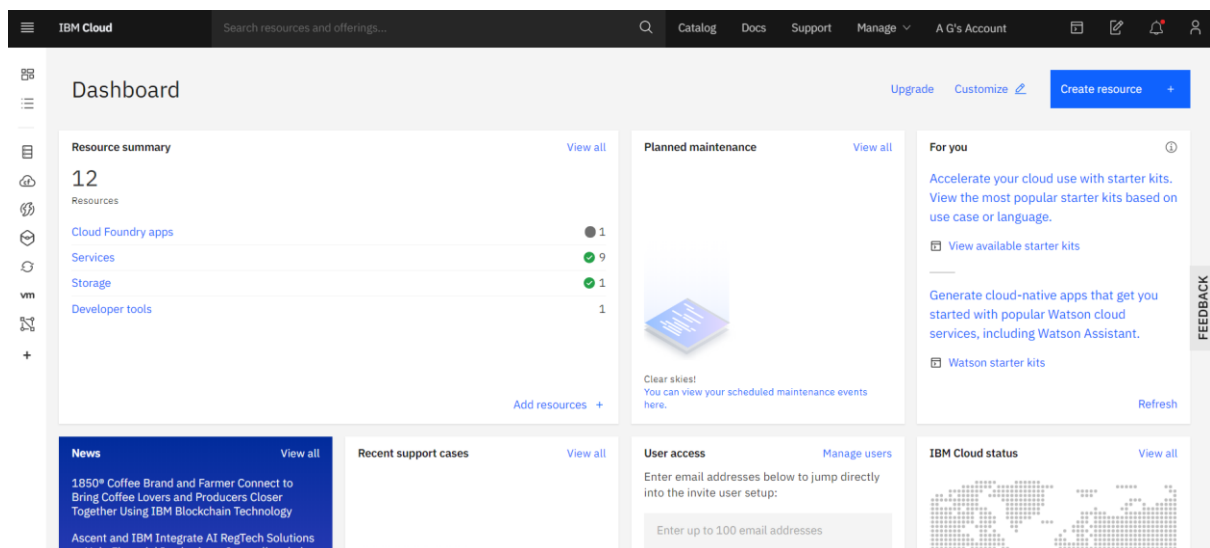
Once done with setting up an IBM ID go to <https://cloud.ibm.com/login> and provide the email ID used to create the previous credential just created. That is the IBM ID.



[Figure 2.2]

Next enter the password and you are in.

Let us look at how an IBM Cloud Account looks like. This is the place where all you IBM Cloud resources are summarized.



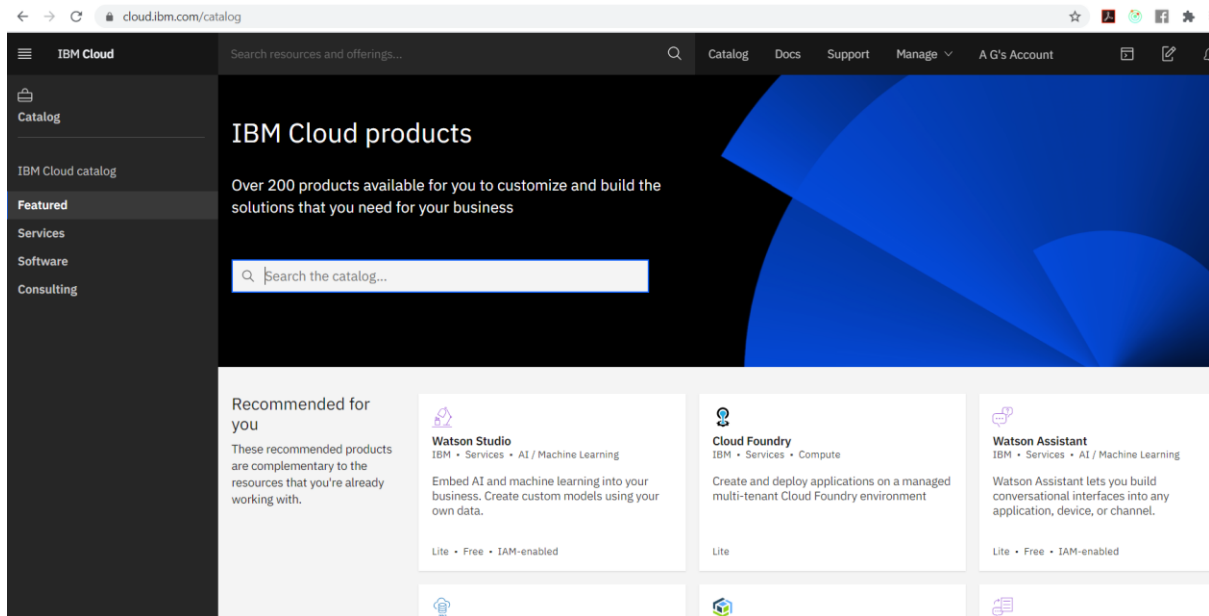
[Figure 2.3]

Next click on the Catalog tab on the top bar just after the search bar and all your resources appear in a new page.



## Create a Resource

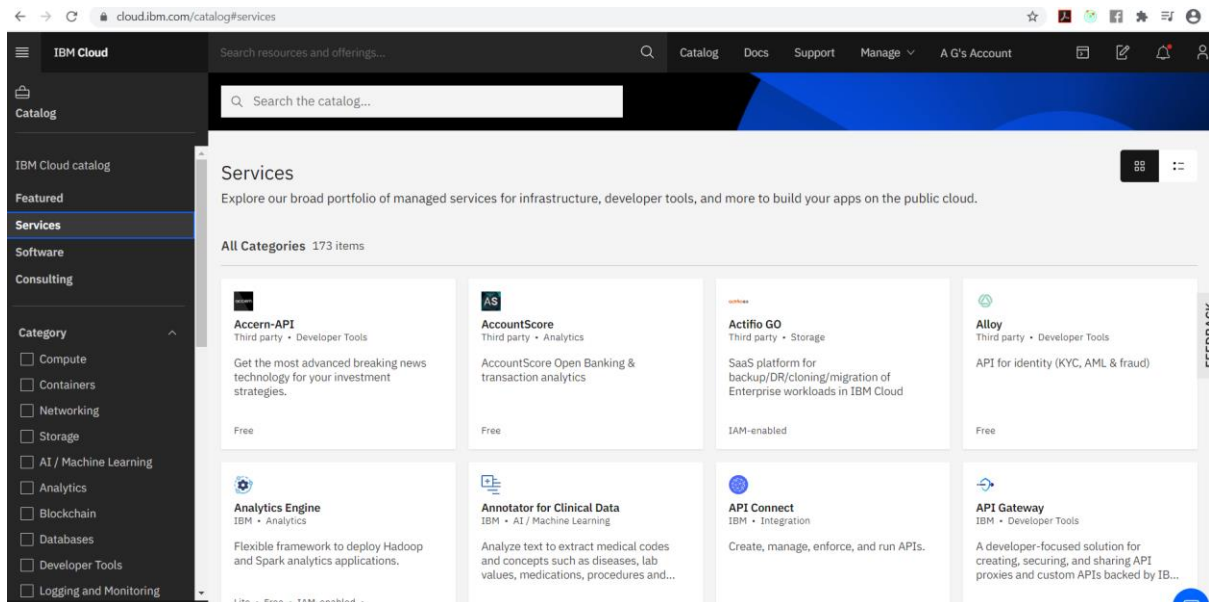
In Figure 2.3, click on the Create Resource button on the top right corner.



[Figure 2.5]

This page initially shows some recommended and featured services as you can see on the left side bar. As you can see it already shows the Watson Assistant Resources.

If you want an extensive list click on Services on the left side bar. This would show a categorical list of all the resources one can use.



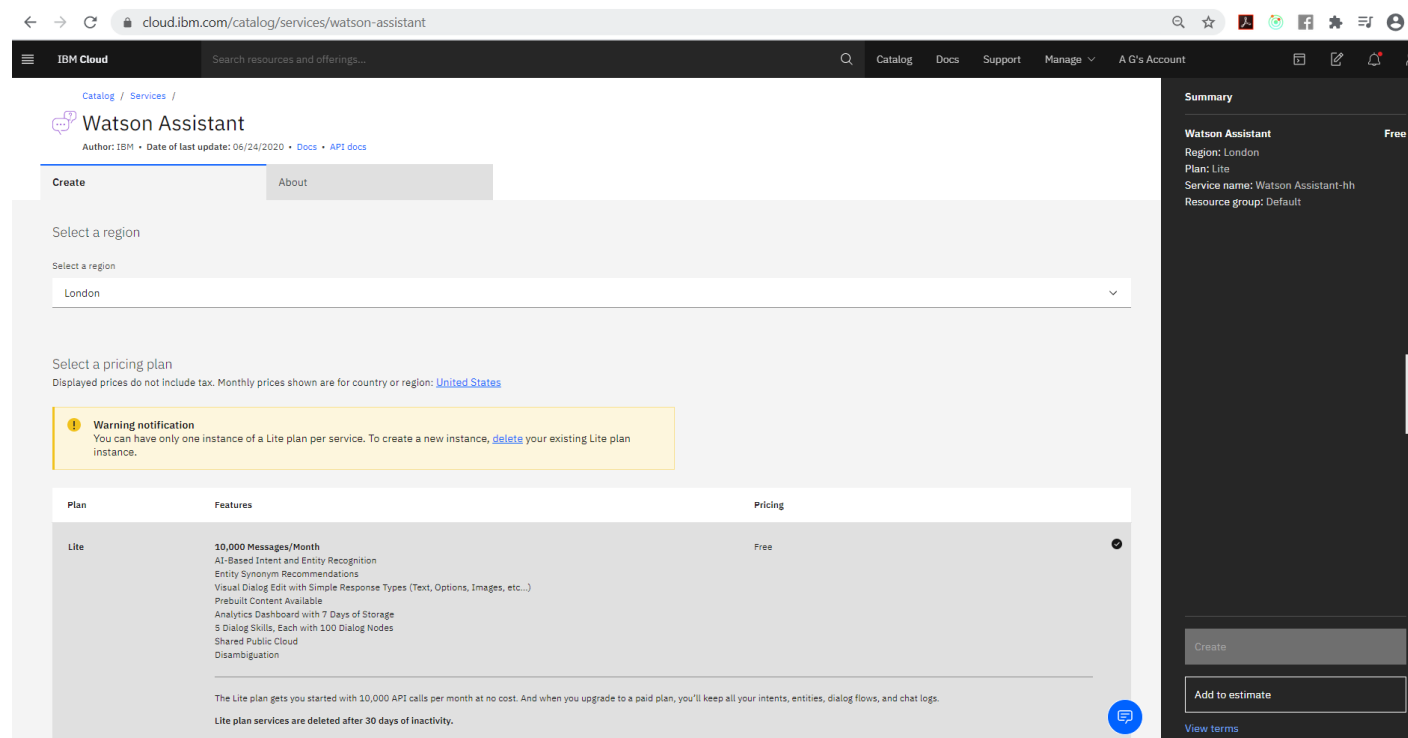
[Figure 2-6]

If you click on the check box beside AI / Machine Learning in the category group on the left bar, it will show you only the list of AI/Machine Learning resources. Watson Assistant is an AI/Machine Learning resource (supposed to be the first resource on the AI/Machine Learning List). So you can click on Watson Assistant card either here

or on the Featured Resources (if it comes up) to create your Watson Assistant Resource.

Also, at any point of time you can type 'assistant' on the search bar that appear on every page and the Watson Assistant card shows up.

Once you click on the Watson Assistant card from any of the resource lists, you will be taken to the Create Watson Assistant page.



[Figure 2.7]

In this page you get a list of pricing for resource creation. Now as a default a user is kept in a Lite account. All services you can create in Lite account are offered free which is pretty good for research and development. But in any case if you want to go for the ones which are not free you will be prompted to upgrade your account and you will be asked to provide payment information. Once you upgrade your account, you can use those.

Although one downside of having a free offering is that you can have only one instance of the service. As you can see from the figure 2.7, since I already have an assistant created from my Lite account, so it is giving me a warning stating the same.

