

Chatbot Deployment with IBM Cloud Watson Assistant

Phase-5

Abstract:

This guide outlines the process of deploying a chatbot using IBM Cloud Watson Assistant. It covers account setup, chatbot creation, integration with platforms, and optimization for real-world applications. Practical tips, use cases, and troubleshooting advice are provided for a streamlined deployment experience.

1. IBM Watson Assistant:

Watson Assistant is at the head of the tech industry of conversations through artificial intelligence. Platform is simple and intuitive, with extremely well-created documentation. In terms of costs, it offers a **free plan** (Lite plan), which allows 10,000 messages per month via chatbot, and the ability to integrate the bot with web services.

2. Setup:

- To access the Watson Assistant platform, but also other services, you need to create an account on **IBM Cloud**: <https://cloud.ibm.com/registration>. After you have created your account and logged in, select "Watson" from the side menu, and in the dashboard displayed you will

find in the *"Getting Started"* section, the option *"Build a chatbot"*.

- From here you will be directed to the initial configuration page of the chatbot (pricing plan, region, service name).
- When you're ready, click *"Create"* and that's it! The bot is created and ready to be configured as you wish, through the dedicated platform.

3.Design a hierarchical flow:

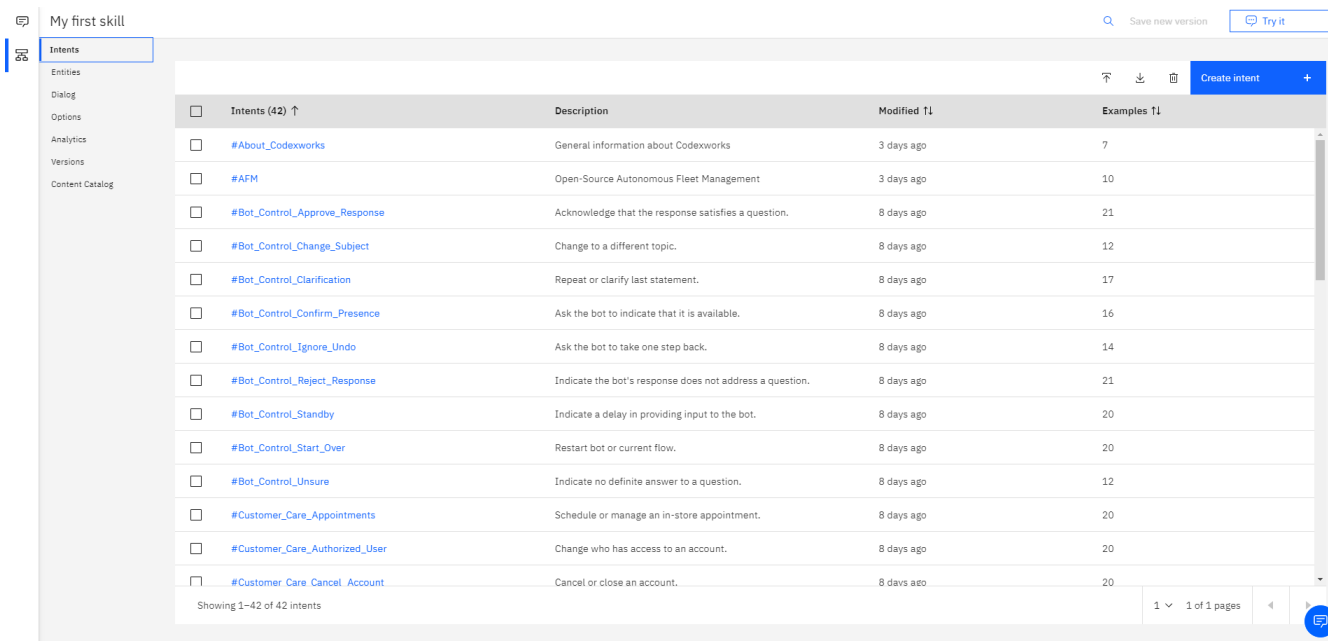
- The process of learning for chatbot involves the development of several **dialogue skills** (up to 5 in the Lite version).
- A dialogue skill consists of **Intents**, **Entities** and **Dialogs**, these 3 components being the basis for learning of virtual assistant, through the IBM Watson Assistant platform.

