IBM Watson

Building Conversational Solutions

Building a Dialog

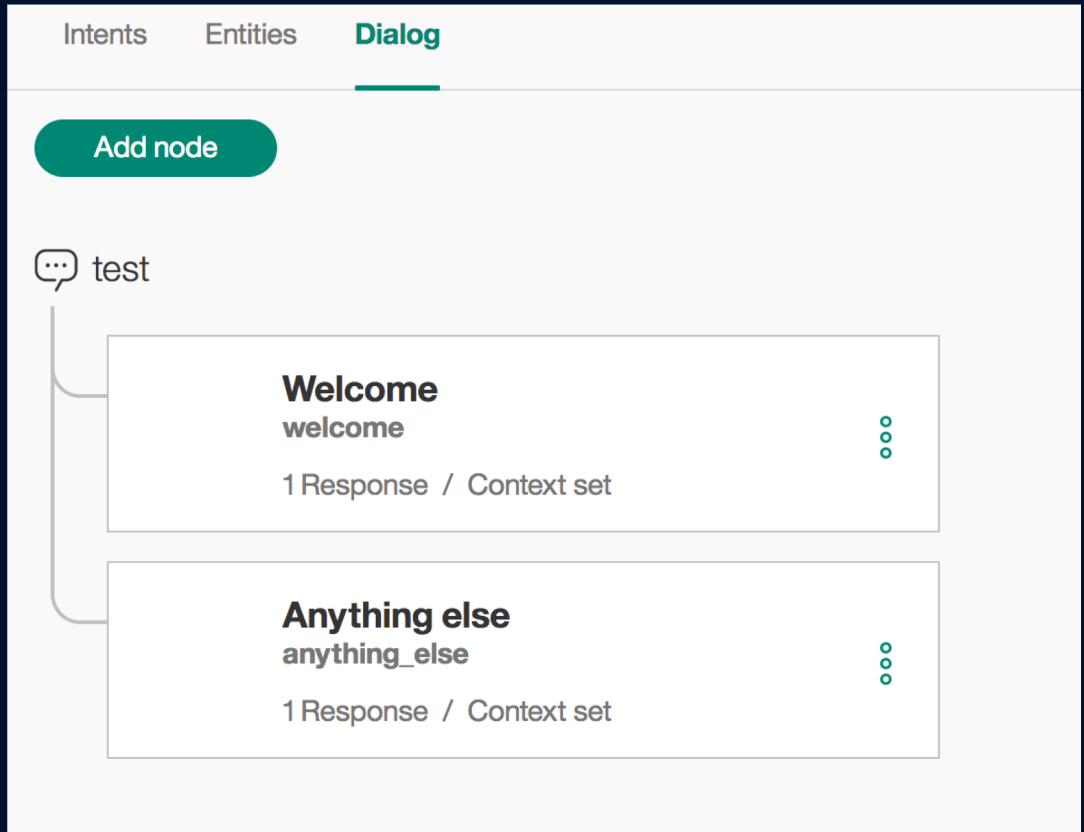
Aco Vidovič

IBM

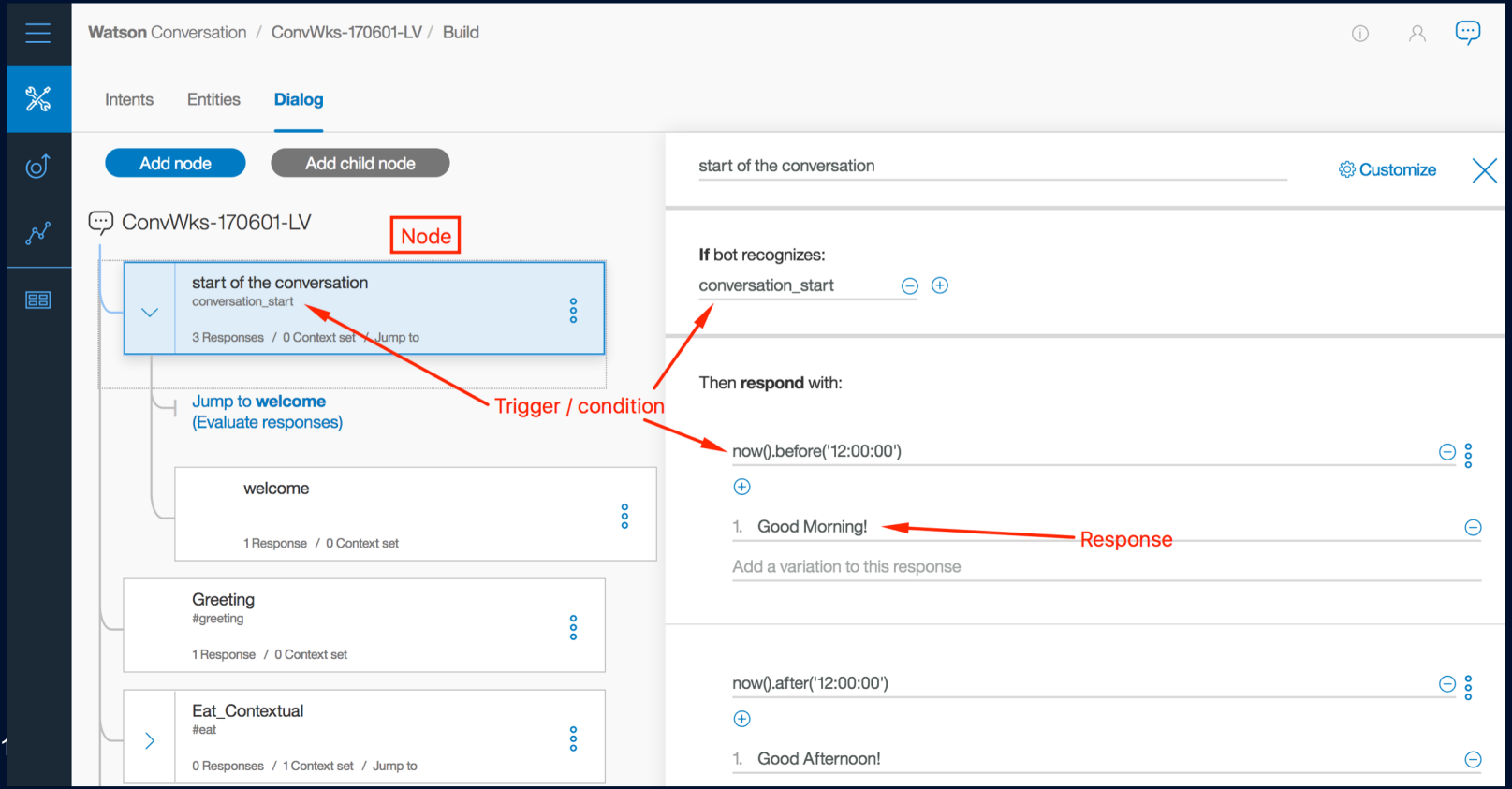
Agenda

- Dialog & node overview
- Folders
- Dialog Turn
- Digressions
- Node Conditions
- Node responses
- Slots
- JSON editor
- IBM Cloud Function

Getting started / anything_else



Dialog refresh



IBM Watson

Node description

Name this node... **Node Name:** Name your node in this field.

Customize

Customize Widget: Turn on Slots & multiple responses.

If bot recognizes:

Enter an intent, entity or context variable... **Condition:** Enter the main condition in this field.

Then respond with:

Advanced editor menu: Delete a response or open the Json editor for the response field.

Text

Move:

Enter response text **Response:** Enter a response for the condition in this field.

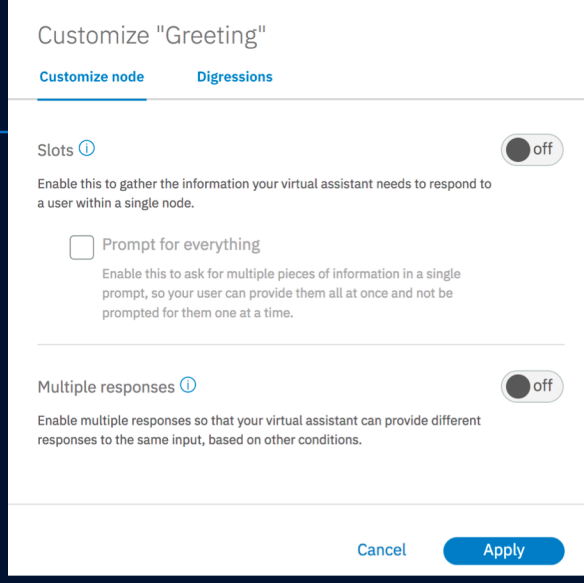
Response variations are set to **sequential.** Set to random

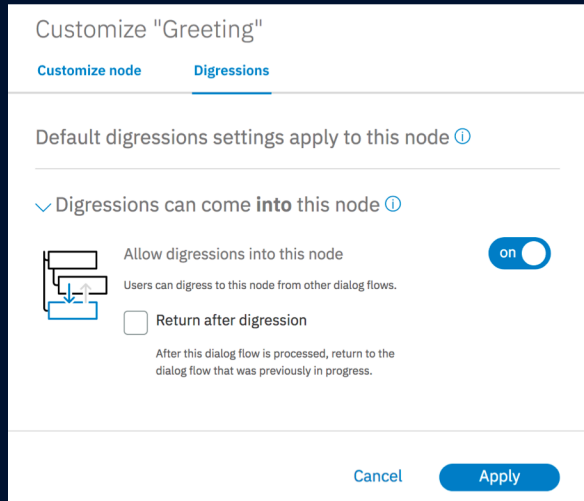
Add response type

And finally

Wait for user input

Next action menu: Drop down menu to choose which action Watson will take next after it returns a response to the user.





IBM Watson

Name this node...

Customize

IBM

If bot recognizes:

Enter an intent, entity or context variable...

Slots Frame

Then check for:

0 Manage handlers

	Check for	Save it as	If not present, ask	Type
1	Enter entity or intent	Enter context variable	Enter a prompt	Optional

Add slot

Multi responses frame

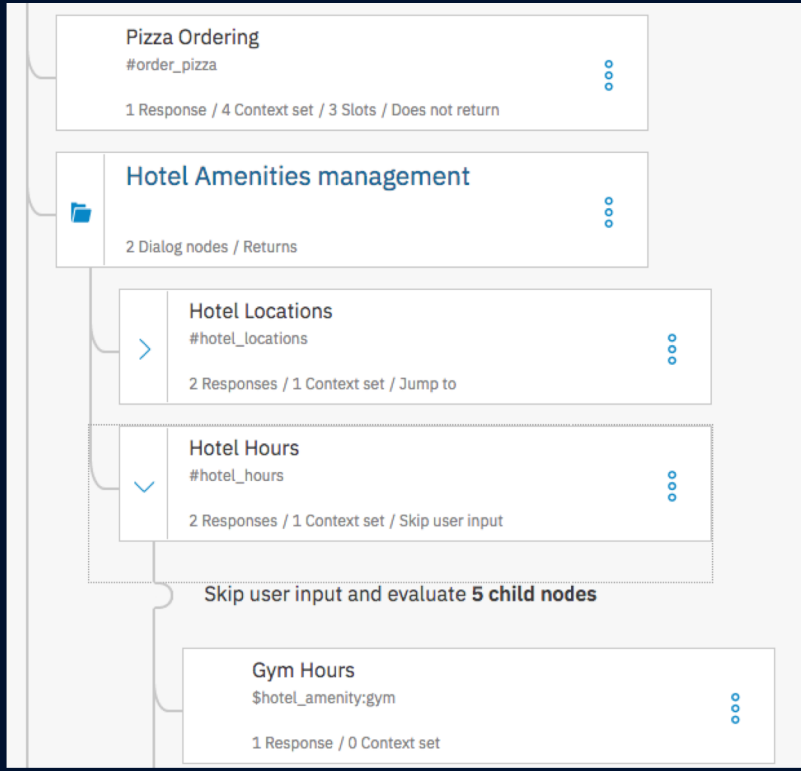
Then respond with:

	If bot recognizes	Respond with
1	Enter an intent, entity or context variab	Enter a response...