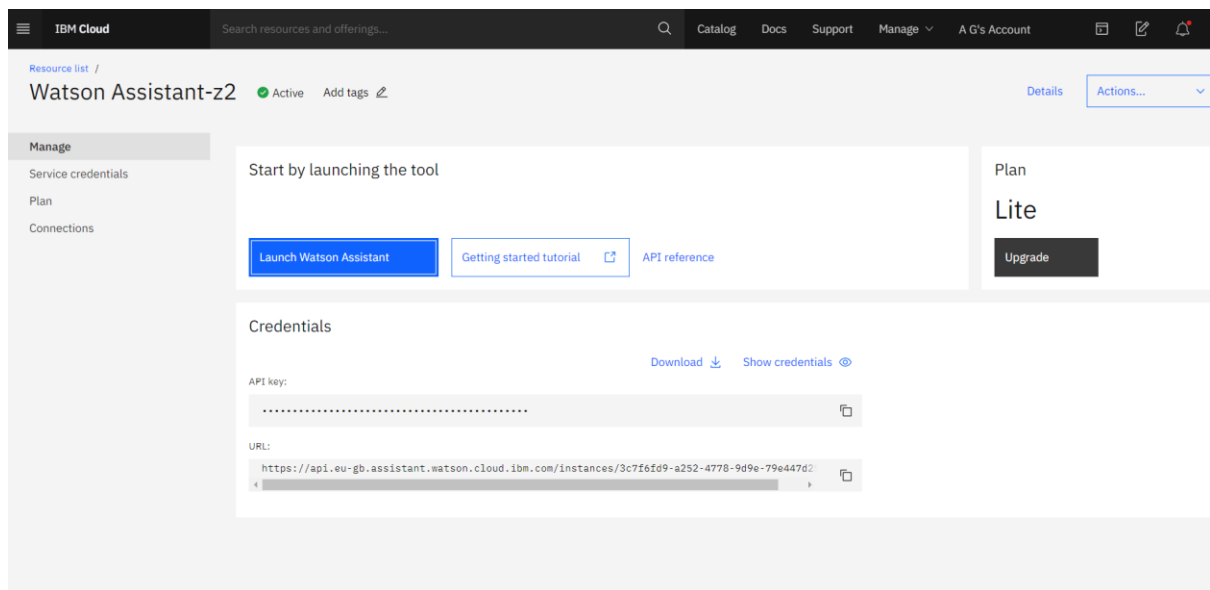
In this book we will use everything from a Lite account.

So select the Lite offering for Watson Assistant Service and you can scroll down to change the name of your service. For Lite account the resource group would be Default. Also at the top of the page you might have noticed *Select a Region* field.

Every Watson Service is taken from IBM Cloud, and these are few of the cloud regions. As of now it won't matter much but we will look at this later. Let us keep it as it is for now.

Click on Create button on the right bar at the bottom and your Watson Assistant resource will be created in about thirty seconds.

Once your Watson Assistant is created you will be taken to the Manage page. You will be able to see the API key and URL for web integration. We will come to this later. You will also see a Getting Started tutorial link which is a great documentation for beginners.

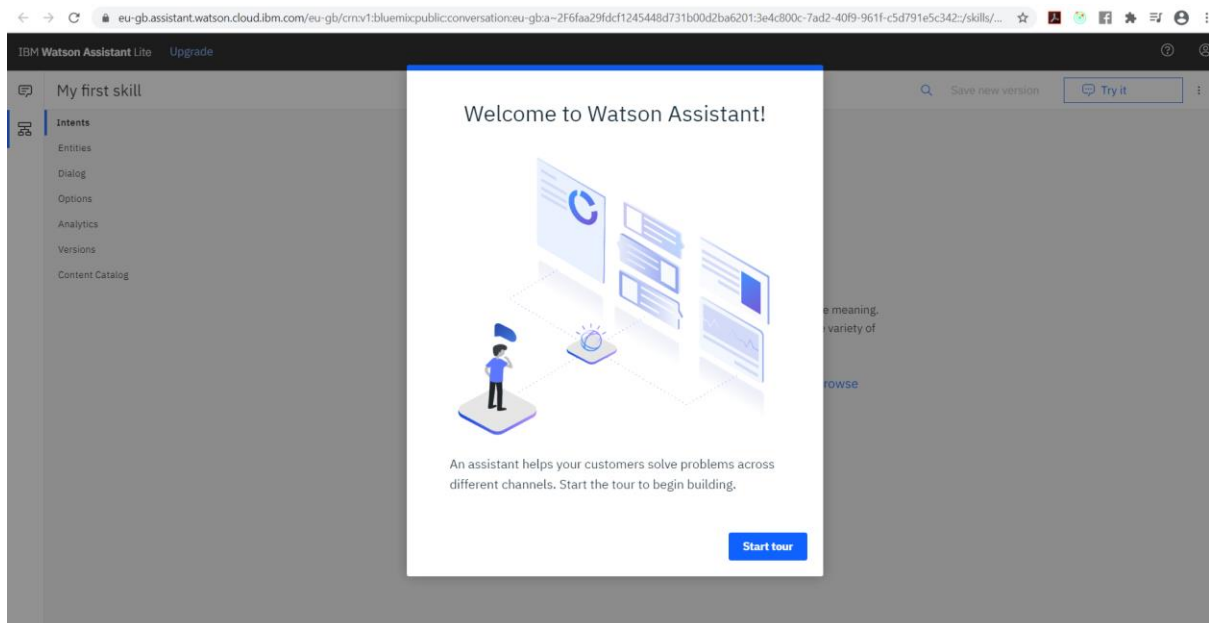


[Figure 2.8]

Click on the Launch Watson Assistant button and your first Watson Assistant will open in a new browser tab.

The Watson assistant page shows the start tour pop up where we can take a tour to understand the basics of Watson Assistant.

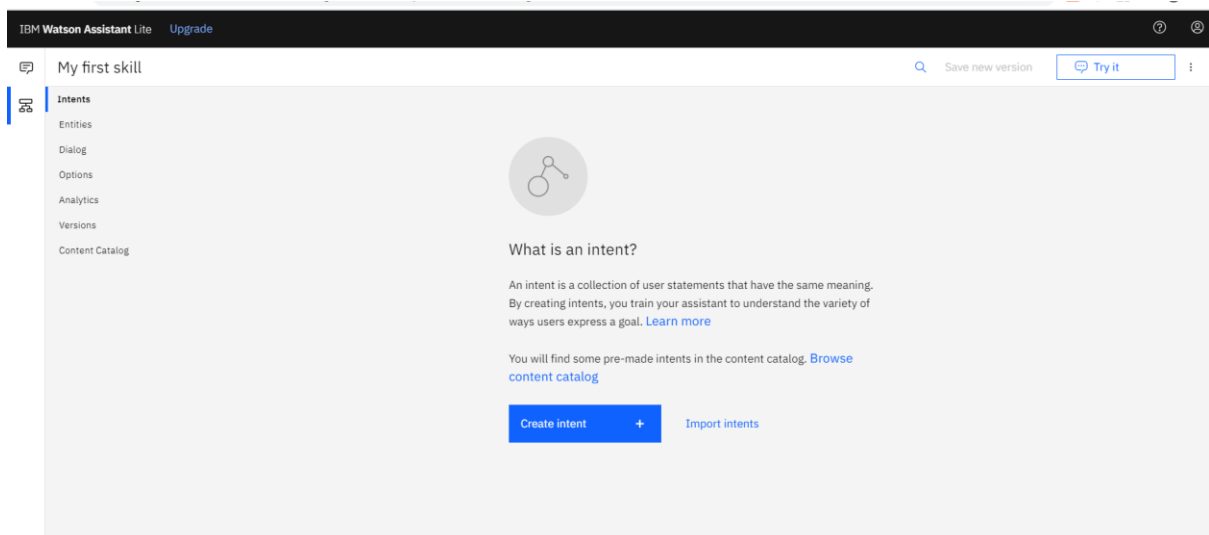




[Figure 2.9]

But we already know what are intents, entities, etc. So let us start and cancel this and skip to creating the assistant.

The Watson Assistant gives us a default skill – My first skill and tells us to create intent. But first let us look around a bit and get a feel of the environment.

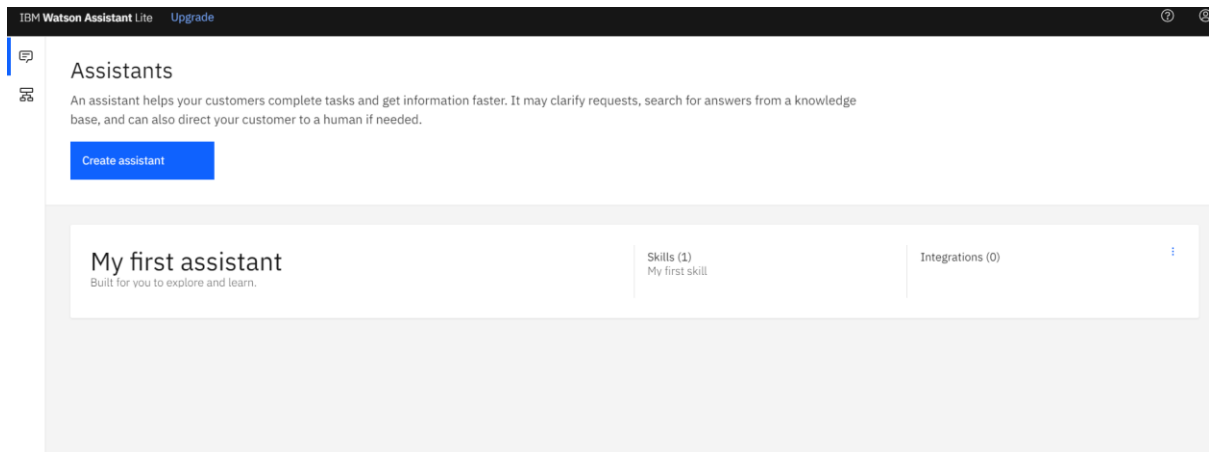


[Figure 2.10]

As you can see we have different tabs for creating and managing intents, entities and dialogs. If you hover on the leftmost bar you will see two tabs – Skills and Assistants and currently we are on the Skills page. An Assistant is the chat bot that we will deploy and each assistant can have a skill set. So we can prepare different skills like `Airport_Customer_care`, `Restaurant_Customer_care` and we can attach each of them

to an Assistant and deploy. For Lite version, we can only attach a single skill to an assistant.

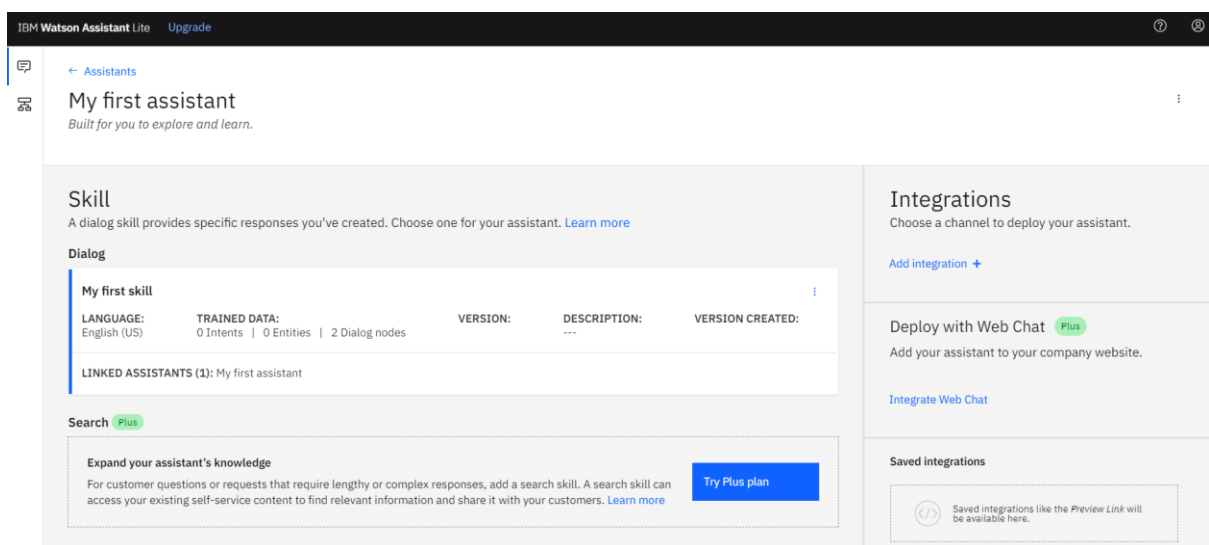
If you click on the Assistant tab you will see that it already has a default assistant for you – My first assistant. It also has attached the default skill, i.e. My first skill to this assistant and it does not have any integrations as of yet. Integrations are needed for deployment. We will come to this later.



[Figure 2.11]

On the right of this assistant you can click on the three dots to see the settings. The settings will show the name, API details and Inactivity timeout. We will need this when we move for deployment.

You can click on the assistant to see the assistant details. In the details page it shows the Skill details attached to it with the number of dialogs, Integrations and web chat deployment.



[Figure 2.12]

Now before proceeding let us frame a business problem and understand it.

## The Use Case

In the situation of a pandemic such as COVID-19 we are to stay indoors. Let us build a customer care for a restaurant which can take your order for a meal and your address and be able to deliver to your home. The food can be customized based on user taste and the chat bot has to understand all this without any human intervention.

For any software development problem, the first step is a proper plan. So before doing anything on the computer let us plan everything.

Let us consider our chat bot can accept food orders, specifically Burger and Chicken preparations. We will ask the user which of the two they want and then based on that we will ask more specific about the item. For this use case let us assume our kitchen can prepare the following burgers - Veg Burger, Chicken Burger, and the following chicken preparations – Chicken Wings, Chilli Chicken, Spicy Grilled.

For any Watson Assistant to come into being, we need to properly define the intents, entities and dialogs. So let us build our intents, entities and dialogs.