4.Intent:

- An intent is a **purpose or intention expressed by the user**, such as asking a question - *"How do I schedule a call?"*, or expressing an intention - *"I would like to pay the bill"*.
- By recognizing the desire of the user, **the virtual assistant can choose the appropriate dialogue flow**, and can

formulate an appropriate response. Most of the time, the user's intention is cataloged by the bot in several categories, giving a probability to each option.

- In the image below you can see several intents defined for the first dialog skill of the bot:



<input type="checkbox"/>	Intents (42) ↑	Description	Modified T1	Examples T1
<input type="checkbox"/>	#About_Codexworks	General information about Codexworks	3 days ago	7
<input type="checkbox"/>	#AFM	Open-Source Autonomous Fleet Management	3 days ago	10
<input type="checkbox"/>	#Bot_Control_Approve_Response	Acknowledge that the response satisfies a question.	8 days ago	21
<input type="checkbox"/>	#Bot_Control_Change_Subject	Change to a different topic.	8 days ago	12
<input type="checkbox"/>	#Bot_Control_Clarification	Repeat or clarify last statement.	8 days ago	17
<input type="checkbox"/>	#Bot_Control_Confirm_Presence	Ask the bot to indicate that it is available.	8 days ago	16
<input type="checkbox"/>	#Bot_Control_Ignore_Undo	Ask the bot to take one step back.	8 days ago	14
<input type="checkbox"/>	#Bot_Control_Reject_Response	Indicate the bot's response does not address a question.	8 days ago	21
<input type="checkbox"/>	#Bot_Control_Standby	Indicate a delay in providing input to the bot.	8 days ago	20
<input type="checkbox"/>	#Bot_Control_Start_Over	Restart bot or current flow.	8 days ago	20
<input type="checkbox"/>	#Bot_Control_Unsure	Indicate no definite answer to a question.	8 days ago	12
<input type="checkbox"/>	#Customer_Care_Appointments	Schedule or manage an in-store appointment.	8 days ago	20
<input type="checkbox"/>	#Customer_Care_Authorized_User	Change who has access to an account.	8 days ago	20
<input type="checkbox"/>	#Customer_Care_Cancel_Account	Cancel or close an account.	8 days ago	20

Showing 1–42 of 42 intents

1 of 1 pages

Create intent +

An advantage of using this platform is that it puts available a **predefined range of intents**, which can be used for free. (check *Content Catalog* section). Thus, the **Bot Control** category, and all associated intents, were added and learned by our assistant.

How exactly does an intent look like on the platform? You can see in the image below how an intent was defined for the flow of data responsible for information about **AFM** (Autonomous Fleet Management).

← | #AFM

Intent name
Name your intent to match a customer's question or goal

#AFM

Description (optional)
Open-Source Autonomous Fleet Management

User example
Add unique examples of what the user might say. (Pro tip: Add at least 5 unique examples to help Watson understand)

[type a user example here]

Add example

☐ User examples (10) ↑

<input type="checkbox"/>	AFM
<input type="checkbox"/>	Autonomous Fleet Management
<input type="checkbox"/>	Do you have any open source project?
<input type="checkbox"/>	Do you have any product?
<input type="checkbox"/>	I heard that you have a internal product idea. Tell me more
<input type="checkbox"/>	Tell me about AFM project idea

Showing 1–10 of 10 examples

Intents (using IBM Watson Assistant API):

import requests

Define Intent

```
intent_data = {
    "intents": [
        {
            "intent": "greetings",
            "examples": [
                {"text": "Hello"},
                {"text": "Hi"},
                # Add more examples ]
            ],
            # Add more intents
        ]
    ]
}
```

```
}  
  
respon=requests.post(  
"https://api.usssouth.assistant.watson.cloud.ibm.com/instances/{instance_id}/v2/assistants/{assistant_id}/intents",  
  
headers={"Authorization": "Bearer {api_key}", "Content-Type":  
"application/json"}, json=intent_data )
```