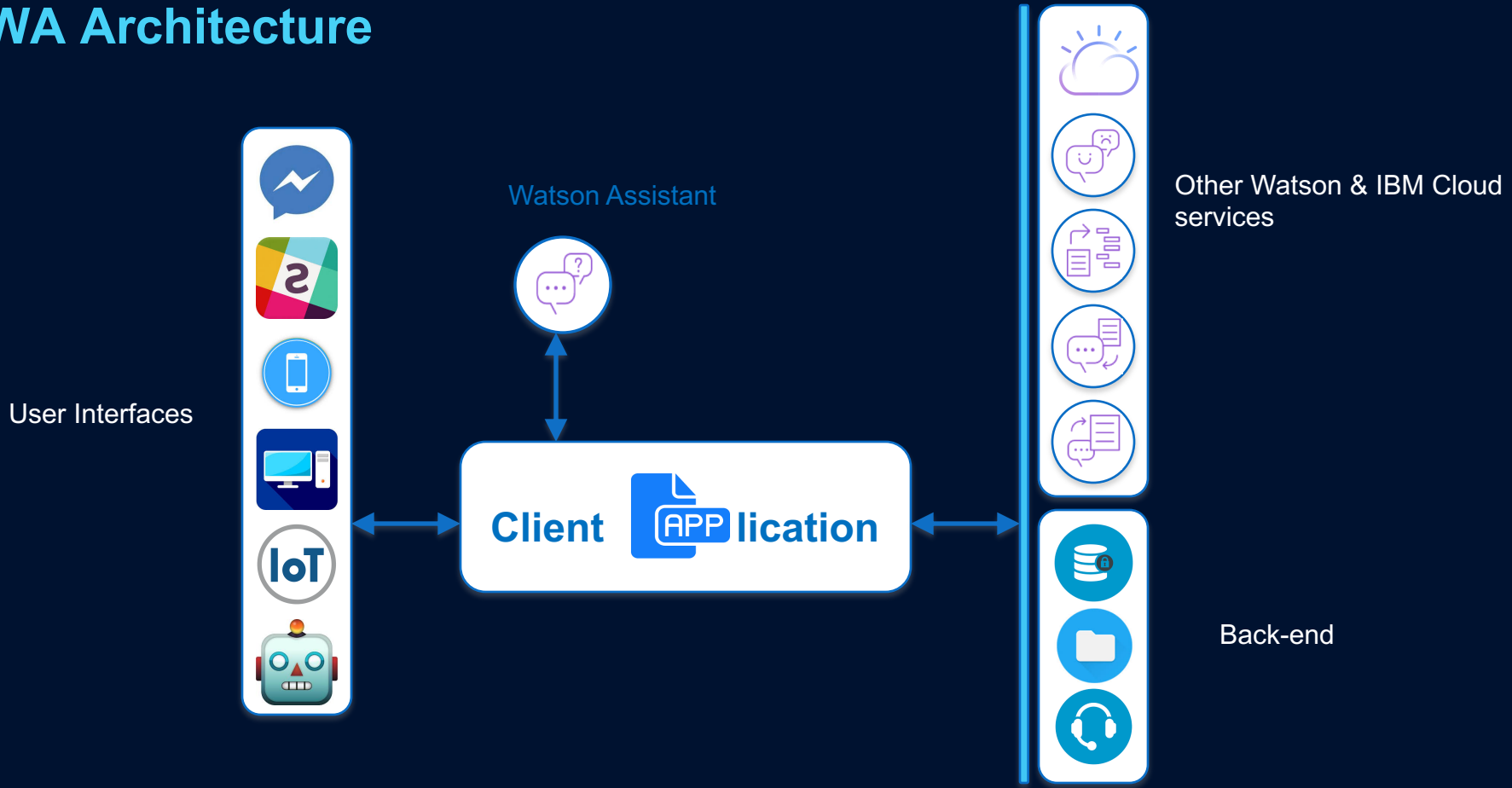
Add response

Folders

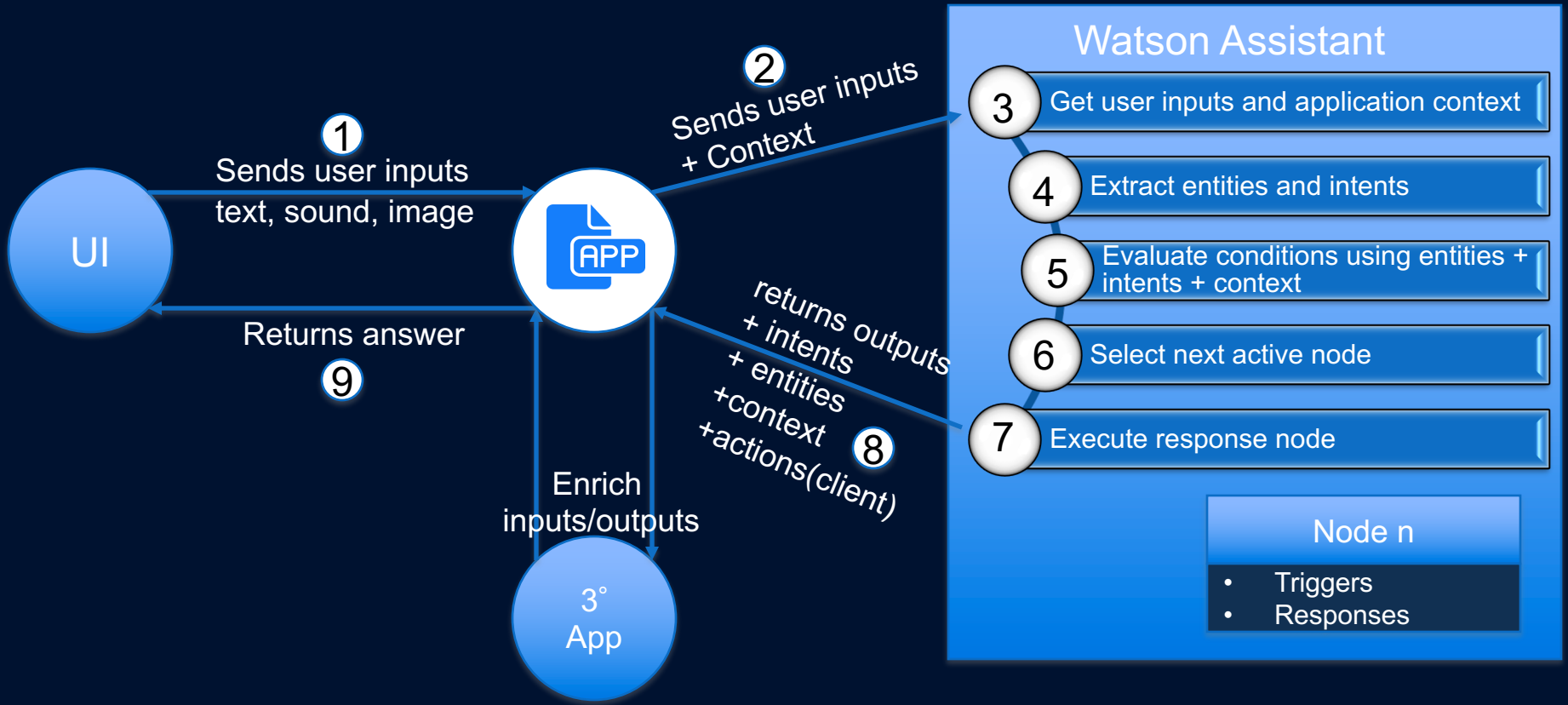
- Allow a dialog designer to organize content based on topics
- Much easier dialog tree navigation and understanding
- Allow performing of bulk setting of node settings at the folder level instead of one by one
- Easier separation of duties for multiple people working on the same bot