## Create Your Intents

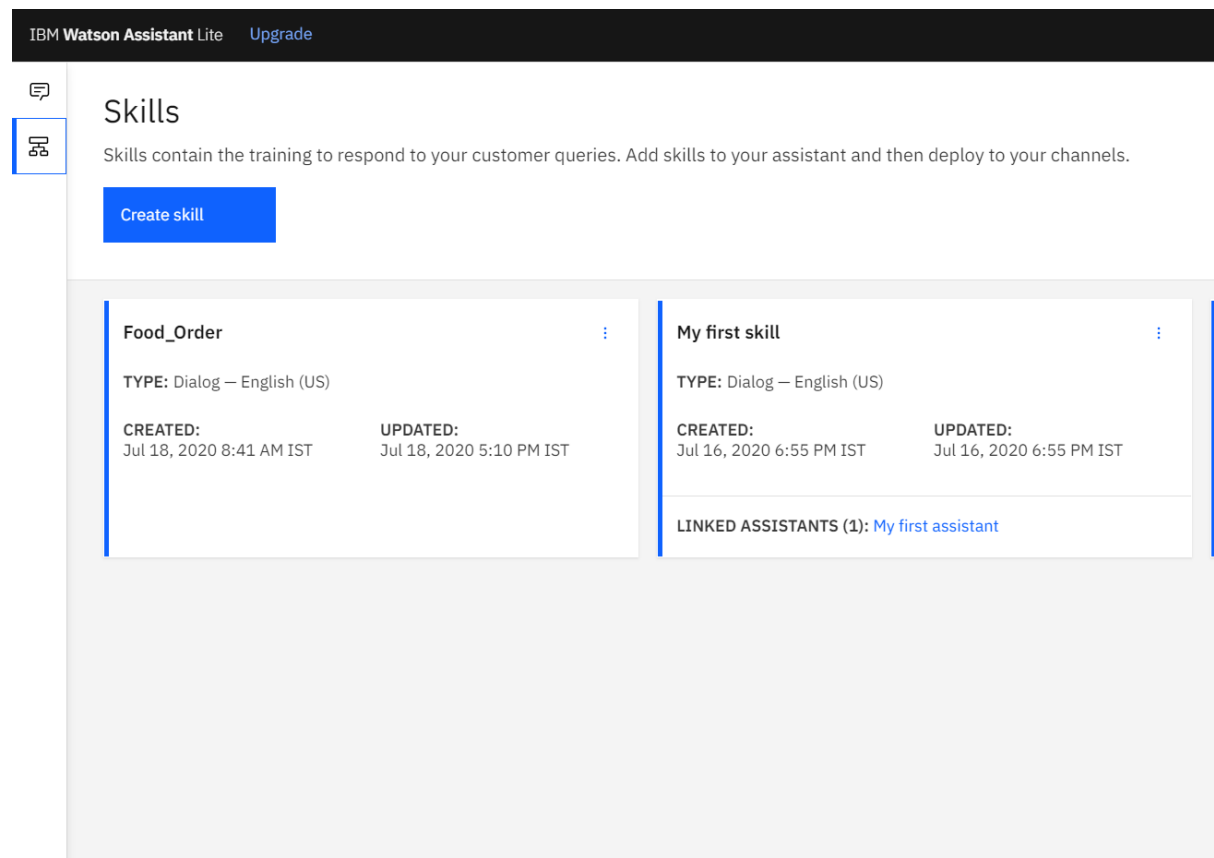
Remember that intents are the goals that the chat bot expects to address. When you are planning you can have as many intents you want and never end. But we must have some finite number of objectives for the chat bot and tell the user to contact a human for anything else. Also keep in mind that the user might think that it is a human. So you must also have some casual conversation objectives as well. All intent names start with #. Let us have the following intents:

- **#greet** : We expect our users to greet when they start their conversations. The IBM Watson provides a welcome dialog node when the chat begins by default. But we also need our assistant to understand when the user greets back. Or else the response to greet would fall into an unknown category and Watson also has a response node to any fallbacks or unknown categories.
- **#order** : This is the main intent of our assistant. We would definitely want our customer to order food. This intent would capture such user requests.
- **#exit** : Just like *#greet*, we also want our assistant to understand when the customer bids adieu.

Let us go ahead create the intents.

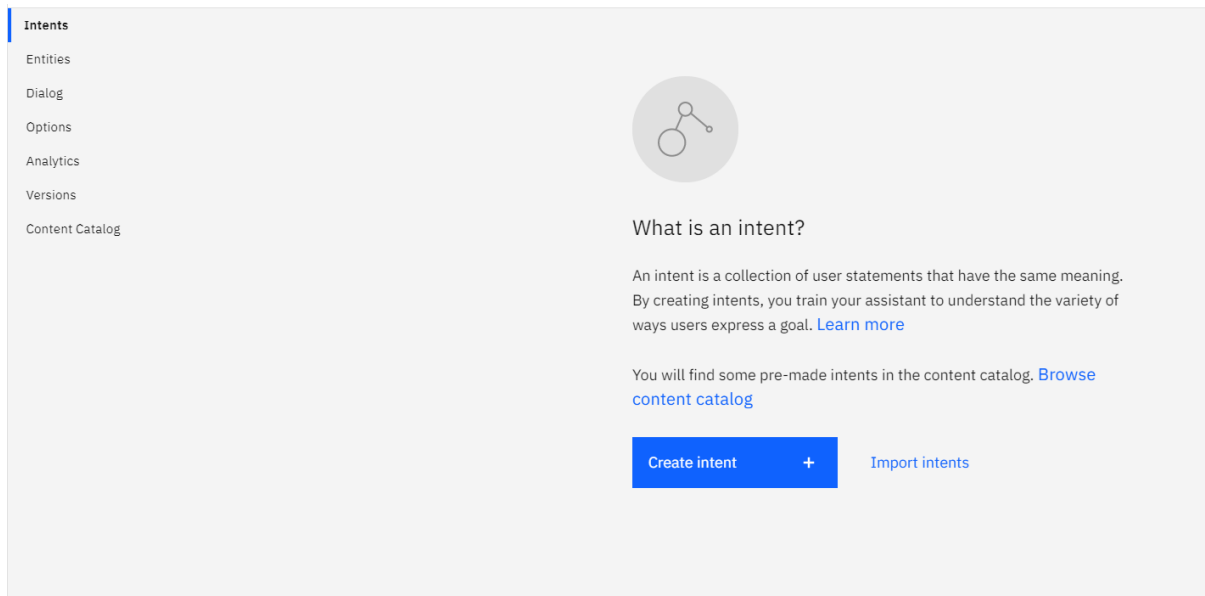
On the left bar, click on Skills and all your skills will be listed. As of now, you will only have the default skill – My first skill. You can go ahead and create your own

food ordering skill here as a new skill or edit the present skill. I have created a new skill by the name **Food\_Order**. You can create a skill by clicking on the *Create Skill* button.



[Figure 2.13]

Once this is done click on the skill to edit and you will be taken to the intents tab. Here you can create intent by clicking on the *Create intent* button. You can also create intents by importing a CSV list of intents using the *Import intents* button.



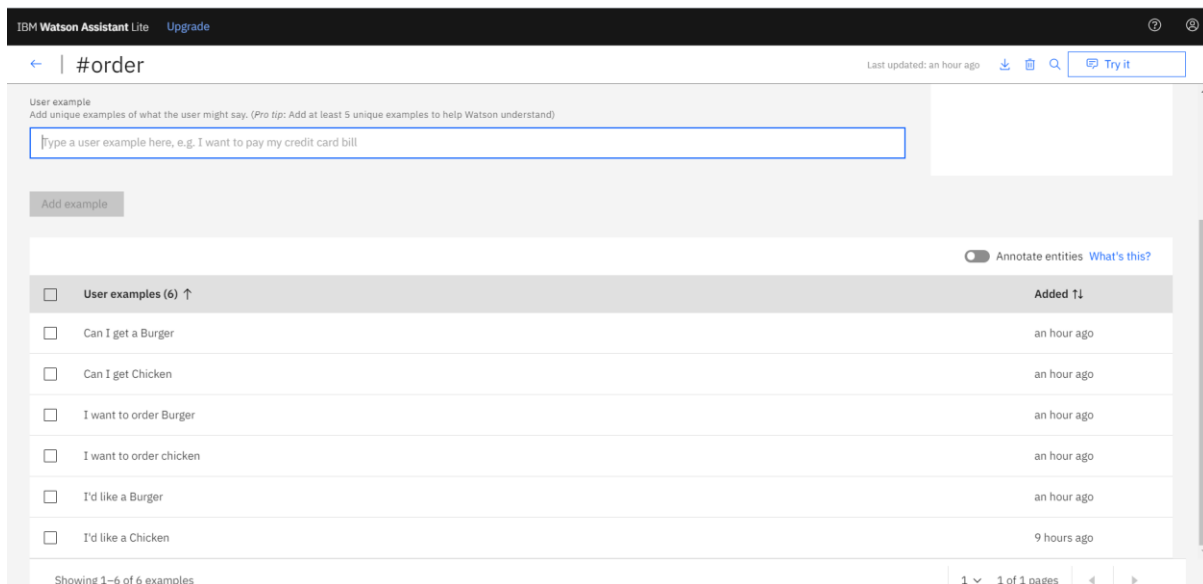
[Figure 2.14]

After clicking the *Create intent* button, you will be asked to enter a name for the intent and an optional description. So, let us create the most important intent which is pertains to our business use case, i.e. *#order*.

Enter the name as order (# is be prefixed already) and click on Create and you will be asked to enter some User examples. In order for the Watson Assistant to understand the intent, we must enter as many user examples as possible. These user examples are actually the training data for Watson to understand the intent.

We can think of several ways in which a customer can ask for an order. For example, the user may ask “Can I get a burger” or he can say “I’d like to order a burger”. As we are dealing with burgers and chicken, it would be better if we enter examples for each case having both burger and chicken, such as “Can I get a burger” and “Can I get Chicken”. You can go ahead and enter as many user examples as you want. The more the better.

I have compiled the following list of user examples which I feel will suffice for our assistant. If you are finding difficult to create your own, feel free to use the following from the Figure 2.15 below.



[Figure 2.15]

Once done, you can go back to the skill details page by clicking on the left arrow button on the top left. You will see your intent created. To create the next intent click on the *Create intent* + button on the top right corner.

Similarly, you can create the **#greet** intent now. This intent will understand the user greetings such as "Hello", "Good Morning", etc. You can enter as many greeting examples as possible. Following are the list of user examples for **#greet** that I had given for this use case:

- Good Afternoon
- Good Evening
- Good Morning
- Hello
- How are you

The last intent we are left with is the **#exit** intent. This is the intent we define for the Assistant to understand when the customer bids adieu. I had given the following list of user examples for this:

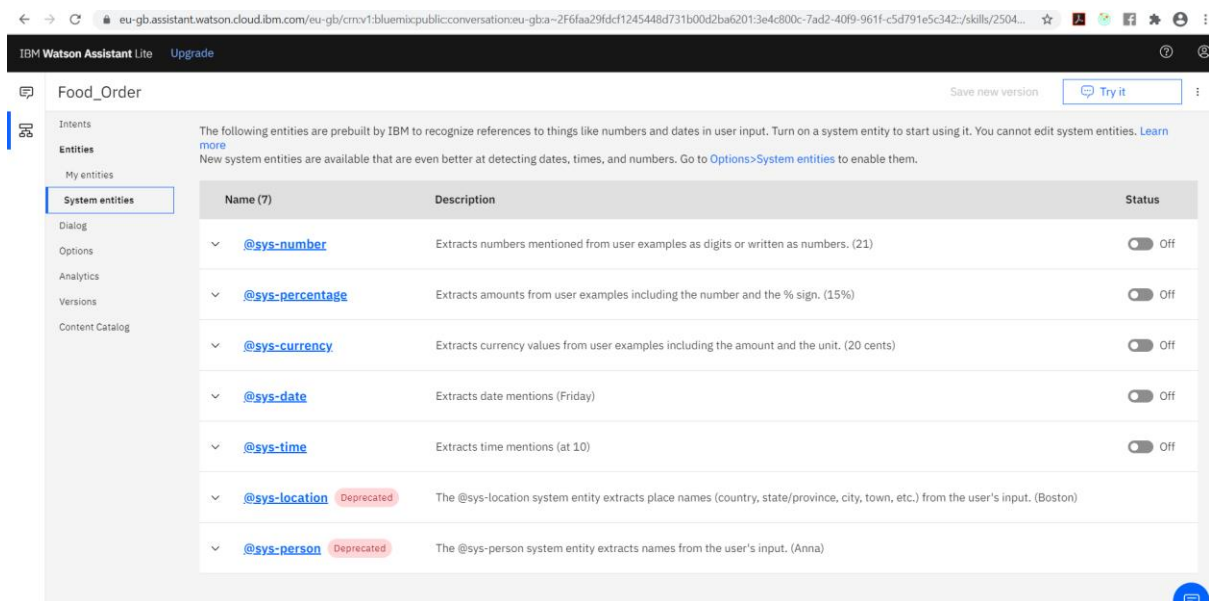
- Bye
- Great
- Thank you
- Thanks

Now we are done with Intents. Let us go ahead and define our entities.

## Create Your Entities

To move on to entities, click on the **Entities** link just below **Intents** on the skill page. You will be able to see that the link expands showing two sub links – **My entities**, **System entities**.