5.Entities:

- An entity is **a class of data/information that is relevant to a user's purpose in expressing intent**. By recognizing the entities, the virtual assistant can choose specific actions, in accomplish an intention.
- For example, a user may want to know more about **CodexWorks** (*"Tell me more about Codexworks"*). At the same time, he may want information about the history of CodexWorks. (*"Tell me more about Codexworks's history"*). There is a difference between the two answers that must be provided. **History is an entity**, changing the context of the intention provided. For the user, the history becomes relevant, not the general information about CodexWorks.
- In the image below you can see some entities defined within the platform:

	Entity (4) ↑	Values	Modified ↑
<input type="checkbox"/>	@bot_age	age	8 days ago
<input type="checkbox"/>	@bot_description	who are you	8 days ago
<input type="checkbox"/>	@bot_profile_image	bot's profile image, picture with you	3 days ago
<input type="checkbox"/>	@history	history	3 days ago

Showing 1-4 of 4 entities

1 1 of 1 pages

Entities (using IBM Watson Assistant API):

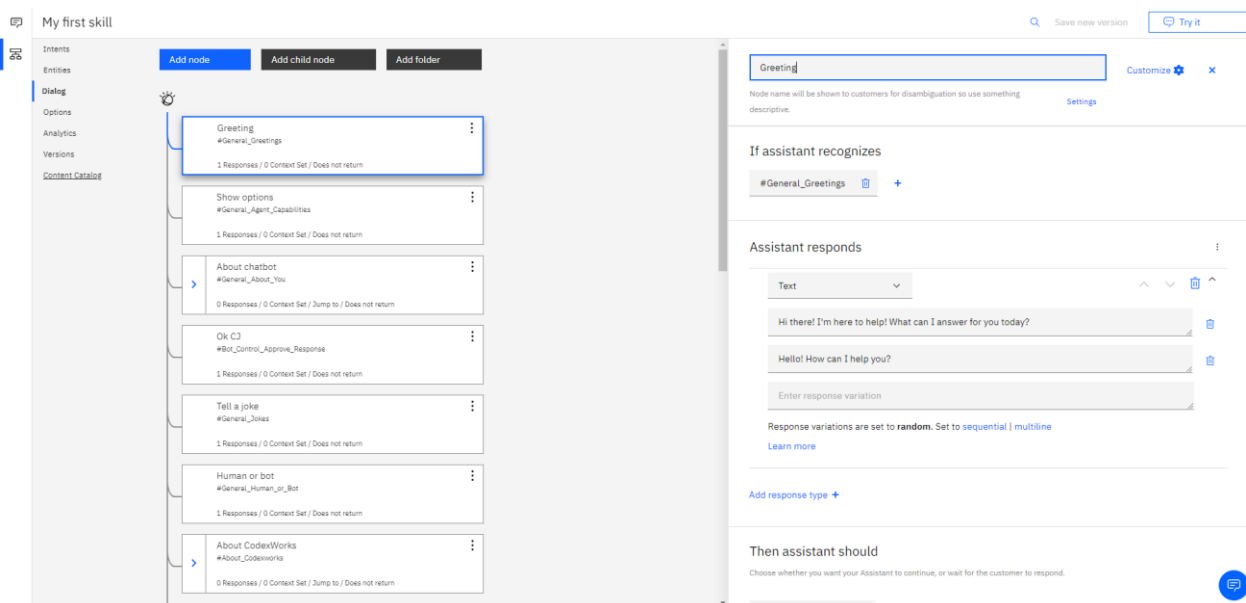
Define Entity Type

```
entity_data = {
    "entities": [
        {
            "entity": "location",
            "values": [
                {
                    "value": "New York",
                    "synonyms": ["NY", "NYC"]
                },
                # Add more values
            ]
        },
        # Add more entity types
    ]
} response = requests.post(
    "https://api.usouth.assistant.watson.cloud.ibm.com/instances/{instance_id}/v2/assistants/{assistant_id}/entities",
```

```
headers={"Authorization": "Bearer {api_key}", "Content-Type":  
"application/json"}, json=entity_data )
```