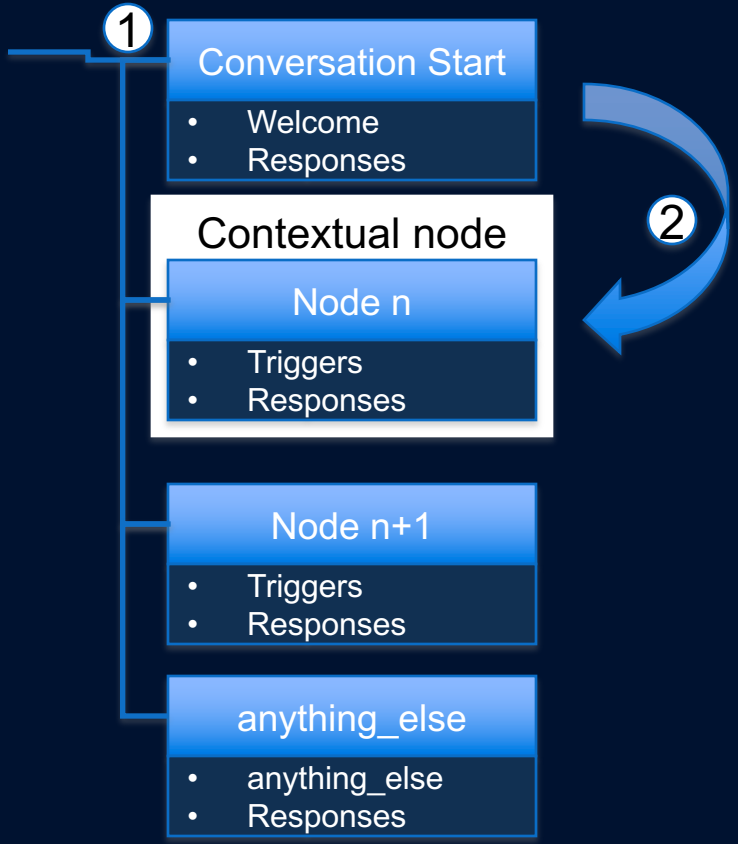
WA Architecture



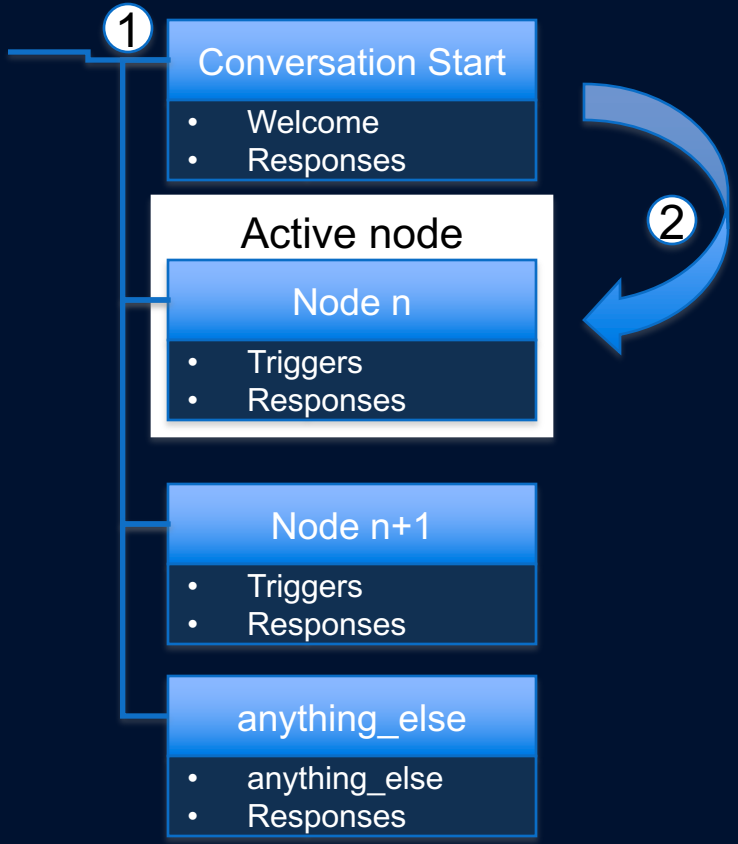
Conversation turn



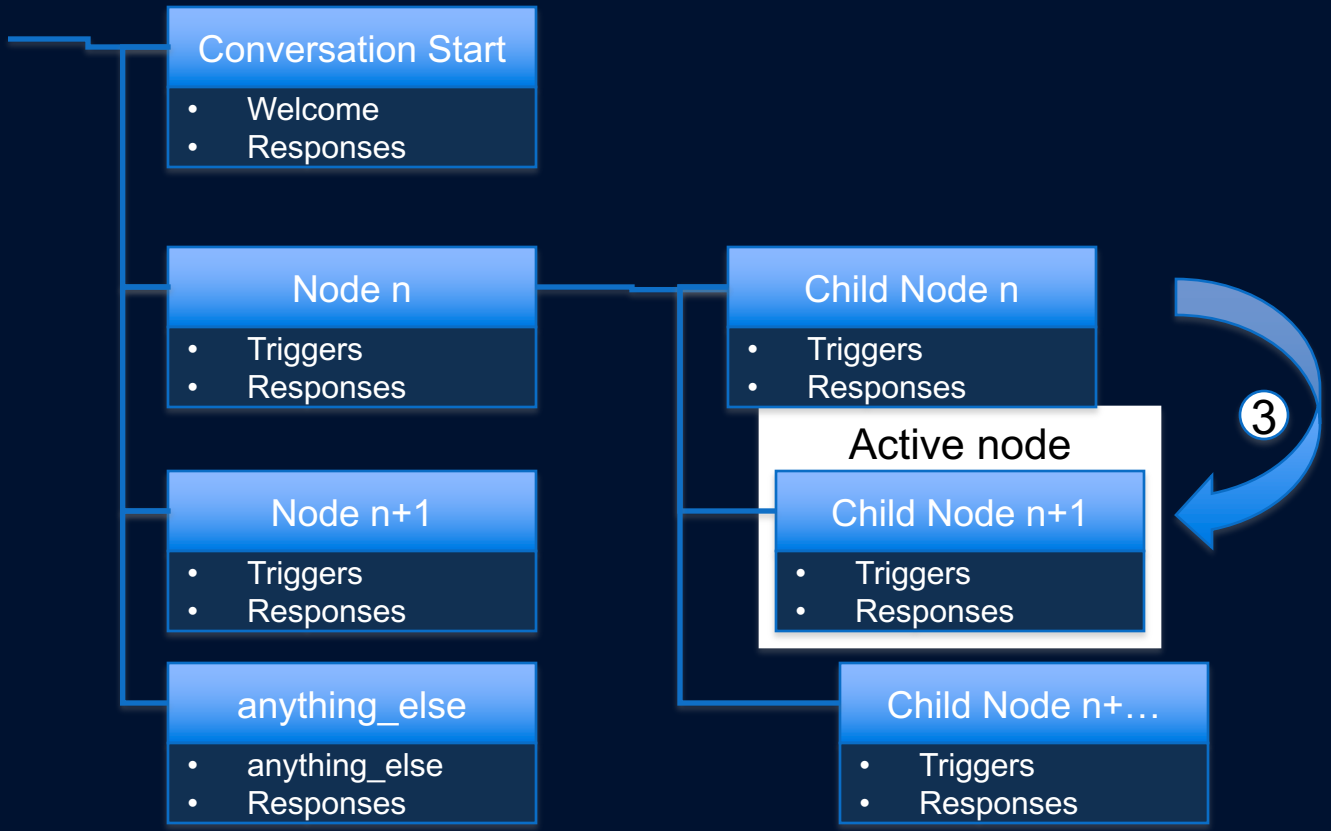
Dialog flow


1
Conversation Start
• Welcome
• Responses
2
Contextual node
Node n
• Triggers
• Responses
Node n+1
• Triggers
• Responses
anything_else
• anything_else
• Responses

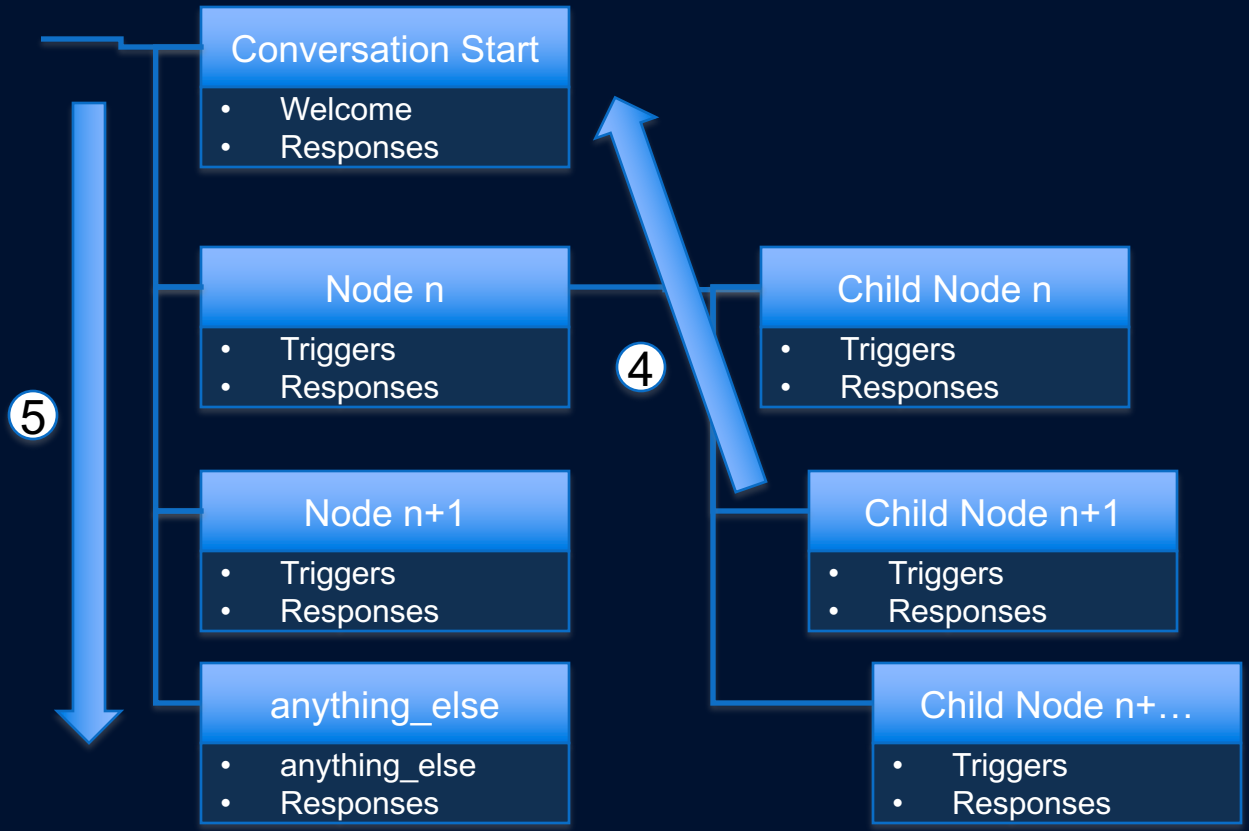
Dialog flow


1
Conversation Start
• Welcome
• Responses
Active node
Node n
• Triggers
• Responses
2
Node n+1
• Triggers
• Responses
anything_else
• anything_else
• Responses

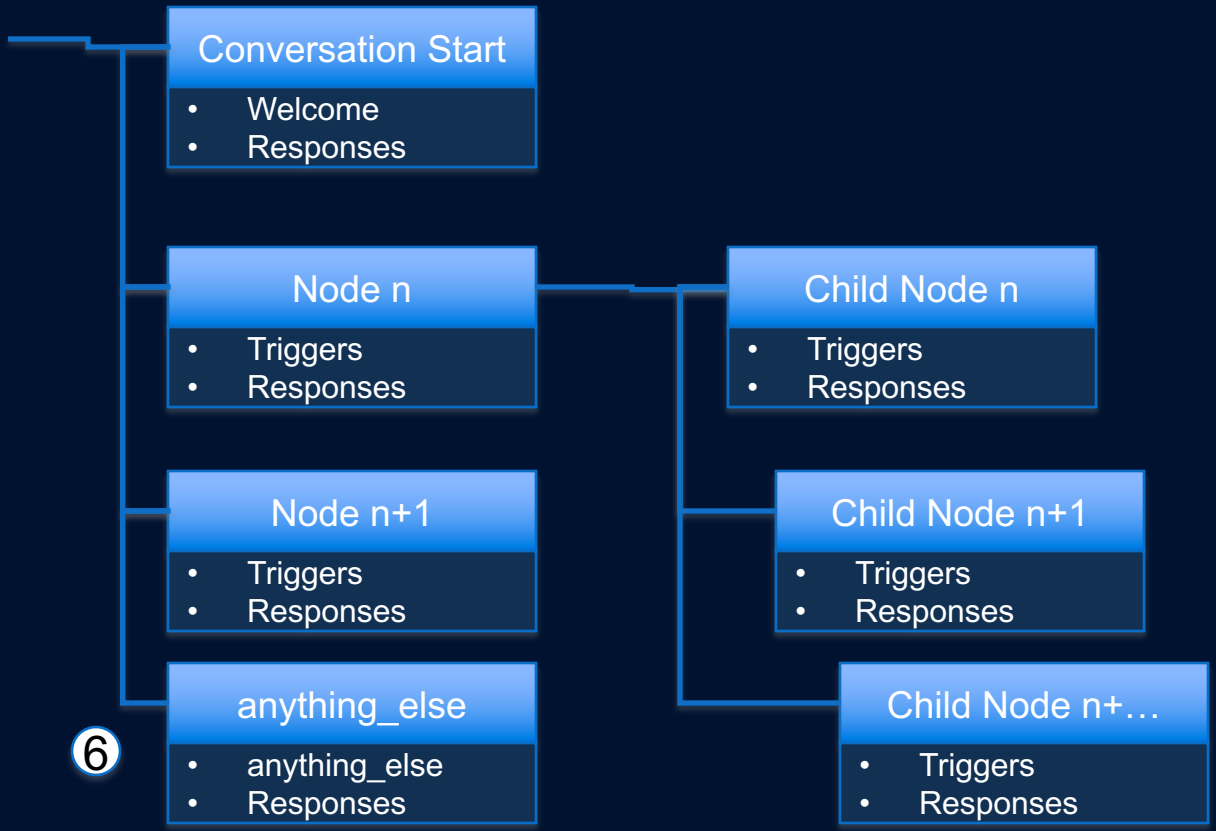
Dialog flow


Conversation Start
Welcome
Responses
Node n
Triggers
Responses
Node n+1
Triggers
Responses
anything_else
anything_else
Responses
Child Node n
Triggers
Responses
Active node
Child Node n+1
Triggers
Responses
3
Child Node n+…
Triggers
Responses

Dialog flow


Conversation Start
• Welcome
• Responses
Node n
• Triggers
• Responses
Node n+1
• Triggers
• Responses
anything_else
• anything_else
• Responses
Child Node n
• Triggers
• Responses
Child Node n+1
• Triggers
• Responses
Child Node n+…
• Triggers
• Responses
4
5

Dialog flow

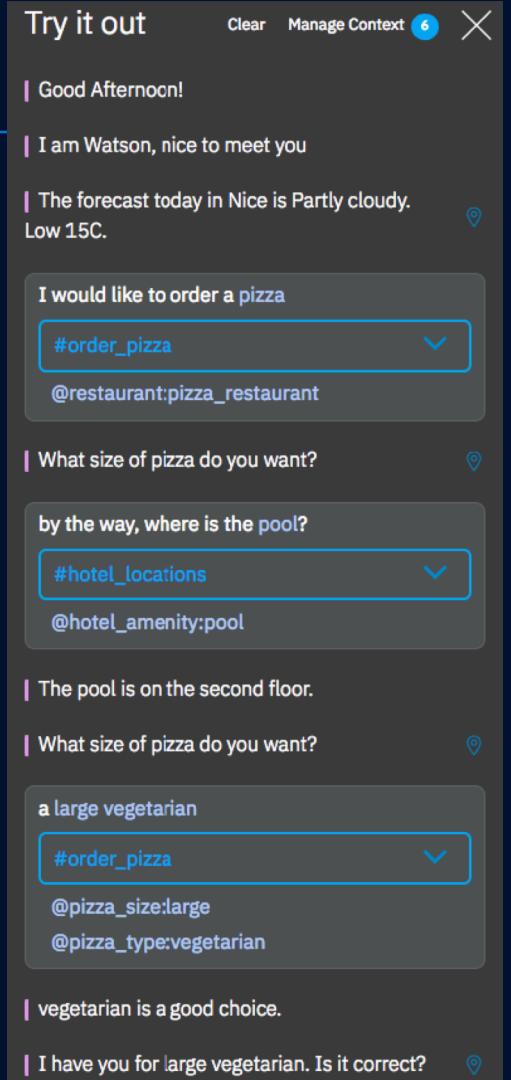

Conversation Start
Welcome
Responses
Node n
Triggers
Responses
Node n+1
Triggers
Responses
anything_else
anything_else
Responses
Child Node n
Triggers
Responses
Child Node n+1
Triggers
Responses
Child Node n+…
Triggers
Responses

IBM Watson

Digressions

- Allow a user to naturally jump out of a process and seamlessly return back

- Typical use cases
  - Asking side questions while going through a process
  - Chit chat during an unrelated conversation
  - Changing topics without abandoning high value processes