Watson already provides with a set of *System entities* for the assistant to understand, such as number, percentage, date, etc. and they are prefixed with “sys”, such as *sys\_date*. Click on *System entities* to view them (Figure 2.16).



[Figure 2.16]

If we want to use any of them, we have to set the status to on by toggling the *on* button on the right of any row beside the entities in the Status column.

In our use case we won't be needing any of them.

As entities are the specifics that the intent will act upon, we can have the following entities (entities start with @) :

- **@food\_type** : This entity will hold the food type, i.e. Burger or Chicken.
- **@burger\_type** : This entity will capture the type of burger. Let us assume, our kitchen can prepare Veg Burger and Mexican Burger.
- **@chicken\_type** : Similar to the one above, this will capture the chicken preparation. Let us assume, our kitchen can prepare Chilli Chicken, Chicken Wings and Spicy Grilled Chicken.

Now that we have decided our entities, let us create them.

Click on *My entities* sublink on the left to create your entities which will be needed for our use case. Similar to intents, there will be no user defined entities in the beginning and we can create by clicking on the **Create entity** button.

To create our first entity, enter the name *food\_type* (as @ is already prefixed) and you will be asked to enter values and their corresponding synonyms. Our *@food\_type* entity can understand the following values as we decided - *burger, chicken*.

Synonyms are added so as to make the assistant understand something similar can also be entered by the user. For this entity values we won't need to enter synonyms. So enter each of the value in the text field below *Value* one by one and click on *Add value* button. Also make sure *Fuzzy Matching* on the top right corner is set *on*. Fuzzy matching matches your values even if the user entry does not match exactly with the value but means the same.

The screenshot shows the IBM Watson Assistant Lite interface for configuring an entity named *@food\_type*. The entity name is entered in the top field. Below it, there are input fields for 'Value' and 'Synonyms'. The 'Value' field contains the text 'Type a value, e.g. Checking'. The 'Synonyms' field contains the text 'Type a synonym, e.g. Deposit'. There is a '+ ' button next to the synonyms field. Below these fields are two buttons: 'Add value' and 'Recommend synonyms'. On the right side, there is a 'Fuzzy matching' toggle switch which is turned 'On'. At the bottom, there is a table with the following data:

Values (2) ↑	Type
<input type="checkbox"/> burger	Synonyms
<input type="checkbox"/> chicken	Synonyms

At the bottom of the table, it says 'Showing 1-2 of 2 values'. On the right side of the bottom, there is a pagination control showing '1' and '1 of 1 pages'.

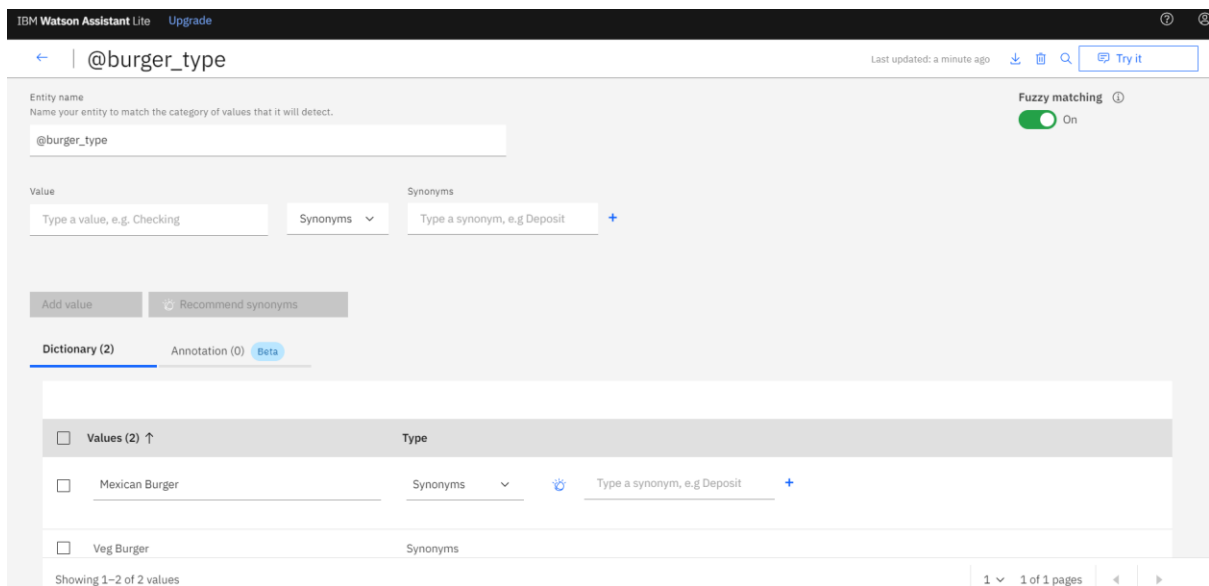
[Figure 2.17]

To move to the previous page click on the right arrow on the left top corner of the page.

Let us create the next entity - *@burger\_type*.

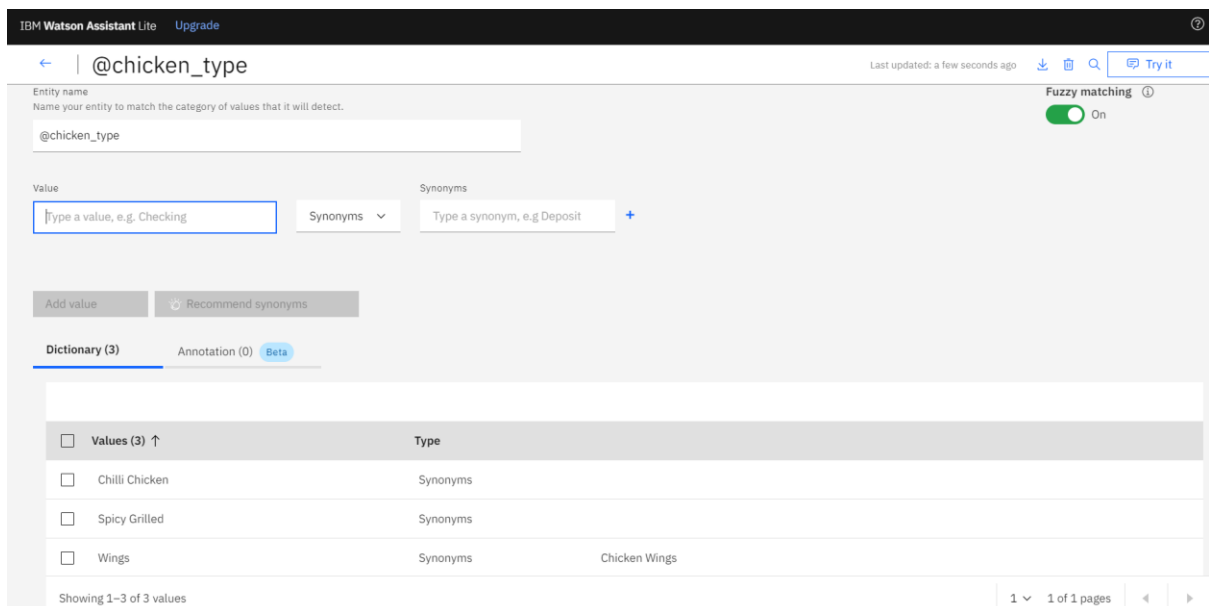
Click on the Create Entity button from the entity list page and type the name as *burger\_type*. We will have the values – *Mexican Burger, Veg Burger* for our use case.





[Figure 2.18]

Similarly we will create *@chicken\_type* entity to capture the chicken preparation. We will have the values - *Wings, Chilli Chicken, Spicy Grilled*. Here we can add a synonym to our *Wings* value – *Chicken Wings*. You can add synonyms for the other values as well.



[Figure 2.19]

Now that we are done with intents and entities, we are left with the most important building block of any assistant – Dialog.

Dialogs define the conversation flow in the chat bot. So before proceeding, we need to plan out the conversation flow.

Any conversation flow is like a set of conditional flow of conversations. For example, when we go out to buy an ice cream, the ice cream seller first asks us whether we need a cone or not, then the flavour, and then any other specifics and so on. We need to model our conversation flow in a similar manner. Also keep in mind that if we ask the ice cream seller to give us *one big cone ice cream of a mango flavour topped with a cherry*, the ice cream seller should not ask us anything else such as flavour, or cone, etc. as he already has all the information.

People with a background of programming have seen how a conditional flow work. It constitutes an if-else-if ladder. If you have no knowledge of conditional programing, imagine a set of conditions being asked one after the other, one being dependent on the previous condition.

Our conditional flow can be something like the following:

1. Greet the user.
2. Respond if the user wants to order in the next step.
3. If the user already asked for an order, then ask to choose between – Burger or Chicken.
4. If the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between – Mexican Burger or Veg Burger.
5. Similarly, if the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between - Wings, Chilli Chicken, Spicy Grilled.
6. Once the user has asked for order (step 2) and has mentioned whether chicken or burger (step 3) and has also mentioned what kind of preparation (step 4 or 5 based on choice of step 3) then go ahead and place the order by asking the address and respond with the time to reach.
7. Check if the user responds with a thanks or bids adieu, then respond accordingly with a warm welcome.
8. Keep a check mentioning that it didn't understand in case the user enters something out of the training data.

Let us model our conditions more formally in terms of intents and entities based on the above conditional flow:

1. Greet the user. - *#greet*
2. Respond if the user wants to order in the next step. - *#order*
3. If the user already asked for an order, then ask to choose between – Burger or Chicken. - *#order AND @food\_type*
4. If the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between – Mexican Burger or Veg Burger. - *@food\_type = burger*

5. Similarly, if the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between - Wings, Chilli Chicken, Spicy Grilled. - *@food\_type = chicken*
6. Once the user has asked for order (step 2) and has mentioned whether chicken or burger (step 3) and has also mentioned what kind of preparation (step 4 or 5 based on choice of step 3) then go ahead and place the order by asking the address and respond with the time to reach. - *@chicken\_type* OR *@burger\_type*
7. Check if the user responds with a thanks or bids adieu, then respond accordingly with a warm welcome. - *#exit*
8. Keep a check mentioning that it didn't understand in case the user enters something out of the training data.

One should clearly understand the difference between a *AND* and *OR*. *AND* requires both of the two conditions to be satisfied to fulfil the request and *OR* requires any of them to be satisfied to fulfil the request.

But there is a flaw in these stair of conditions. And that is because a computer processes conditions sequentially. The problem is with the sequencing of the steps. If the user asks for a order first, and then specifies whether it is burger and chicken and then which preparation of chicken, then this sequence would run fine. The assistant would run through the steps one by one and would respond based on the condition. But what if the user directly orders a Mexican Burger? The assistant would move to step 2 and find a match because his *#order* entity matches. Although the perfect match would be the one in step 6. Hence the assistant would respond asking which type of order would the user prefer – burger or chicken. But the user have already mentioned what kind of burger, and this could lead to poor user feedback as he would be faced with all the queries that he has already mentioned in his first request.

To avoid this from happening, we need to re-order the conversation flow. We need to put the most specific request at first and then the next less specific one, until we reach the dialog checking only *#order*.

1. Greet the user. - *#greet*
2. Once the user has asked for order (step 2) and has mentioned whether chicken or burger (step 3) and has also mentioned what kind of preparation (step 4 or 5 based on choice of step 3) then go ahead and place the order by asking the address and respond with the time to reach. - *@chicken\_type* OR *@burger\_type*

3. If the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between – Mexican Burger or Veg Burger. - *@food\_type = burger*
4. Similarly, if the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between - Wings, Chilli Chicken, Spicy Grilled. - *@food\_type = chicken*
5. If the user already asked for an order, then ask to choose between – Burger or Chicken. - *#order AND @food\_type*
6. Respond if the user wants to order in the next step. - *#order*
7. Check if the user responds with a thanks or bids adieu, then respond accordingly with a warm welcome. - *#exit*
8. Keep a check mentioning that it didn't understand in case the user enters something out of the training data.

But there is still a flaw. If the assistant works in sequence, won't step 2 lead to step 3. Hence after knowing Mexican Burger, it is going to ask which type of burger, and then either burger or chicken. To fix this, we will also need to add a jump statement, so that after step 2, it can directly move on to step 7. Similarly, after step 3, the assistant should jump to step 2 and then to step 7. Let us add these jump statements and reorder a bit.

1. Greet the user. - *#greet*
2. If the user has asked for order and has mentioned whether chicken or burger and has also mentioned what kind of preparation then go ahead and place the order by asking the address and respond with the time to reach. - *@chicken\_type OR @burger\_type*
3. If the user already asked for an order, then ask to choose between – Burger or Chicken. Jump to Step 5. - *#order AND @food\_type*
4. Respond if the user wants to order in the next step. Jump to step 3. - *#order*
5. If the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between – Mexican Burger or Veg Burger. Jump to step 2. - *@food\_type = burger*
6. Similarly, if the user has asked for order and has mentioned Burger either in step 2 or 3 then ask to choose between - Wings, Chilli Chicken, Spicy Grilled. Jump to step 2. - *@food\_type = chicken*
7. Check if the user responds with a thanks or bids adieu, then respond accordingly with a warm welcome. - *#exit*
8. Keep a check mentioning that it didn't understand in case the user enters something out of the training data.

Please note that after step 3, we are moving to step 5 even if we don't know whether the user entered Burger or Chicken because even if the user chose Chicken but jumping to burger would now move to the next condition sequentially and would land to step 6.

Now we are in order. Try to run this in your mind and you will see that this will preserve the natural sequence of ordering food. We will refer these steps as our conversation model.

Let us start building this sequence of dialogs in our Watson Assistant.

## Create the Dialogs

Similar to Intents and Entities, click on the Dialog link on the left and you will be able to see that Watson has already built a very basic conversation model. Each of the boxes are called **dialog nodes** and they are linked with lines. These dialog nodes are each of the conversation steps that we chalked out earlier. Now, one of the nodes is the *Welcome* node, which greets the user with "Hello. How can I help you?" and the other one is the *Anything else* node which responds when the assistant can't understand the user words (step 8).

Now we have already defined a `#greet` intent. So we will edit the welcome node and add our greeting message as well.

Now to edit the node you have to click on it and on clicking it you will be able to see the following appear on the right:

The screenshot displays the IBM Watson Assistant configuration interface. At the top, there is a dark header bar with a question mark icon and a user profile icon. Below this, a navigation bar contains a search icon, the text "Save new version", a "Try it" button with a speech bubble icon, and a vertical ellipsis menu icon. The main configuration area is divided into several sections:

- Node Name:** A text field containing "Welcome". Below it, a note states: "Node name will be shown to customers for disambiguation so use something descriptive." To the right of this note is a "Settings" link.
- If assistant recognizes:** A section with a list of recognized intents/entities. Currently, it contains "welcome" with a trash icon and a plus sign to add more.
- Assistant responds:** A section for defining the response.
  - A dropdown menu is set to "Text". To its right are up, down, trash, and expand icons.
  - A text field contains the response: "Hello. How can I help you?". To its right is a trash icon.
  - Below the text field is a placeholder: "Enter response variation".
  - A note states: "Response variations are set to **sequential**. Set to [random](#) | [multiline](#)".
  - A link "Learn more" is provided.
- Add response type:** A blue link with a plus sign.
- Then assistant should:** A section for setting the next action.
  - A dropdown menu is set to "Wait for reply".
  - Below it, a note states: "Choose whether you want your Assistant to continue, or wait for the customer to respond."

A blue circular chat icon is located in the bottom right corner of the interface.

[Figure 2.20]

Now let's look at each of the components of a dialog.

The first text field at the top specifies the name of the node. This is optional.

The text field below **If assistant recognizes** is the field where we need to mention the conditions as intents and entities. In this case since `welcome` is neither an intent or an entity, it is a literal.

The text field below **Assistant responds** is the response type and value. If you click on the dropdown mentioned *Text*, you will see that Watson offers a range of ways to

respond other than simple text such as Option (if we need to provide options to users, mainly used in case of lists), Pause (in case you want the assistant to wait for a certain amount of time), or an Image. We will in general use only Text type.

We can also add multiple response and display them sequentially or randomly by clicking on *Add response type* + link. We will see this in action for the next node.

Lastly the text field below **Then assistant should** is how the conversation should move to the next node. There are two options for this – *Wait for reply* and *Jump to*. We can either wait for the user to respond and then move to the next sequential node, or we can jump to a specific node based on a condition (which we needed to do in our conversation flow).

Now we already have a greeting intent in our assistant - *#greet*.

In the condition field we will add this condition as well so the assistant also recognizes when the user greets and responds on this node.

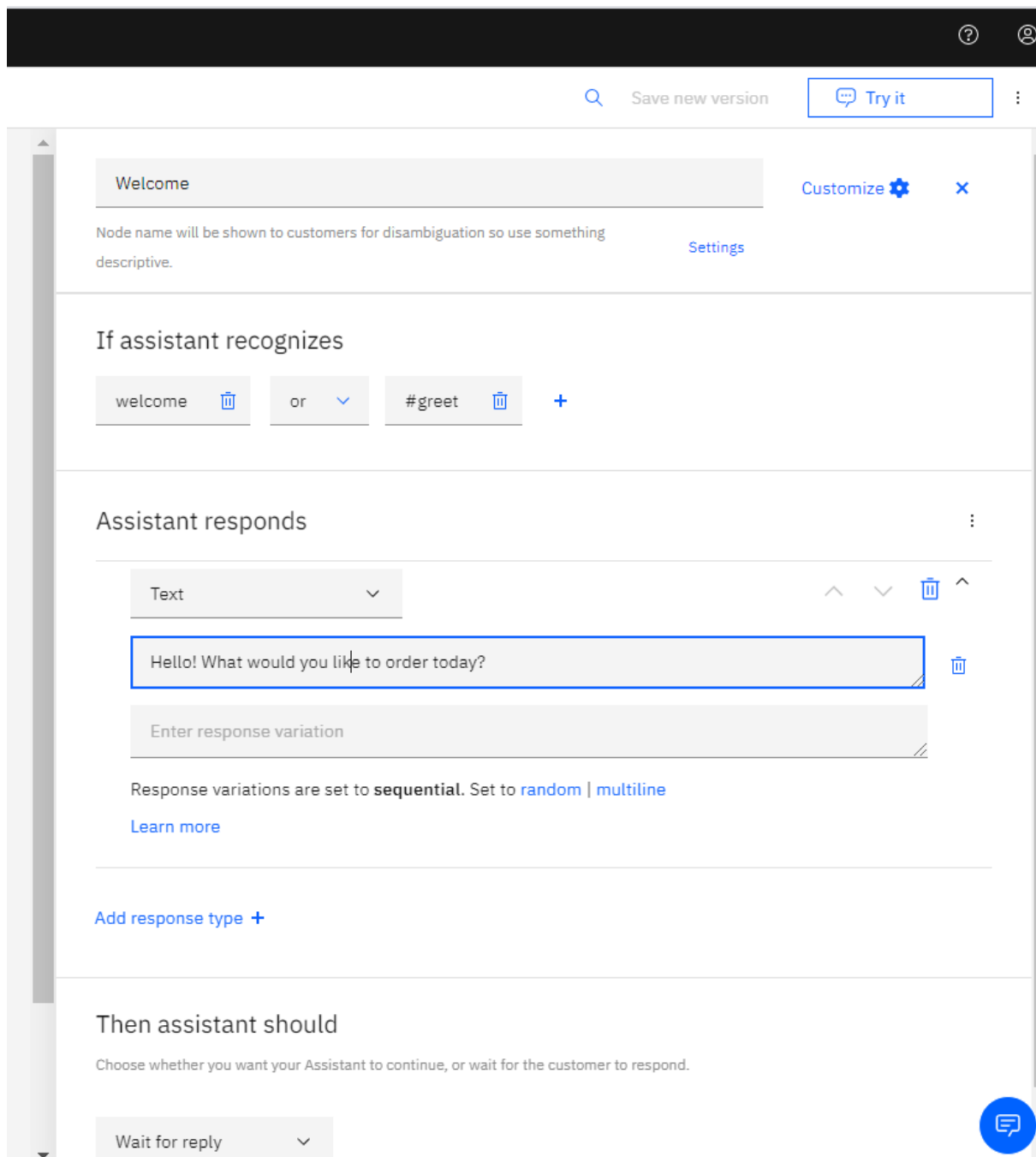
Click on the + sign beside *welcome* and Watson creates another condition field and a *and* dropdown. If you click on the dropdown, you will see it also gives an option to conjugate the conditions using *or*. This is the Operator field, which is used to conjugate multiple conditions.

We will use the *or* operator to conjugate *welcome* with our own intent *#greet*. So, go ahead and select *or* from the operator field and type *#greet* in the next condition field. You will even see an auto complete when you type # in the condition field, which is really helpful.

Now let us change the response text as well by asking the user to order. So, enter the following in the Response Text field:

Hello! What would you like to order today?

When you are done, your node should look like this:



[Figure 2.21]

Our first dialog node is ready.

Now we already have a *Anything else* dialog node by default. Lets keep it as is and click on it to see what it contains.



?

Save new version

Try it

Anything else

Customize

Node name will not be shown to customers for disambiguation.

Settings

If assistant recognizes

anything\_else

+

Assistant responds

Text

I didn't understand. You can try rephrasing.

Can you reword your statement? I'm not understanding.

I didn't get your meaning.

Enter response variation

Response variations are set to **sequential**. Set to [random](#) | [multiline](#)

[Learn more](#)

Add response type

+

Then assistant should

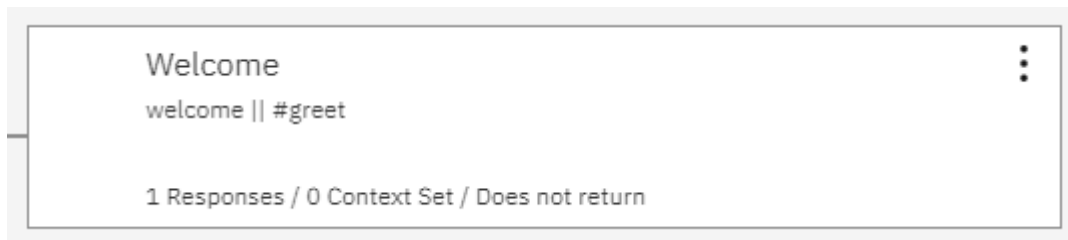
Choose whether you want your Assistant to continue, or wait for the customer to respond.

Wait for reply

[Figure 2.22]

This dialog node is almost the same as the default welcome node except that in the response field, it has multiple text responses with responses set to **sequential** as mentioned below. So the first time the dialog node is reached, it will respond with the response on the first line. The next time, with the one on the second line. Then the next time the one on the third line. And again if the node is reached the fourth time, it will respond with the response on the one on the first line (rolls back up) and it continues. We can also set it to **multiline**, in which case the response will show all the lines every time. If the response is set to **random**, then it will randomly pick up one line from those multiple lines each time.

Also if you notice the dialog flow diagram, you will see that each dialog node box shows the summary. For example the welcome dialog node box shows the following:



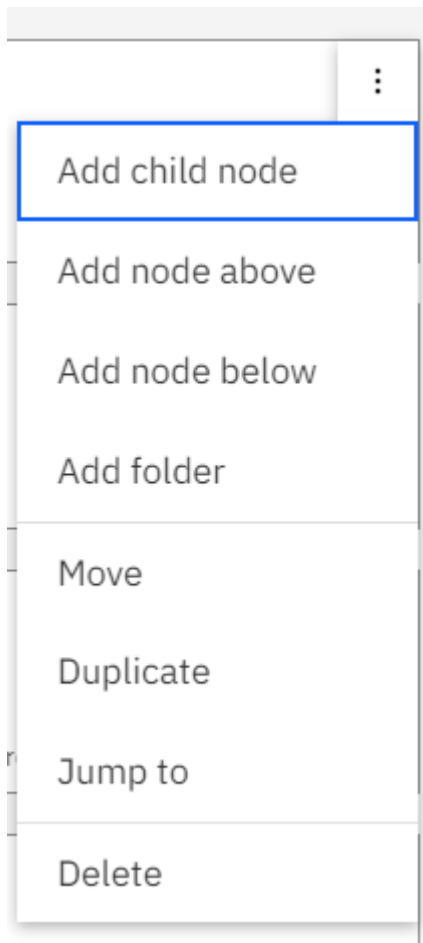
[Figure 2.23]

It shows the name of the node *Welcome*, the condition below *welcome || #greet* (in programming terms `||` symbolizes *OR* and `&&` symbolizes *AND*), and a small summary showing the number of responses, context variables set (we will see this soon) and whether it waits for user (*Does not return*) or jumps to another node.

Let us start adding our won dialog nodes based on the conversation flow we decided.

Lets add step 4 first, i.e. *#order*.

Click on the three vertical dots at the end of the Welcome dialog node and you will see the following dropdown in Figure 2.24.



[Figure 2.24]

Select *Add node below* from the options in the dropdown and a new node will be added in sequence just beneath it. The other possible options to add a node as seen from the dropdown are –

- *Add node above* - to add a node as a previous step.
- *Add child node* - this allows us to add a node as a child node in sequence which is generally needed if we want a sequence condition check to occur if and after the condition in the parent node is satisfied.
- *Add folder* – this is needed when we want to group together a set of nodes. This is needed when there are a large number of nodes. The sequence or activity is not affected by this.
- *Move*- in case we want to re-order the sequence of a node.
- *Duplicate* - in case we want to make a copy of the node.
- *Jump to* – if we want to set jump to a specific node after completion of this node activity (like we added jump statements in our conditional steps).
- *Delete* – if we want to remove the node.

In this node we are going to ask for an order and let the user choose between Burger and Chicken.

In the condition field (the field below the label *If assistant recognizes*), add our order intent. Type # and you will be able to select *#order* from the autocomplete options.

Now we will not add a Text response, nor will we add an option response because adding an option response lets user choose but does not store the value chosen. To store the value chosen we need to know two new concepts – **slots** and **context variables**.

**Slots** allow you to prompt user for a certain entity until the user enters a value for it and stores the value in a **context variable**.

**Context Variables** can store a value for the whole life cycle of the assistant until the user quits the Watson assistant so that we can use the value later (in case we want to display).