Defining a Condition / Trigger

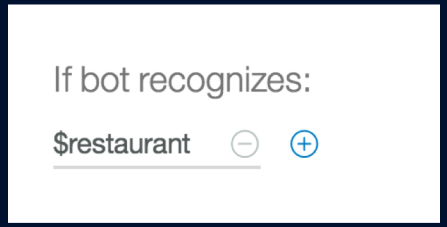
- The simplest condition is a single intent : #intent
- The check for a particular value for an entity: @entity:value
- The check for any value for an entity: @entity
- The check for not existing object : !@entity, !#intent
- The check for context variable: $variable_name:value
- Value can be evaluated: @price<100 ; @price AND @price<100
- Conditions can be combined : and, && , or , || , ()
- input.text.matches('^[^\d]*[\d]{11}[^\d]*$')

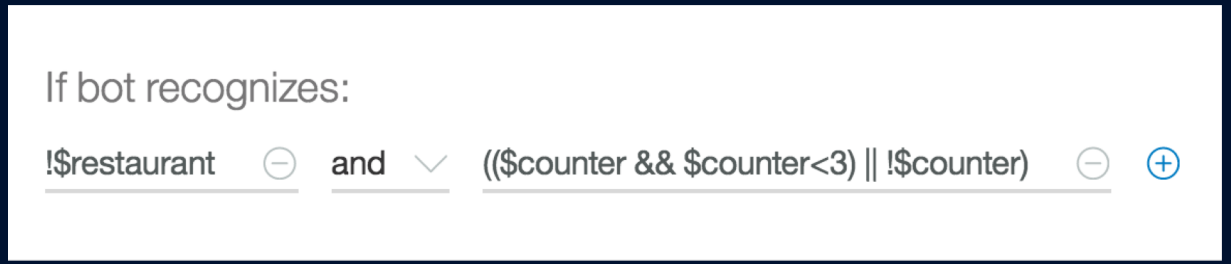
Note : conditions are written in the Spring Expression (SpEL) language and (Regex) Regular Expression

Special Conditions / Triggers

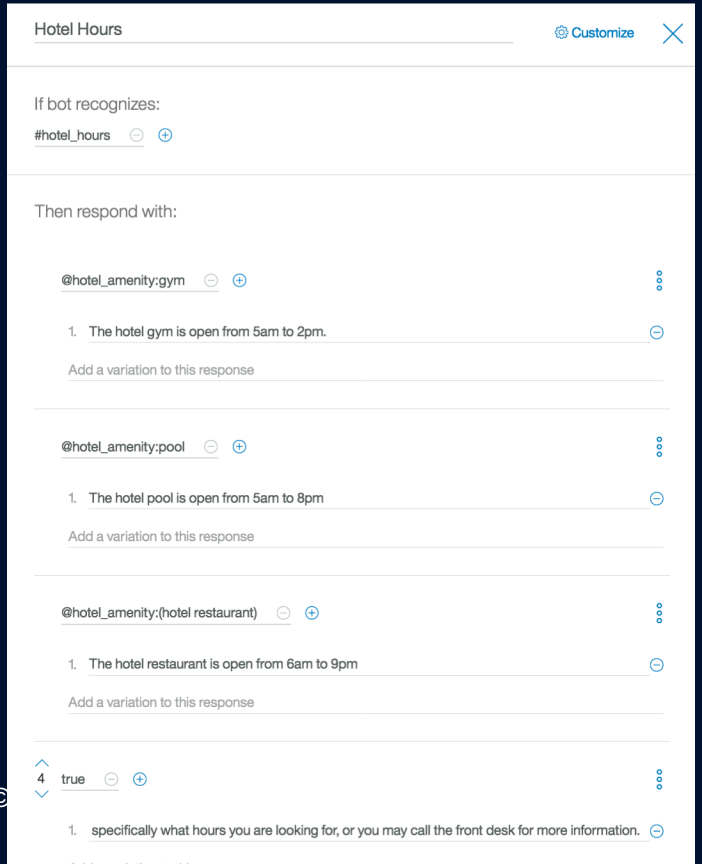
- **welcome**: This condition is evaluated as true during the first dialog turn (when the conversation starts), only if the initial request from the application does not contain any user input. It is evaluated as false in all subsequent dialog turns.
- **conversation_start**: Like **welcome**, this condition is evaluated as true during the first dialog turn, but unlike **welcome**, it is true whether or not the initial request from the application contains user input.
- **anything_else**: You can use this condition at the end of a dialog, to be processed when the user input does not match any other dialog nodes.
- **irrelevant**: This condition will evaluate to true if the user's input is determined to be irrelevant by the Conversation service.
- **true**: This condition is always evaluated to true. You can use it at the end of a list of nodes or responses to catch any responses that did not match any of the previous conditions.
- **false**: This condition is always evaluated to false. You might use this at the top of a branch that is under development, to prevent it from being used, or as the condition for a node that provides a common function and is used only as the target of a **Jump to** action.

Multi conditions / triggers

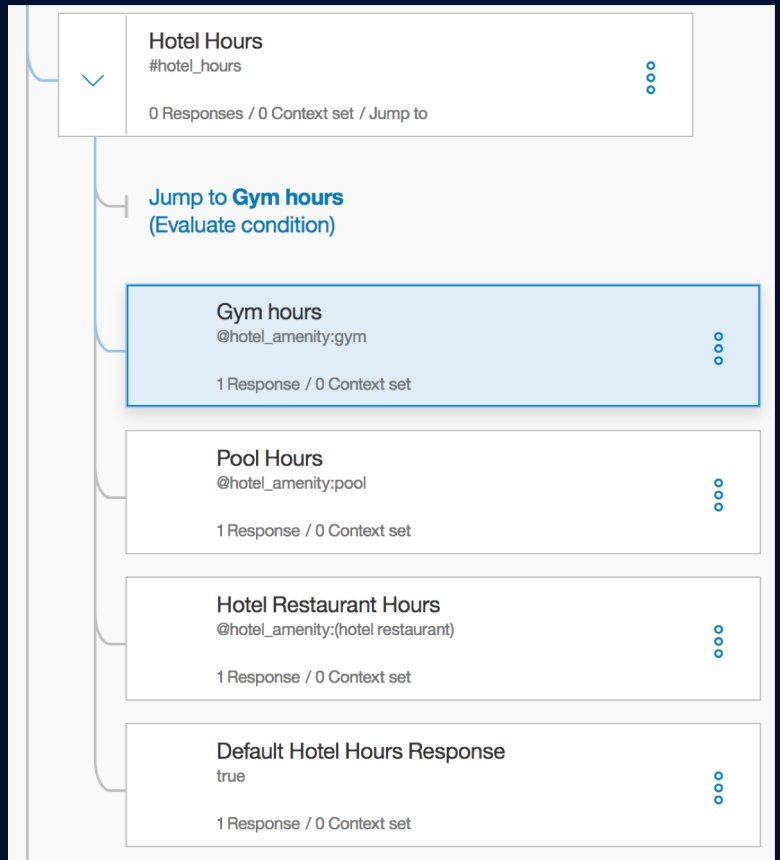




Multi conditions / triggers



OR



IBM Watson

Using Time Method / Context Variable

Responses

if now().before('12:00:00')

1. Good Morning!

Add a variation to this response

if now().after('12:00:00')

1. Good Afternoon!

Add a variation to this response

3

Add response condition

1. Hi!

Add a variation to this response

Fulfill with a response

Jump to...

if $time == "morning"

1. Good morning

Add a variation to this response

if $time == "evening"

1. Good evening

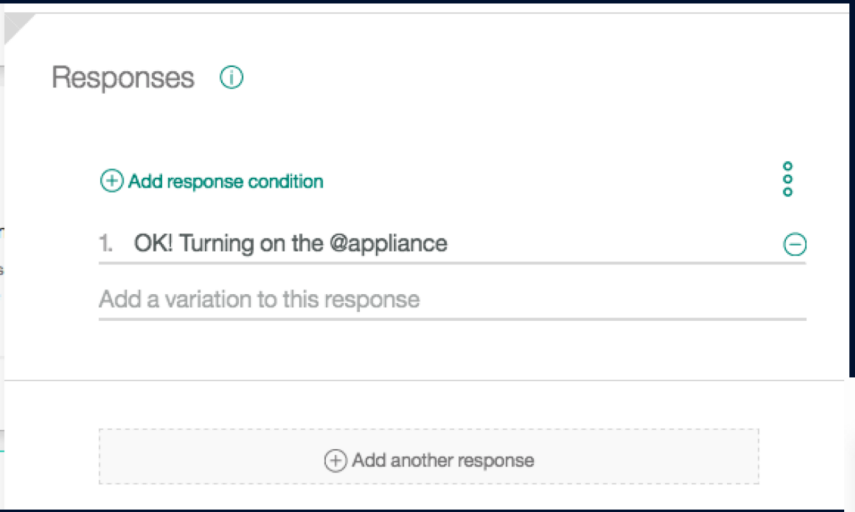
Add a variation to this response

if anything_else

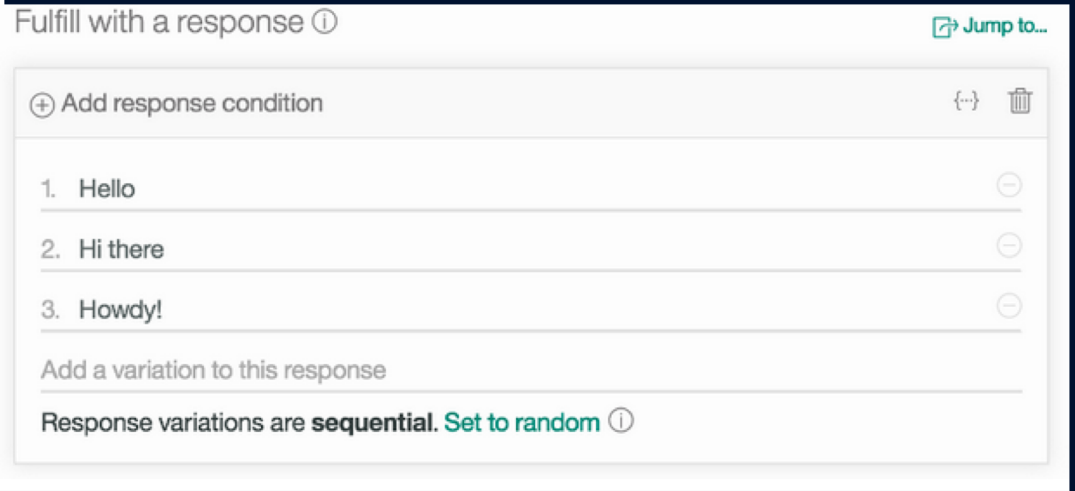
1. Hello

Add a variation to this response

Defining a response



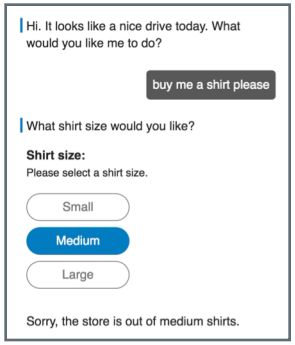
1. Update the dialog context;
2. Provide a response to the user;
3. What to do next!
   - Standby and **Wait for user input** (default)
   - Execute child nodes with **Skip user input** option
   - Change the flow of the dialog by using a **Jump to** action

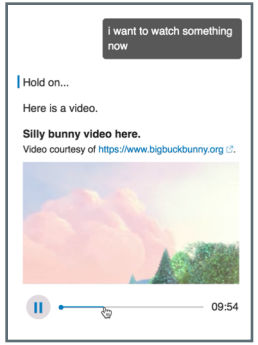


Response Types

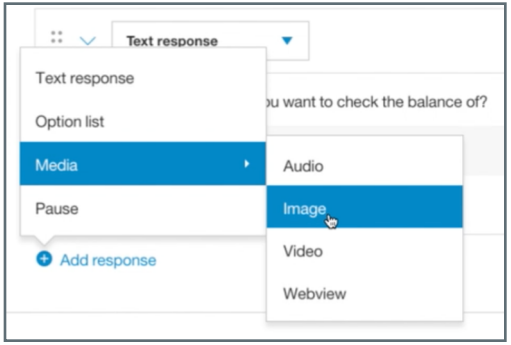
Enable end user experiences beyond simple text with **response types**
E.g. Buttons, images, videos, pause delays etc.