Now to enable slots click on the *Customize* link on the top left of the dialog details pane just beside the dialog node name. Under Customize Node tab, turn on the toggle button at the right corner and make sure to check the box – *Prompt for everything* and click on *Apply*.

Customize "#order" ×

Customize node

Digressions

Slots ⓘ

On

Enable this to gather the information your bot needs to respond to a user within a single node.

☒

 Prompt for everything

Enable this to ask for multiple pieces of information in a single prompt, so your user can provide them all at once and not be prompted for them one at a time.

Webhooks

Off

Enable this setting to send a POST request from this dialog node to the webhook URL. The URL and headers are defined in the Webhooks settings of the Options tab. After you enable this setting, the Multiple conditional responses setting is enabled automatically to support adding a response to show when the request is successful and another response to show if

Cancel


Apply

[Figure 2.25]

Once you switch on the Slots, you will get a slot field set after condition field – *Then check for*. We need to prompt user to enter food type (i.e. burger or chicken) here. So enter our entity *@food\_type* beneath *Check for*. Beneath the label *Save it as*, we need to enter a **context variable**. A context variable starts with \$. Let's name the context variable same as our entity (to avoid confusion) - *\$food\_type*, as this context variable will store the value of the entity *@food\_type*. Let *Type* be Required.



descriptive.

If assistant recognizes

#order  +

Then check for

0 [Manage handlers](#)

	Check for	Save it as	If not present, ask	Type		
1	@food_type	\$food_type	Option	Required		

[Add slot +](#)

If no slots are pre-filled, ask this first:

[Figure 2.26]

Click on the button just beside *Required* to configure the slot.

In the Configure Slot pop up, select *Option* from the dropdown below *If slot context variable is not present ask:*. Let's leave Title and Description blank as it is optional and click on *Add option* to have two options. Fill the list label and value as Burger and Chicken as shown in Figure 2.27.

Configure slot 1

Check for: @food\_type

Save it as: \$food\_type

If slot context variable is not present ask: Slot is required

	Option	Title	Description (optional)
1	Burger	Add title text	Add description
2	Chicken		

Add option +

Cancel Save

[Figure 2.27]

Click on Save and you have your slots ready.

You can optionally add a response type after *If no slots are pre-filled, ask this first:*, if you want to ask before you prompt.

Leave everything else as it is. Watson automatically saves whatever changes you make to a dialog on runtime dynamically.

Later we will add a jump statement to jump to step 3 to ask to choose which preparation of Chicken or Burger based on the choice the user made on the **Slot** here (as decided in our conversation model). But for that first we need to create the dialog node corresponding to step 3.

So, either add a node above *#order* node or add a node below *Welcome* node by clicking on the three dots of *#order* node or *Welcome* node.

We are not going to add any slot here, as the user already chose between Chicken and Burger in the *#order* node.

Add the condition in the condition field as decided `#order` and `@food_type`. Here `and` is the operator. But remember that we have stored `@food_type` value in context variable `$food_type`. So add another `or` operator and another condition `$food_type`. But then why do we even need `@food_type`. It is because, if the user directly asks for an order along with the `@food_type` we will need to capture that here as well. Finally add a Text response asking user to choose the particular preparation as shown in Figure 2.28 below.

Enter node name (optional) Customize ⚙️ ✕

Node name will be shown to customers for disambiguation so use something descriptive. [Settings](#)

---

If assistant recognizes

#order 🗑️ and ▼ @food\_type 🗑️ or ▼ \$food\_type 🗑️ +

---

Assistant responds ⋮

Text ▼ ⬆️ ⬇️ 🗑️ ⬆️

Great please choose among the following: 🗑️

Enter response variation

Response variations are set to **sequential**. Set to [random](#) | [multiline](#)

[Learn more](#)

[Add response type +](#)

---

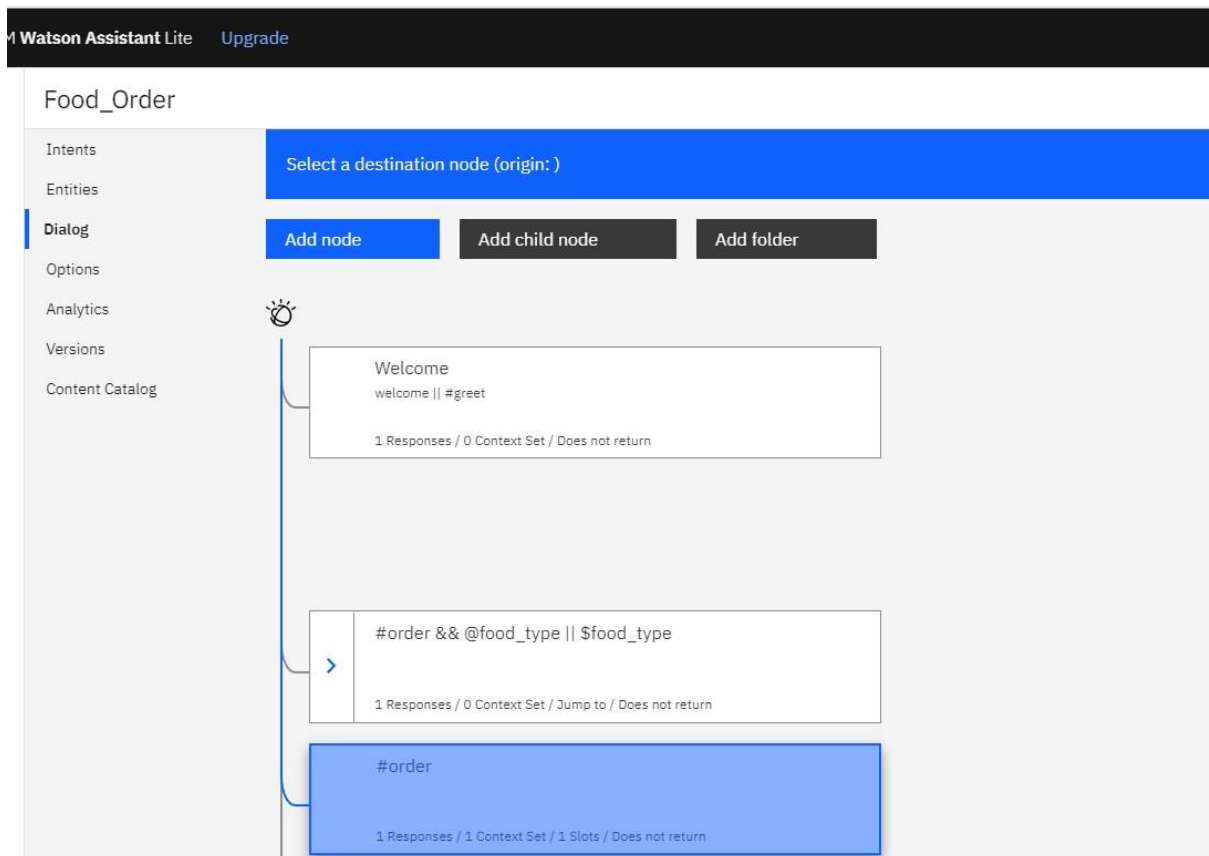
Then assistant should

[Figure 2.28]

Now that we have our step 3 ready, let us add a Jump from the `#order` node (step 4) as discussed.

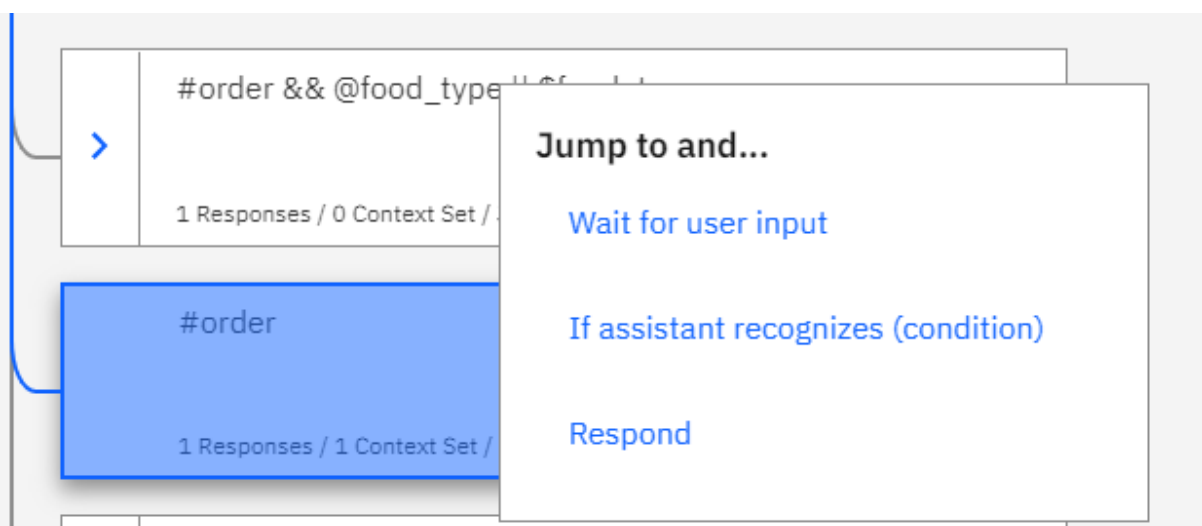
So click on `#order` dialog node and on the details pane in the left scroll down to *Then assistant should* dropdown and select *Jump to*. As soon as you click on this, it will ask you to select a Destination node as shown in figure 2.29 below.





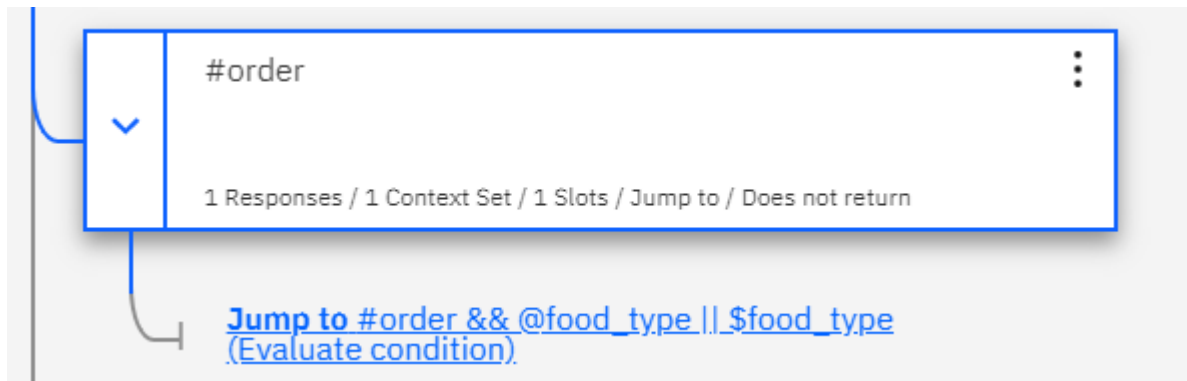
[Figure 2.29]

Click on the `#order && @food_type || $food_type` node and it will prompt with a dropdown to select from – *Wait for user input*, *If assistant recognizes (condition)* and *Respond*.



[Figure 2.30]

Select the option *If assistant recognizes (condition)* as we want our jump to take place once the condition for this node is satisfied. As soon as you select this a jump statement will be added to `#order` node as shown in figure Figure 2.31.



[Figure 2.31]

Let us now create the nodes for steps 5 and 6 to ask user to enter the specific preparation for Burger or Chicken based on user input.

Let us first create Step 5 for choosing burger preparation.

Create a node just below *#order* node. Here we are going to give the condition *@food\_type:burger* so that the entity *@food\_type* is burger. So any mention of this entity would lead to this node. Since we are going to ask for choice and for convenience store it in context variable to be used later, we will add slots like we did previously.

So enable slot by clicking on customize and enable Slot with *Prompt every time* option checked as shown in Figure 2.25. Next we will add *Then check for* values –

- *Check for* - *@burger\_type*
- *Save it as* - *\$burger\_type*

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

Settings

@food\_type:burger

+

Then check for

0 Manage handlers

	Check for	Save it as	If not present, ask	Type		
1	@burger_type	\$burger_type	Option	Required		

Add slot +

Assistant responds

Text

Enter response text

[Figure 2.32]

Click on the button beside *Required* to add the slot options as shown in figure Figure 2.33.

Configure slot 1

×

Check for

@burger\_type

Save it as

\$burger\_type

---

If slot context variable is not present ask:

Slot is required ⓘ

⋮

Option

⌵

⌴ ⌵ 🗑 ⌴

Title

Please Select Type of Burger

Description (optional)

Add description

	List label	Value	
1	Veg Burger	Veg Burger	🗑
2	Chicken Burger	Chicken Burger	🗑

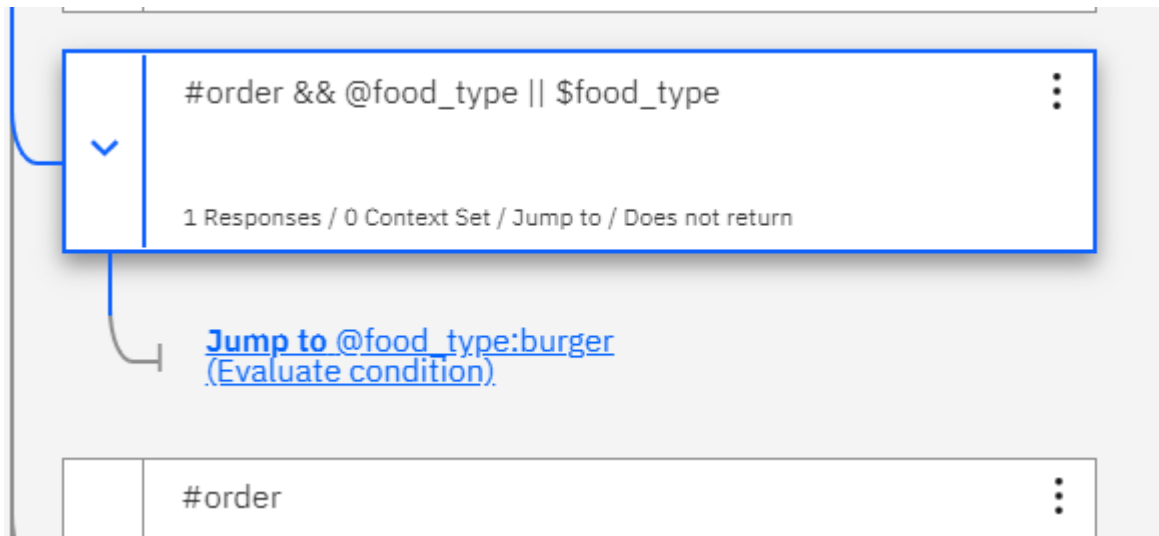
Cancel

Save

[Figure 2.33]

Click Save and you node is ready.

Now that we have our dialog node corresponding to Step 5 is ready, let us add the jump statement to node `#order && @food_type || $food_type` which corresponds to step 3 in our conversation model. So click on `#order && @food_type || $food_type` node and scroll down to *Then assistant should* at the very bottom. Select *Jump to* and select the node `@food_type:burger` as destination (keeping *If assistant recognizes condition* as shown in Figure 2.30) by clicking on it. Finally the node should be displayed like the Figure 2.34 in the flow diagram.



[Figure 2.34]

Similarly let us create a similar node for Chicken.

Create a node just below *@food\_type:burger*. Here we are going to give the condition *@food\_type:chicken* so that the entity *@food\_type* is chicken. Enable slot by clicking *Customize* and toggling *Enable Slot* button and keeping *Prompt* for everything as shown in Figure 2.25.

Add the condition for slots in *Then check for* as follows ( as shown in Figure 2.35) :

- *Check for* - *@chicken\_type*
- *Save it as* - *\$chicken\_type*

Save new version
Try it

Customize

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

Settings

### If assistant recognizes

@food\_type:chicken
+

### Then check for

0 Manage handlers

	Check for	Save it as	If not present, ask	Type	
1	@chicken_ty	\$chicken_typ	Option	Required	Settings Delete

Add slot +

### Assistant responds

Text
^
v
Delete
^

[Figure 2.35]

Now click on the settings button beside *Required* and add the slots as shown in the figure below (Figure 2.36):

Configure slot 1
⋮
×

Check for

@chicken\_type

Save it as

\$chicken\_type

---

If slot context variable is not present ask:

Slot is required ⓘ

⋮

---

Option
▼
^
▼
🗑️
^

Title

Add title text

Description (optional)

Add description

---

	List label	Value	
1	chilli chicken	chilli chicken	🗑️
2	wings	wings	🗑️
3	spicy grilled	spicy grilled	🗑️



Cancel

Save

[Figure 2.36]

Now we need to create Step 2 of our conversation model to add a jump statement to both the nodes *@food\_type:burger* and *@food\_type:chicken*.


Create a new node below *Welcome* node. Here we need to check whether the user has already provided any of the two entities - *@burger\_type* or *@chicken\_type* to specify which preparation of burger or chicken does the user desires to order. So in the condition add *@burger\_type* and *@chicken\_type* conjugating using *or* operator in between as shown in Figure 2.37.


Customize  


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

Settings

### If assistant recognizes

@burger\_type 

or 

@chicken\_type 


+


[Figure 2.37]

Now here we do not need any slots but we need to set a context variable to let the assistant know that the user has already provided the specific preparation of the food item and so that we can go ahead and ask for the address. Let this context variable be *address* and we will set it to true if this dialog node is reached.

To set a new context variable, we need to open the context editor. To open the context editor, click on the three vertical dots beside *Assistant responds* label as shown in the Figure 2.38 below and click on *Open context editor*.

Assistant responds

Text 



Open JSON editor  
Open context editor

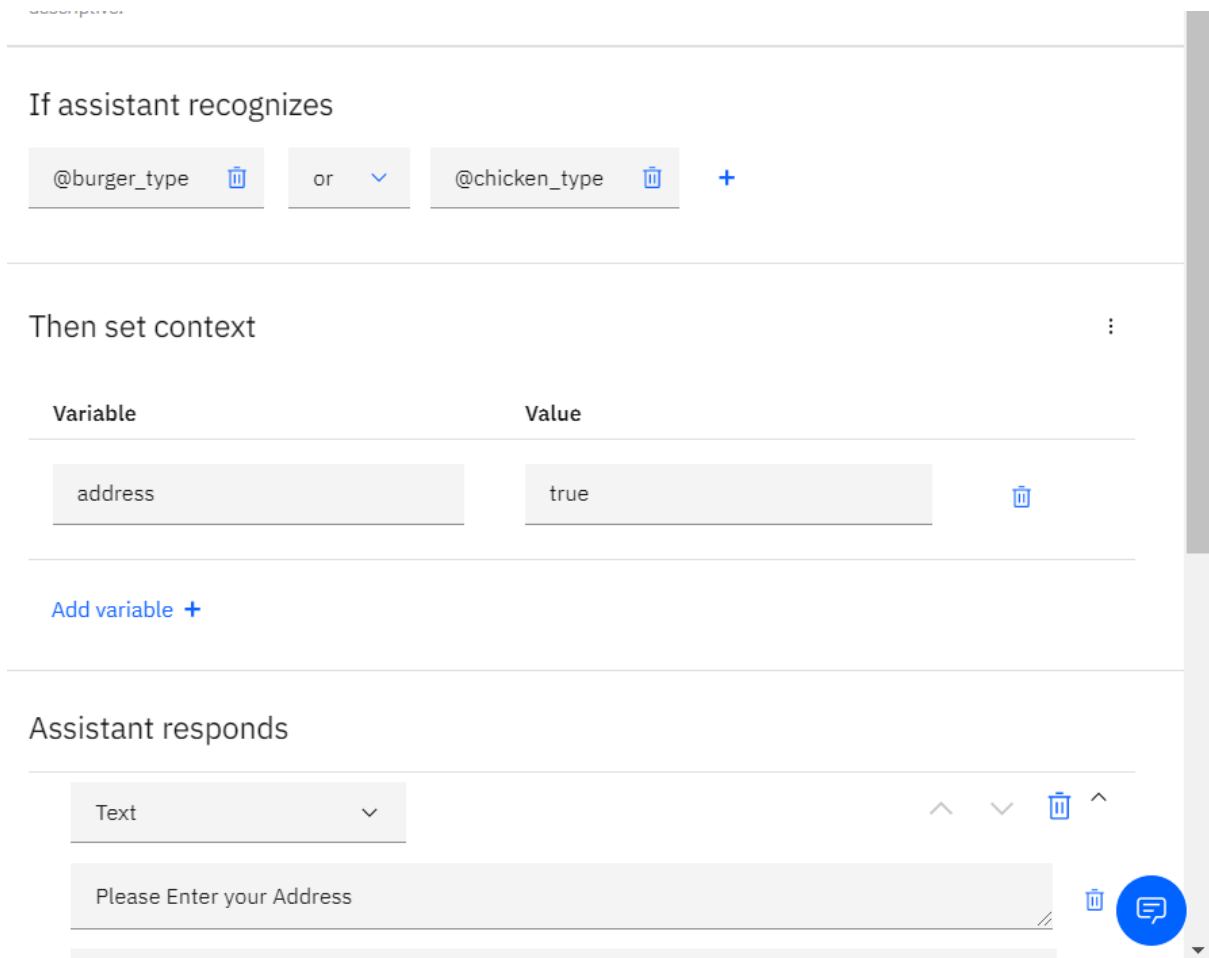
[Figure 2.38]

For programmer who feel more comfortable in JSON can also edit or create context editors in JSON by clicking *Open JSON editor*.

Once you click on *Open context editor*, Then *set context* section will open, where you can set a new context variable or add an existing variable and value. Add the *variable* as *address* and *value* to *true*(Figure 2.39).

Since we need to ask for address, let us add a text response as shown in Figure 2.39.



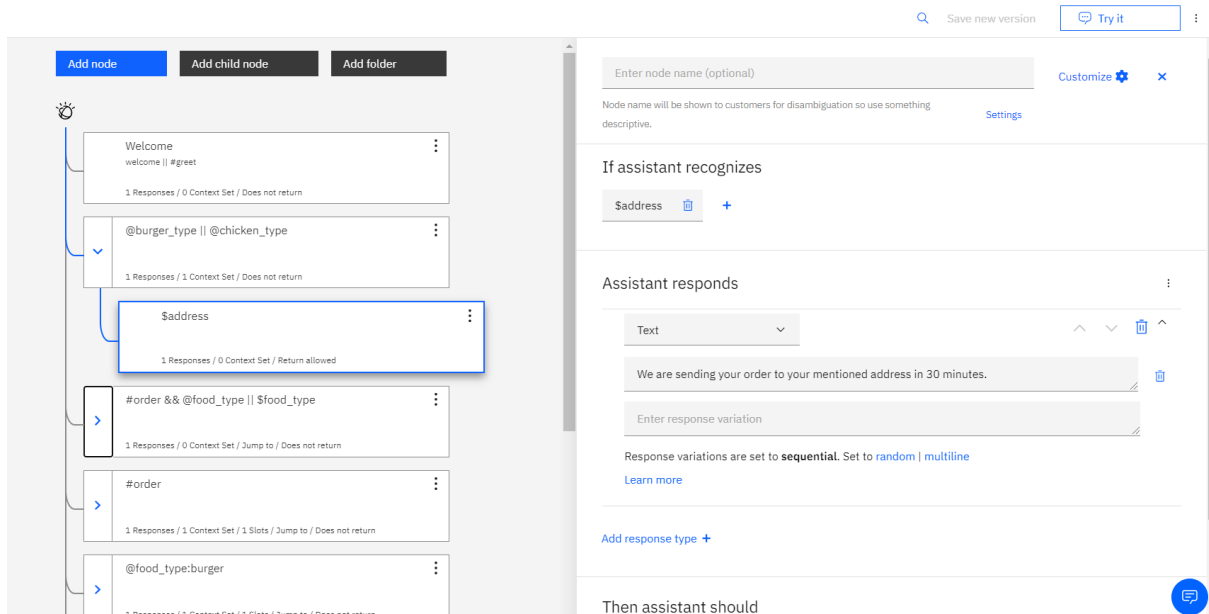


[Figure 2.39]

Next, let us accept the order after getting the address and acknowledge the user. We will create a new dialog node for this.