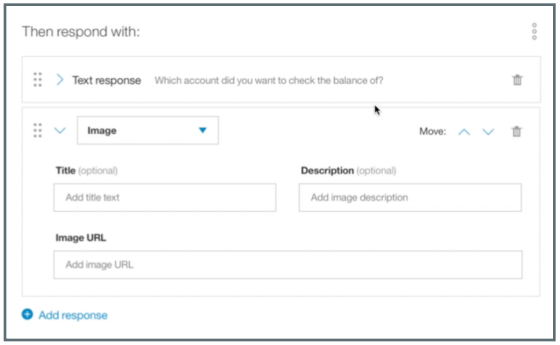
End User Experience





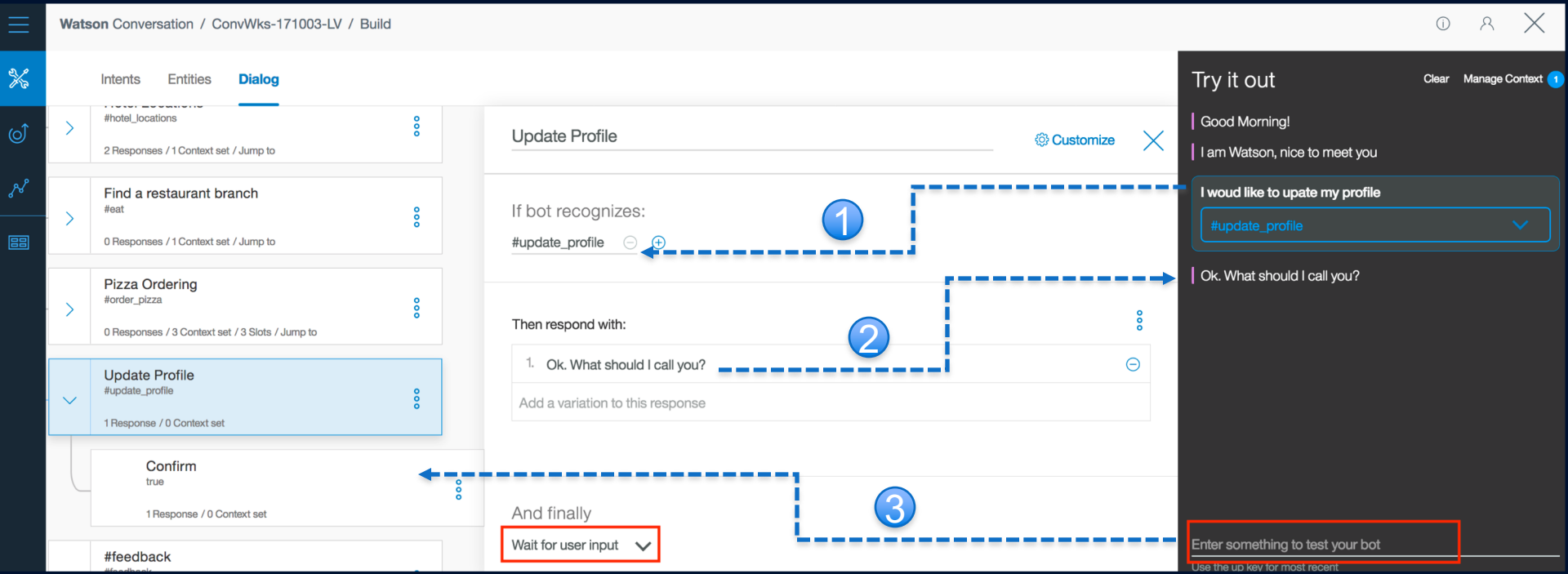
Authoring Experience (Dialog)





IBM Watson

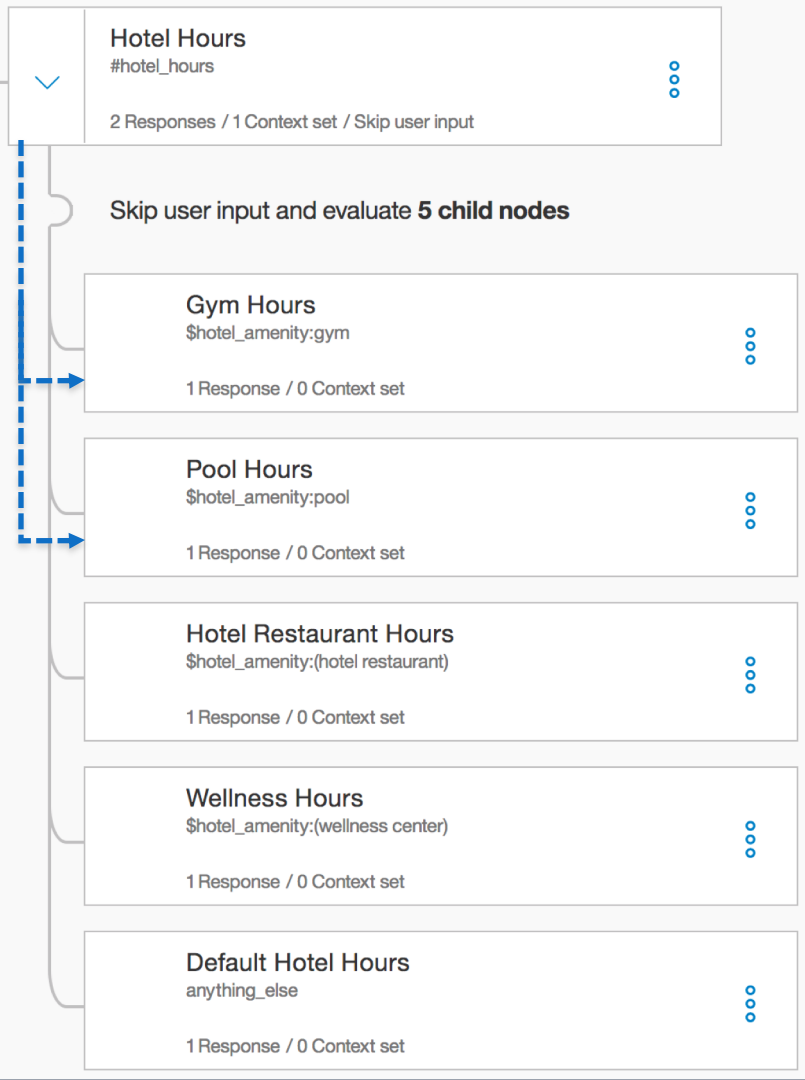
Wait for user input



IBM Watson

Skip user input ... acts like a **GoTo** child nodes

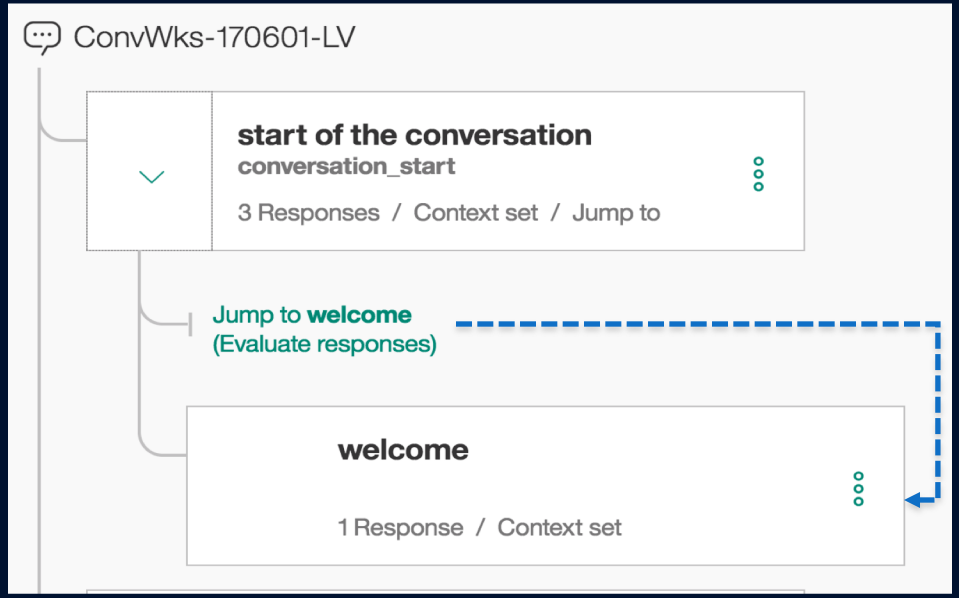
- Doesn't wait for user input

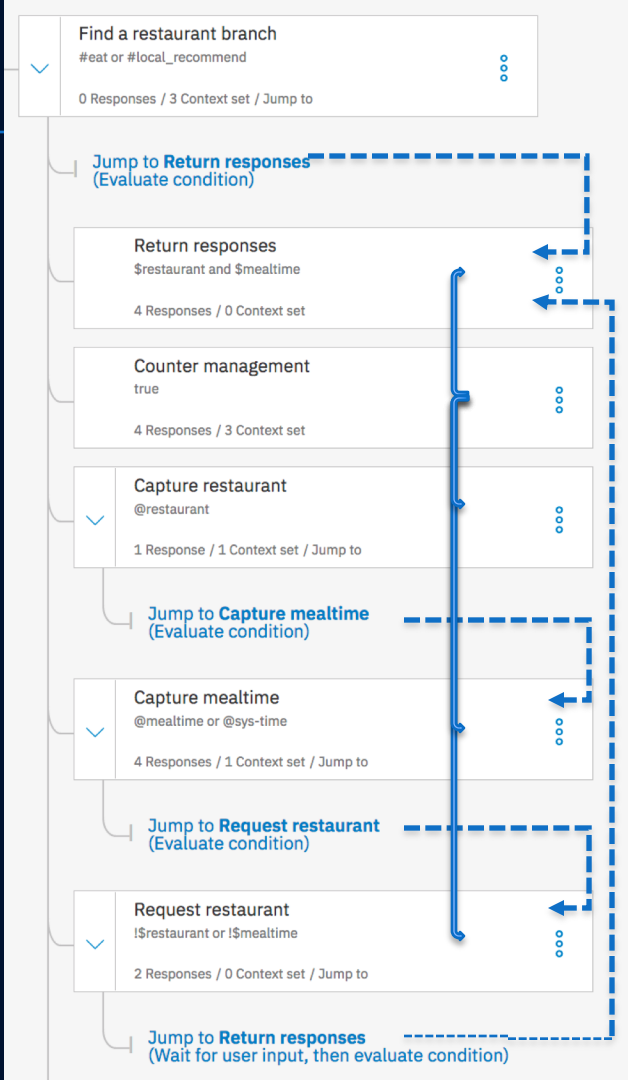


IBM Watson

Jump to ... acts like a GoTo

- Condition without waiting for user input
- Response without waiting for user input
- Wait for user input and then condition

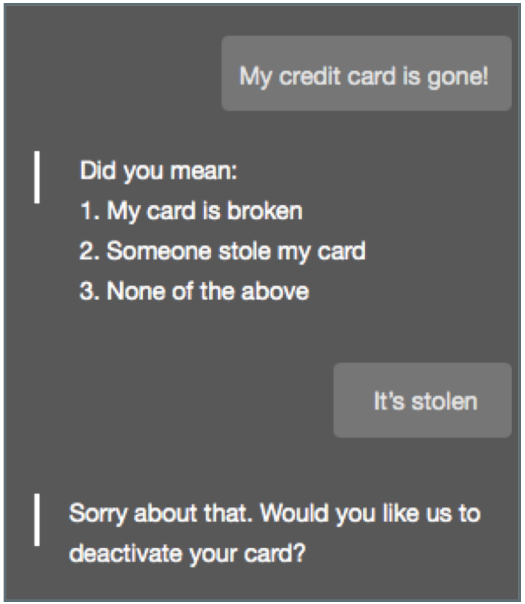




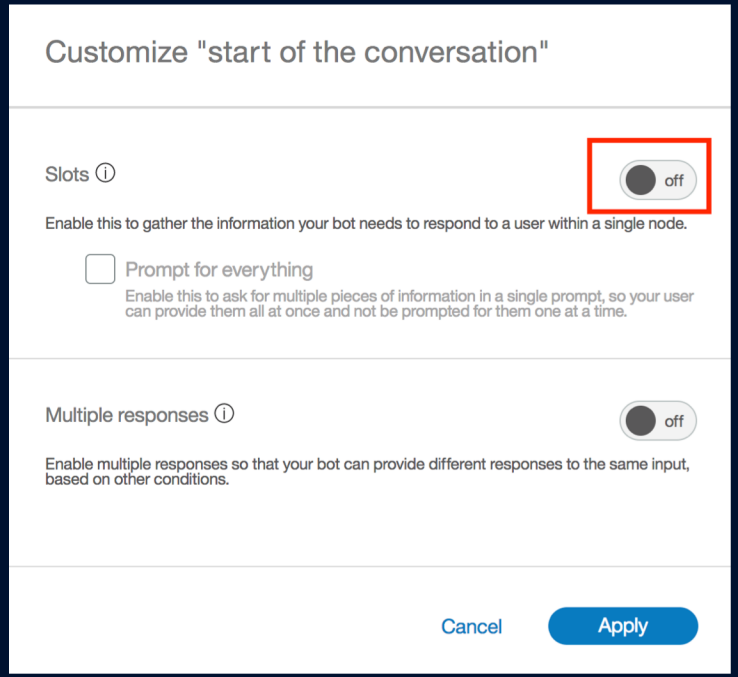
Disambiguation – Premium only

Disambiguate a users utterance (Did you mean....)

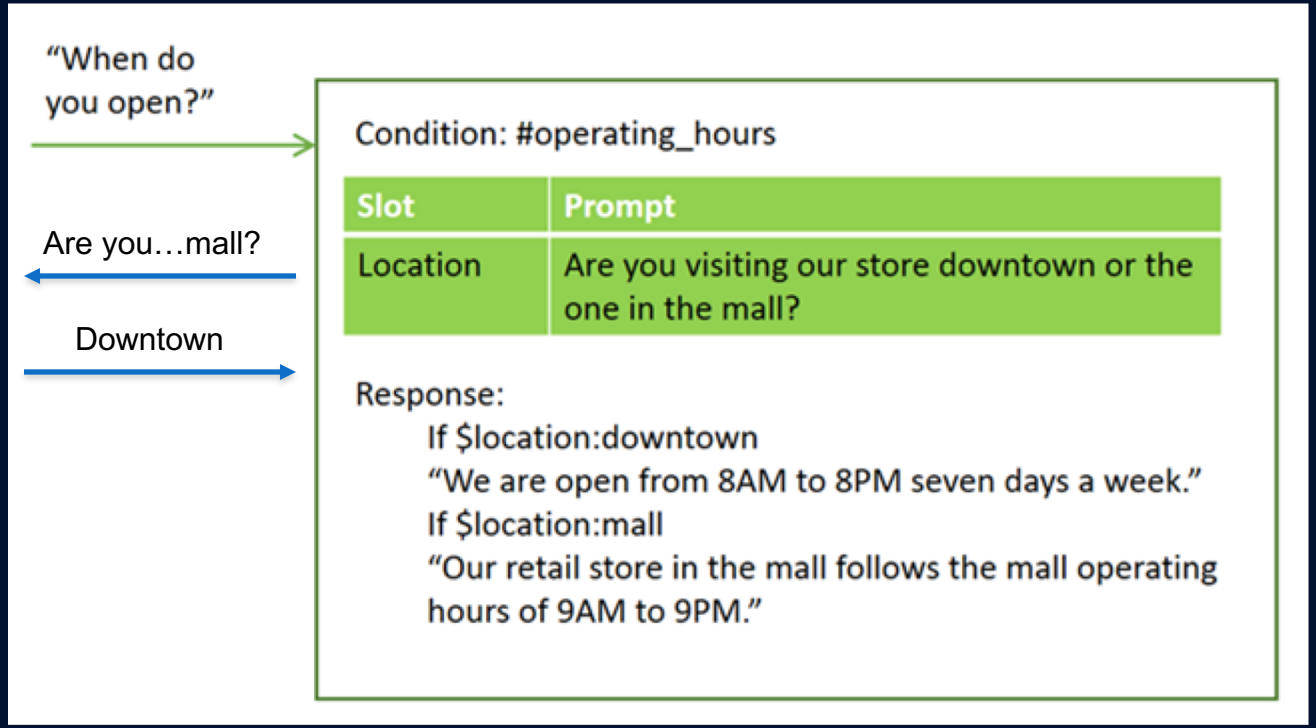
- End-user picks best option from a list
- Let's your virtual assistant help customers better, which means less transfers over to live agent



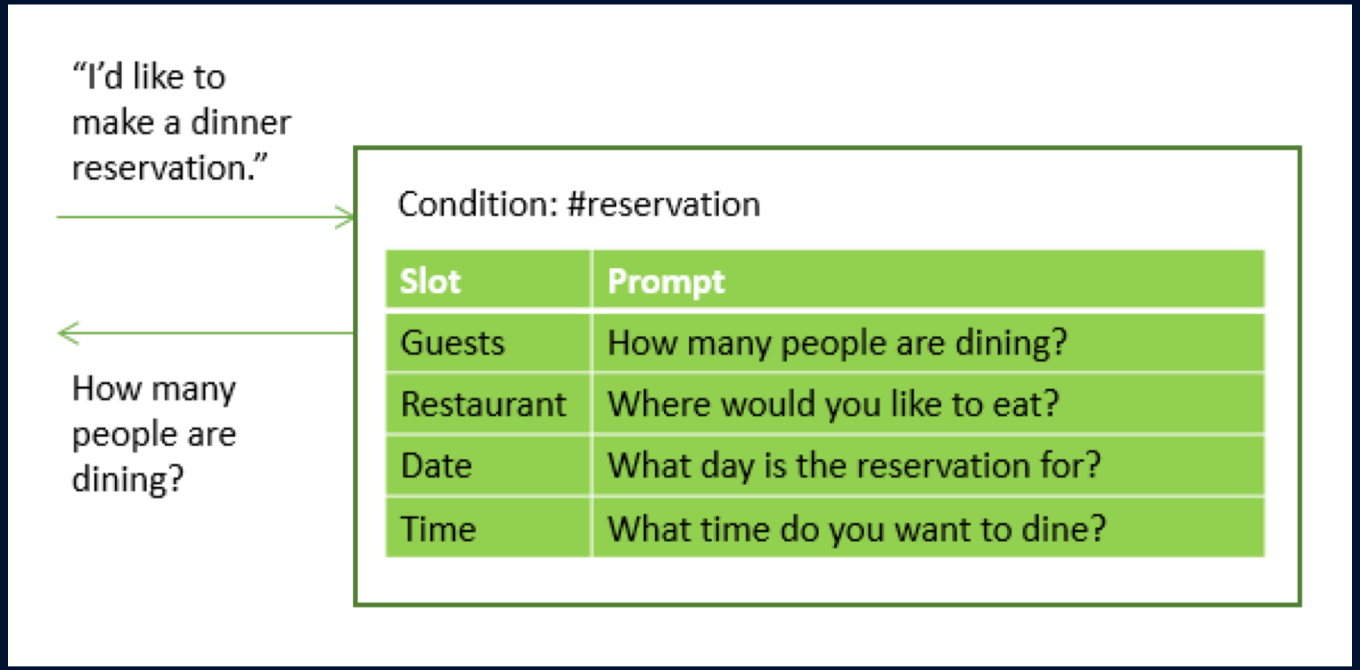
Slots: Ask for everything at once



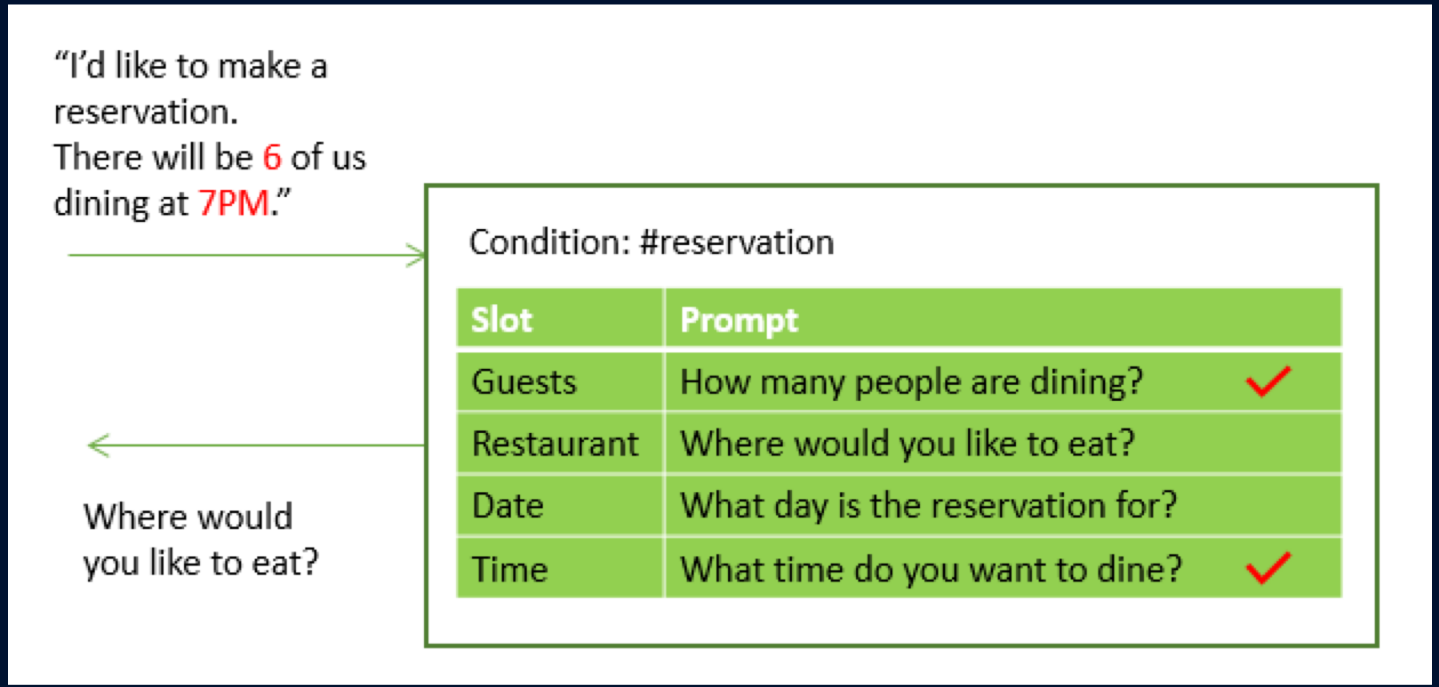
How slots Work



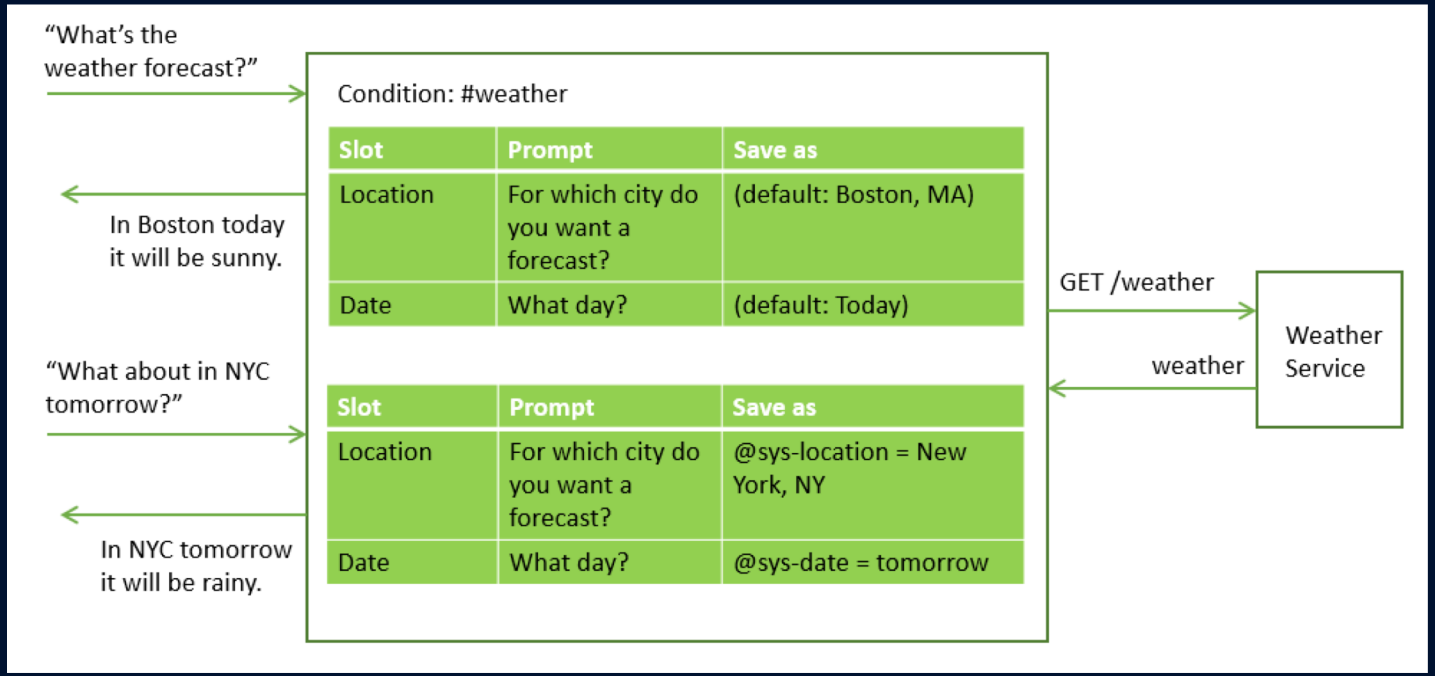
Slot: Collect multiple pieces of information



Slot: Collect multiple pieces of information



Update a collected value



Slot: user inputs and follow up

Information	Check for	Save as	Prompt	Follow-up if found	Follow-up if not found
Size	@size	$size	"What size pizza would you like?"	"$size it is."	"What size did you want? We have small, medium, and large."
DeliverBy	@sys-time	$time	"When do you need the pizza by?"	"For delivery by $time."	"What time did you want it delivered? We need at least a half hour to prepare it."