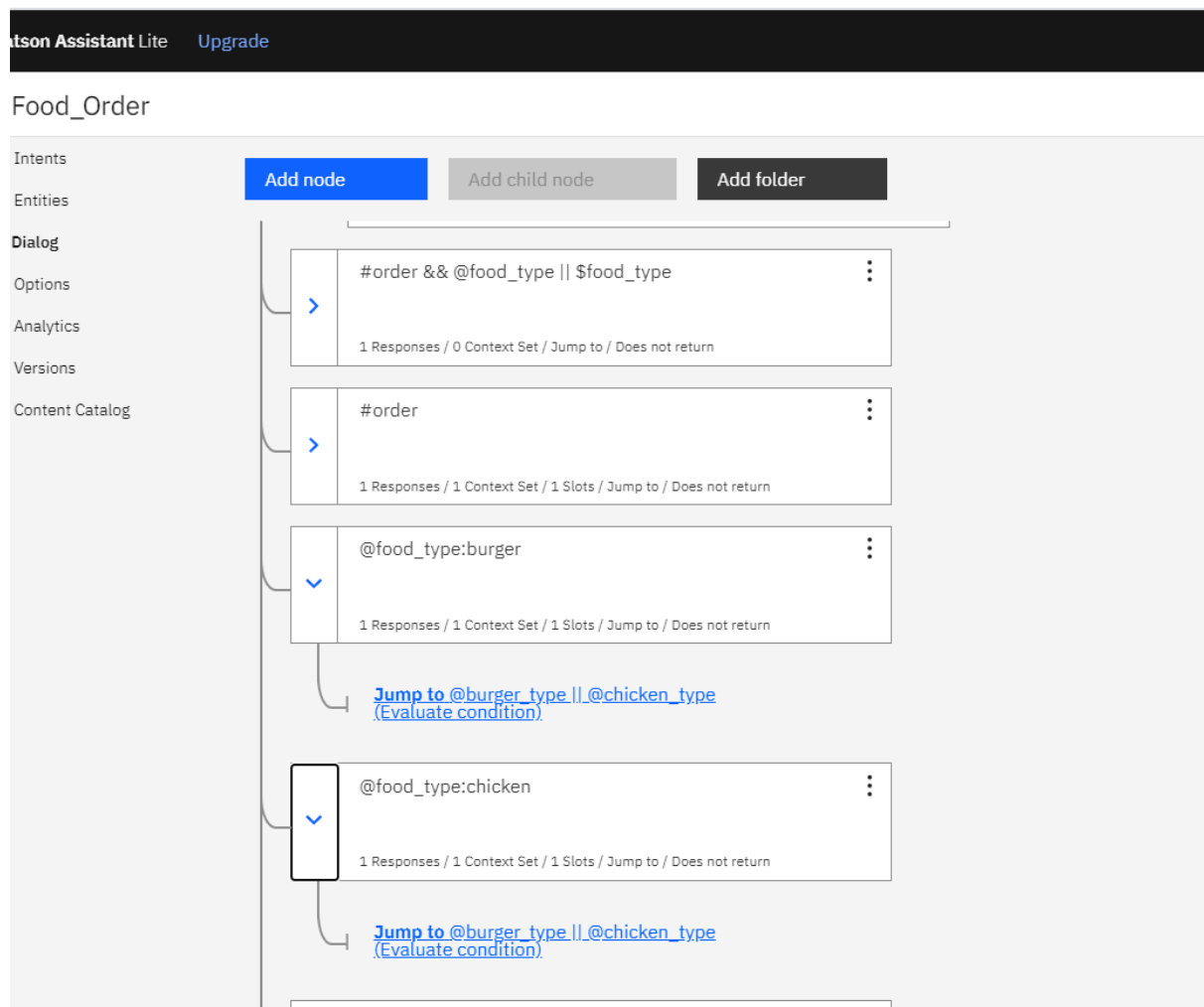
Add a child node of the node we just created, i.e. `@burger_type` || `@chicken_type`. Click on the three vertical dots and click on the *Add child node* option from the dropdown (from Figure 2.24). A new child node will be created beneath `@burger_type` || `@chicken_type` node.

Set the condition as *\$address* as we want to only check if the user has provided the address and set a test response such as “We are sending your order to your mentioned address in 30 minutes.” (refer Figure 2.40).



[Figure 2.40]

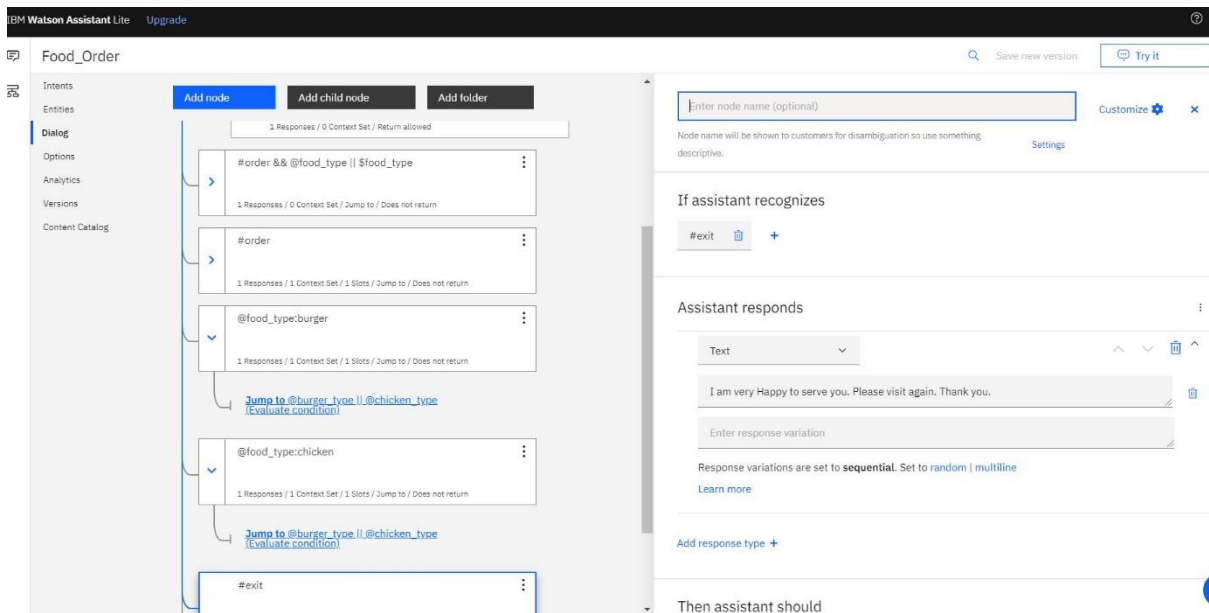
We haven't yet set the jump statements for *@food\_type:burger* and *@food\_type:chicken* nodes (Step 5 and 6 of our conversation model). So add the jump statements to these nodes to jump to *@burger\_type || @chicken\_type* node as destination with *If assistant recognizes (condition)*.



[Figure 2.41]

Now we are left with the last node to set, which is our last step in conversation model.

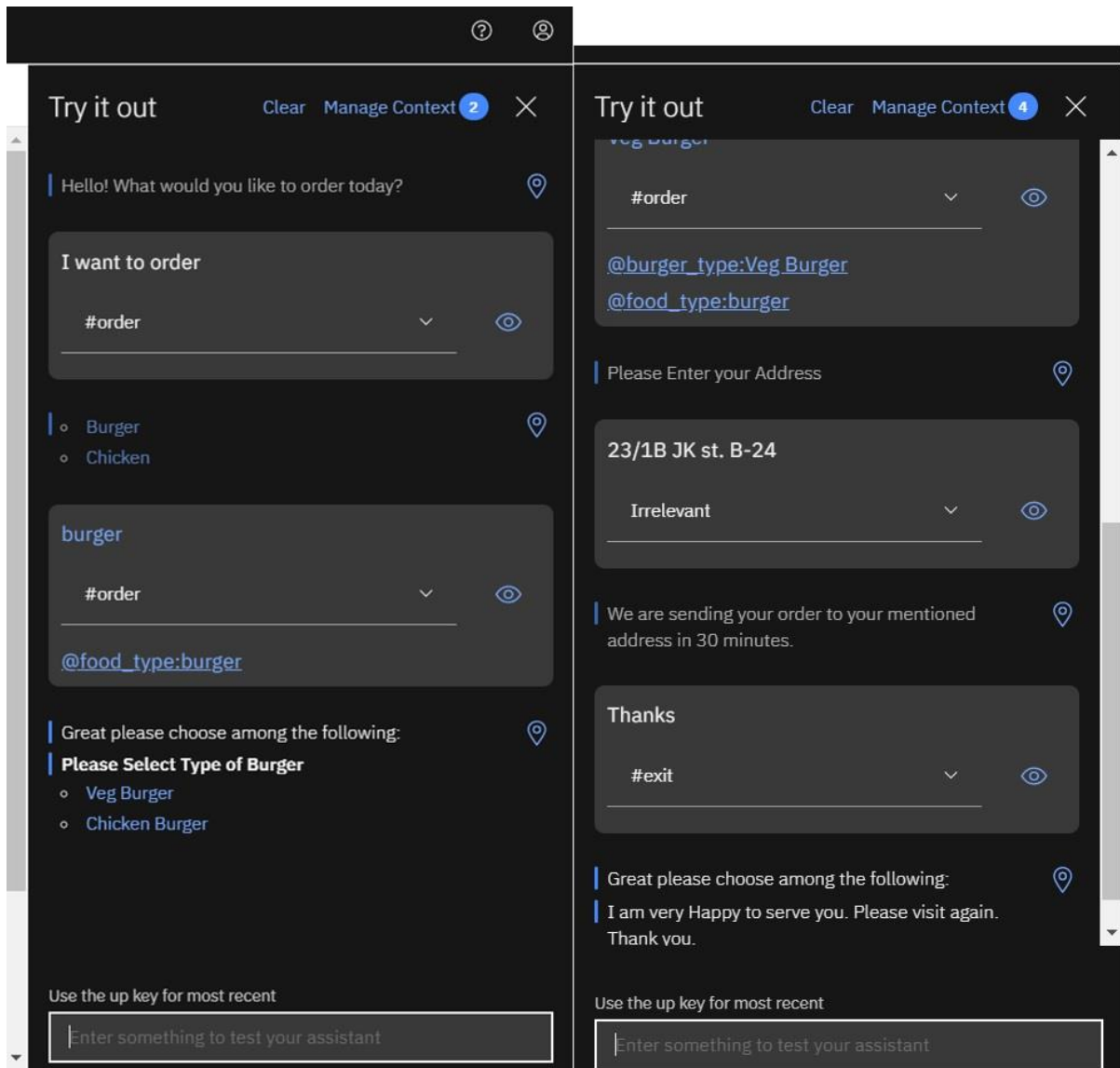
So create a node just above *Anything else* node. Set the condition as *#exit* intent and keep a text response “I am very Happy to serve you. Please visit again. Thank you.” As shown in the Figure 2.42.



[Figure 2.42]

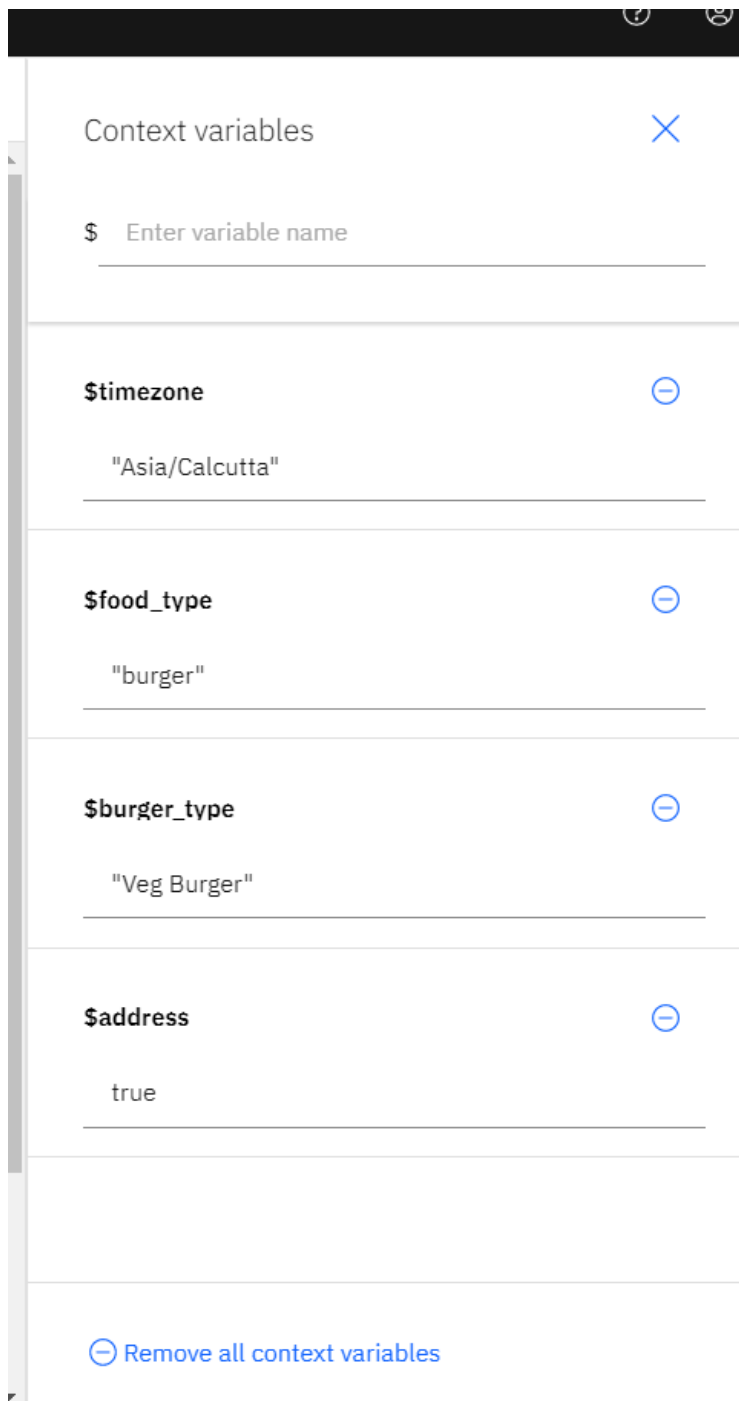
Congratulations!! You have a complete chat bot for food ordering.

Go ahead and test by clicking on the *Try it* button on the top left of the page. Go ahead and ask the assistant "I want to order" or "I want to order a burger". Some of the tests I conducted are given in figure 2.43 and 2.44. At times the Watson might be training while you are going to try out and might ask you to wait for some time.



Notice that at each conversation it tells us what intent and entities it found. You can click on the conversations to see which node it corresponds in the diagram.

Moreover you can click on the *Manage Context* on the top left to check what context variables it has stored.



[Figure 2.44]

As you can see it already has a few system variables (like *\$timezone*) along with the context variables we defined.

Test out a bit more of the chat bot you just created and in the next chapter we will see how to deploy this into production.