Slot: Optional slot

Information	Check for	Save as	Prompt	Follow-up if found	Follow-up if not found
Size	@size	$size	"What size pizza would you like?"	"$size it is."	"What size did you want? We have small, medium, and large."
DeliverBy	@sys-time	$time	"When do you need the pizza by?"	"For delivery by $time."	"What time did you want it delivered? We need at least a half hour to prepare it."
Wheat restriction	@dietary	$dietary	*Empty*		

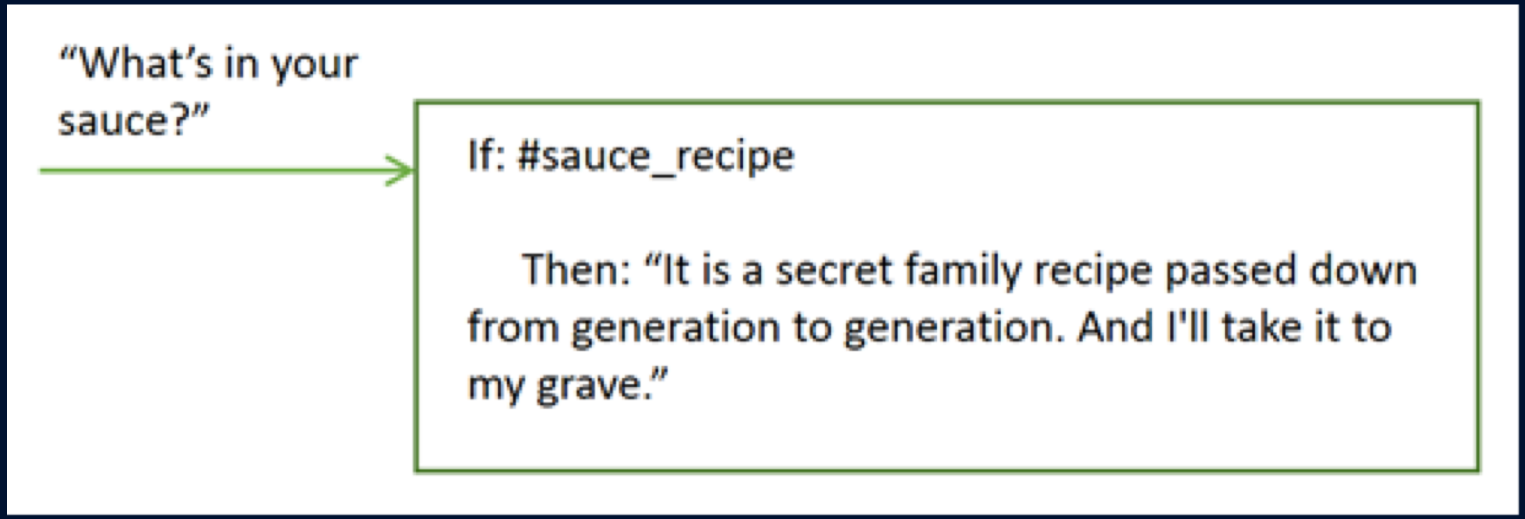
Slot: Multiple Values

Information	Check for	Save as	Prompt	Follow-up if found	Follow-up if not found
Size	@size	$size	"What size pizza would you like?"	"$size it is."	"What size did you want? We have small, medium, and large."
DeliverBy	@sys-time	$time	"When do you need the pizza by?"	"For delivery by $time."	"What time did you want it delivered? We need at least a half hour to prepare it."
Toppings	@toppings. values	$toppings	Any toppings on that?	"Great addition."	"What toppings would you like? We offer ..."

Slot: getting confirmation

Information	Check for	Save as	Prompt	Follow-up if found	Follow-up if not found
Size	@size	$size	"What size pizza would you like?"	"$size it is."	"What size did you want? We have small, medium, and large."
DeliverBy	@sys-time	$time	"When do you need the pizza by?"	"For delivery by $time."	"What time did you want it delivered? We need at least a half hour to prepare it."
Confirmation	#yes	$order-confirmed	"I'm going to order you a $size pizza for delivery at $time. Should I go ahead?"	"Your pizza is on its way!"	*See below*

Slot: Keep Users on track (handlers)



Specific slot's Conditions / Triggers

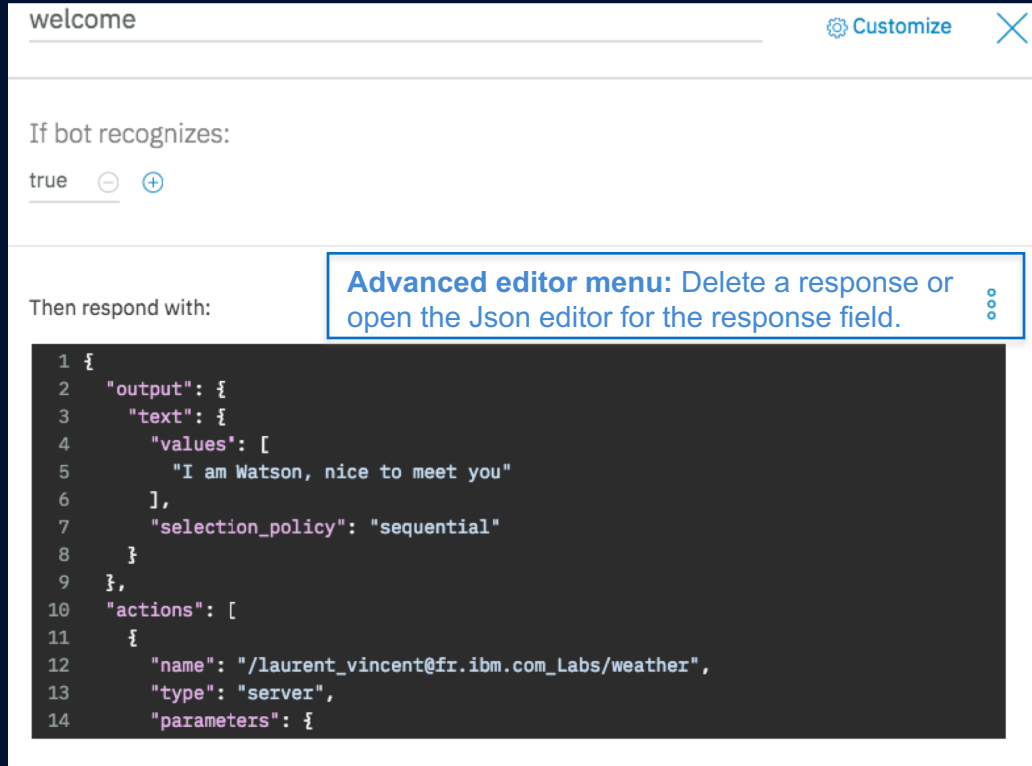
- slot_in_focus
  You are looking for a response for this slot
- event.previous_value
  You want to evaluate a change in the context variable of this slot. It associated with event.current_value
- all_slots_filled
  You want to assess that's all slots are filled

What to do next?

- Found
  - Move on (Default)
  - Clear slot and prompt again
  - Skip to response
- Not found
  - Wait for user input (Default)
  - Prompt again
  - Skip this slot
  - Skip to response
- Handler
  - Prompt again (default)
  - Skip current slot
  - Skip to response

Advanced Editor: Json editor

```
{
  "context": {
     "{any_objects}": " {any_value}"
   },
  "actions": [ {
     "name":"<actionName>",
     "type":"client | server",
     "parameters": {
       "<parameter_name>":"<parameter_value>",
       "<parameter_name>":"<parameter_value>" },
     "result_variable": "<result_variable_name>",
     "credentials": "<reference_to_credentials>"
  } ],
  "output": {
     "text": [ "text one" ],
     "generic": [ object ],
     "{action}": " {any_value}",
   }
}
```



Context

```
{
  "context": {
     "{any_objects}": " {any_value}"
   },
   "actions": [ {
     "name":"<actionName>",
     "type":"client | server",
     "parameters": {
       "<parameter_name>":"<parameter_value>",
       "<parameter_name>":"<parameter_value>" },
     "result_variable": "<result_variable_name>",
     "credentials": "<reference_to_credentials>"
   } ],
   "output": {
     "text": [ ”text one" ],
     ”generic": [ object ],
     "{action}": " {any_value}”,
   }
}
```

Context is what makes your bot more conversational

- Repeats
- Turn Counters
- Capturing information
- Analytics

Within Context, you can by using SpEL and Regex

- Manipulate info
- Increment/decrement values
- Reformat
- Simple Math
- Manage Array

Call to actions

```
{
  "context": {
     "{any_objects}": " {any_value}"
   },
   "actions": [ {
     "name":"<actionName>",
     "type":"client | server",
     "parameters": {
       "<parameter_name>":"<parameter_value>",
       "<parameter_name>":"<parameter_value>" },
     "result_variable": "<result_variable_name>",
     "credentials": "<reference_to_credentials>"
   } ],
   "output": {
      "text": [ "text one" ],
      "generic": [ object ],
      "{action}": " {any_value}",
   }
}
```

Actions enables a smooth integration

- Callout to cloud functions
- Callout to any APIs

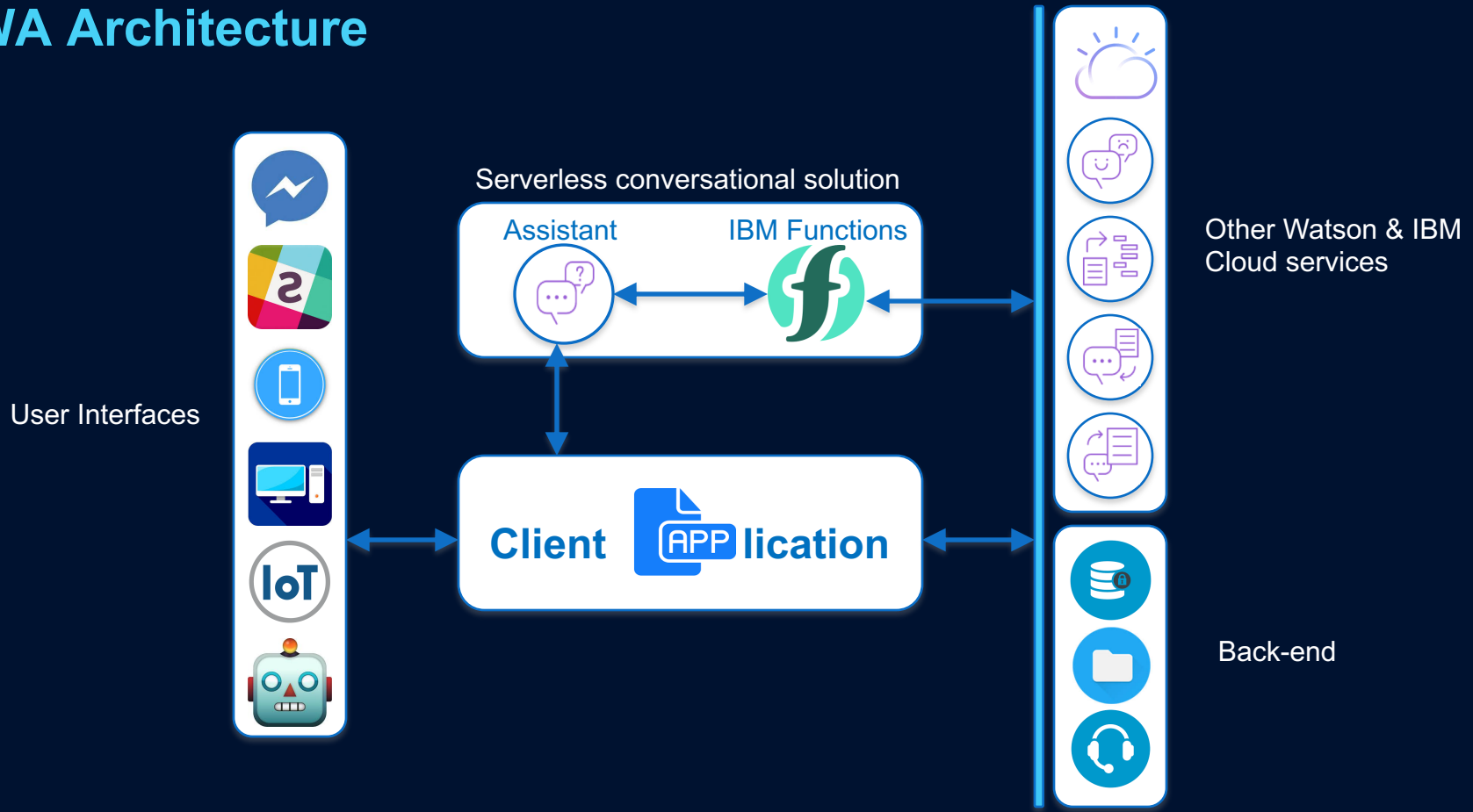
Context

```
{
  "context": {
     "{any_objects}": " {any_value}"
   },
   "actions": [ {
     "name":"<actionName>",
     "type":"client | server",
     "parameters": {
       "<parameter_name>":"<parameter_value>",
       "<parameter_name>":"<parameter_value>" },
     "result_variable": "<result_variable_name>",
     "credentials": "<reference_to_credentials>"
   } ],
   "output": {
     "text": [ "text one" ],
     "generic": [ object ],
     "{action}": " {any_value}",
   }
}
```

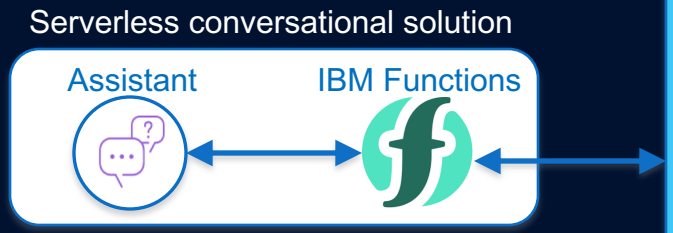
Ouput can be a way to provide details to your application

- To callout any APIs
- To show a form
- To display buttons
- To display any useful information with a custom format

WA Architecture

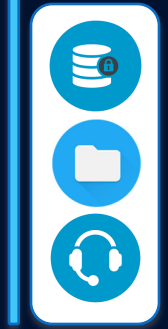


Enrichment



Intents, entities, variable context passed to **IBM Cloud Functions** for business logic implementation

Other Watson & IBM Cloud services



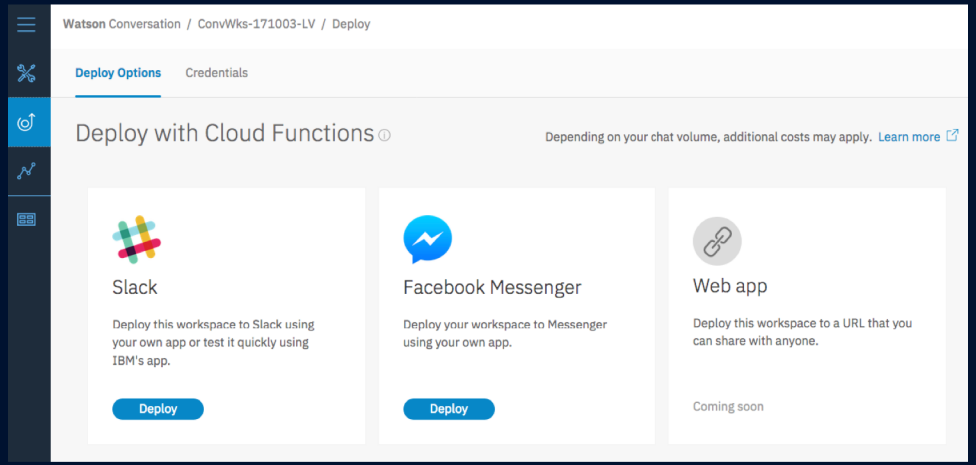
Back-end

Currently, you can call an action from a Cloud Functions instance that is hosted in the US South region from Conversation instances that are hosted in the US South region.

IBM Cloud Functions

- Overview
  - Built on OpenWisk
  - Fully Managed public cloud offering
  - Out of the bow API Gateway support
- Integrations
  - Prebuilt integration with IBM Cloud Services: MessageHub, AppConnect, Couldant, Watson Conversation

Cloud Functions Features

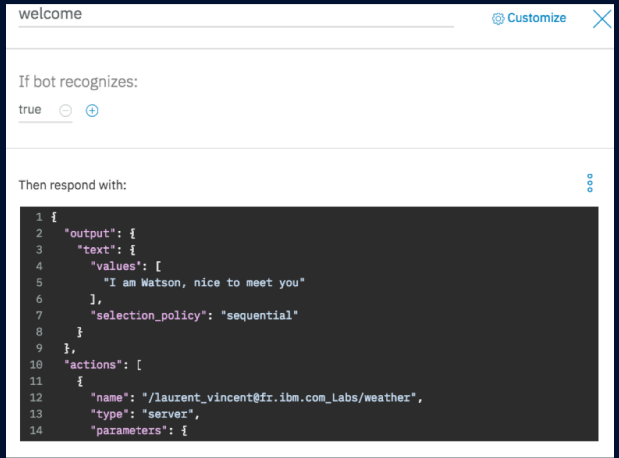


Messaging Service Integration

- Rapid setup of Actions that mediate between messaging applications and the Watson Conversation services
- Support deployment of test chatbots to

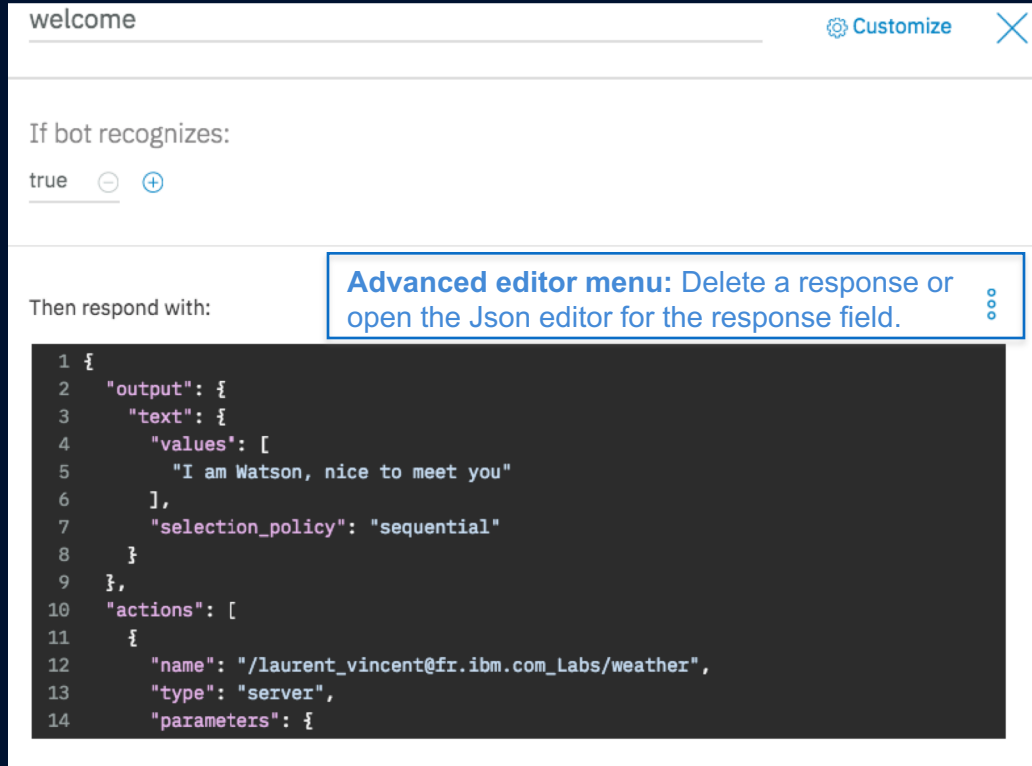
Watson Dialog Integration

- Call out to Actions directly from a Watson Conversation dialog
- Via embedded JSON-based declarations of actions, parameters , and results

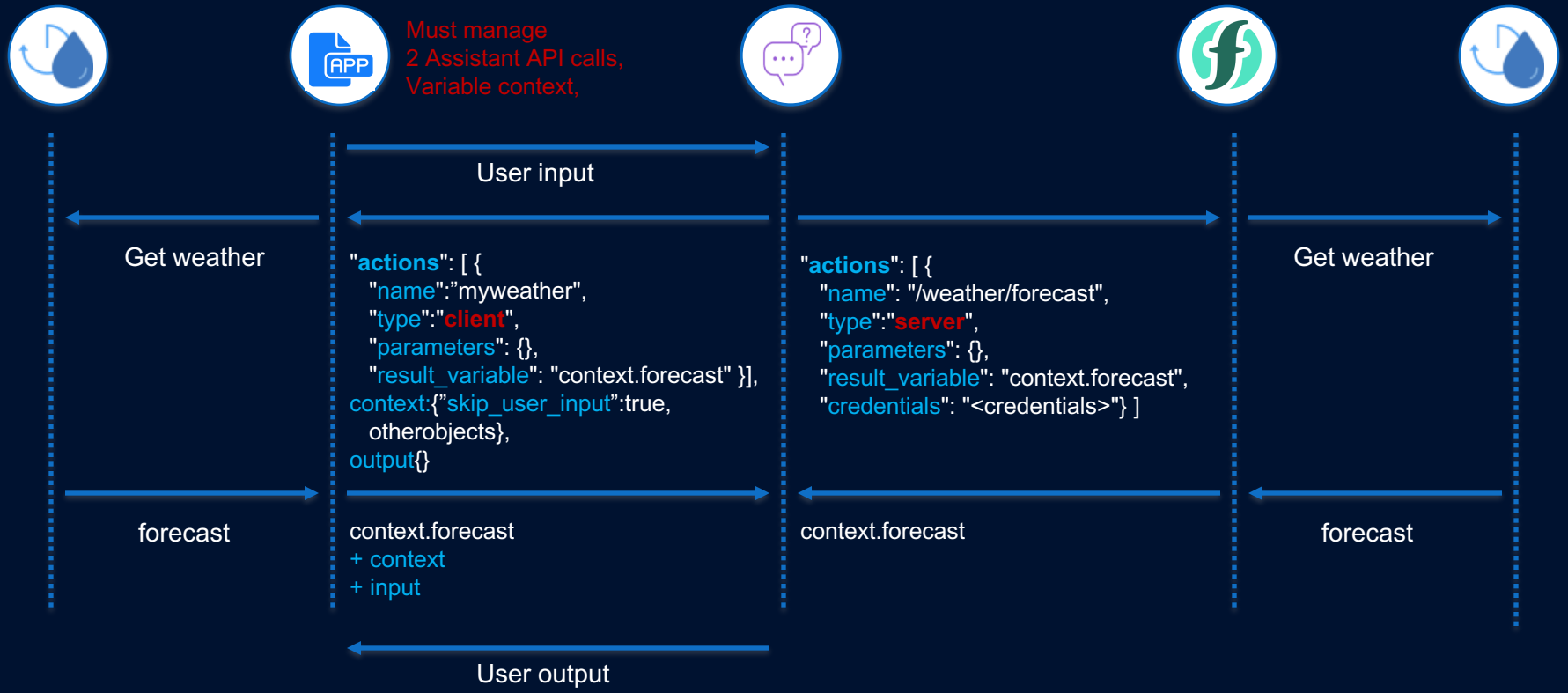


Advanced Editor: JSON editor

```
{
  "context": {
     "{any_objects}": " {any_value}"
   },
   "actions": [ {
      "name":"<actionName>",
      "type":"client | server",
      "parameters": {
         "<parameter_name>":"<parameter_value>",
         "<parameter_name>":"<parameter_value>" },
      "result_variable": "<result_variable_name>",
      "credentials": "<reference_to_credentials>"
   } ],
   "output": {
      "text": [ "text one" ],
      "generic": [ object ],
      "{action}": " {any_value}",
   }
}
```



Making calls from a dialog node



Let’s get started !