6.Dialogs:

- A dialog is **the component that uses discovered Intents and Entities based on inputs provided by user, in order to find and provide an appropriate response**. The dialogs have a linear tree structure. Thus, a branch is created for each intent (or group of intents), together with the associated entities. Each dialog offers the possibility to prepare several answers, being provided only one answer (chosen sequentially, or randomly).
- The image below shows a part of the **tree structure** of the dialogues for our bot, in which a well-established order of them can be observed. On the right you can see how the dialog for the **Greetings** use case is defined, and the possible answers that the bot can offer, if it recognizes in the user's input a form of greeting.



Dialog Nodes (using IBM Watson Assistant API):

Define Dialog Node

```
dialog_data = {  
  "dialog_nodes": [  
    { "dialog_node": "greetings",  
      "conditions": "#greetings",  
      "output": {  
        "generic": [  
          { "response_type": "text",  
            "values": [ { "text": "Hello! How can I assist you today?" } ] } ] },  
      "values": [ { "text": "Hello! How can I assist you today?" } ] } ] } },
```

Add more dialog nodes

```
] }  
}
```

```
response = requests.post(  
  "https://api.ussouth.assistant.watson.cloud.ibm.com/instances/{instance_id}/v2/assistants/{assistant_id}/dialog_nodes",  
  headers={"Authorization": "Bearer {api_key}", "Content-Type":  
    "application/json"}, json=dialog_data )
```

Interact with your chatbot:

Test:

- After defining a few dialogues, the bot can already be successfully tested! His training is done **in real time**, immediately after adding a new element, of any type.

- In the upper right corner you can see the *Try it* button that opens the chat for the bot you created, with which you can interact and see in which category maps the input (question or intention) provided by the user.

Deploy:

- An extremely valuable advantage of these platforms is that **the deployment process is very simple**. To create a deployment channel, go to the main panel of the platform and select *Integrations*. From there you will be able to create a new integration, with whatever service you want. The most common is Web chat.

Web chat

Close Saved

Integration name
Web chat

Style Home Screen Live agent Security Embed

Customize your chat UI

Update the style to match your brand and your website. A developer can also add more advanced styling changes with code, [learn more](#).


Assistant's name as known by customers

CJ Assistant

Primary color
Chat header
#FFFFFF

Secondary color
User message bubble
#1111a1

Accent color
Significant and interactive objects
#01060c



Change avatar image

Restart conversation

CJ Assistant

Hi! I'm a virtual assistant. How can I help you today?

Type something...

Get started

[About CodexWorks](#)

[Business sectors](#)

Step 1: Integrate with Facebook Messenger:

To integrate your chatbot with Facebook Messenger, follow these steps:

1. Set Up a Facebook App and Page

- Create a Facebook App on the [Facebook for Developers portal] (<https://developers.facebook.com/apps/>).
- Create a Facebook Page if you don't already have one.

2. Obtain App Credentials

- Get your App ID and App Secret from the Facebook App settings.

3. Set Up Webhooks

- Configure a webhook for your Facebook App to receive messages. This webhook will forward messages to your chatbot.

Code:

```
# This is typically done through the Facebook Developer Portal
# You'll need to provide a URL for your webhook endpoint (e.g., using
Flask, Django, etc.)
```

```
# Verify the request using the provided verification token
```

```
# Example Flask endpoint
```

```
@app.route('/webhook', methods=['GET'])
```

```
def verify_webhook():
```

```
    verify_token = 'your_verification_token'
```

```
    if request.args.get('hub.verify_token') == verify_token:
```

```
        return request.args.get('hub.challenge')
```

```
    return 'Invalid verification token'
```

```
4. Implement Webhook Endpoint (using a web server or cloud
function)
```

- Create an endpoint to handle incoming messages from Facebook. This will be the URL you provided in the webhook setup.

5. Process Incoming Messages

- Parse the incoming messages from Facebook and send them to your Watson Assistant using its API.

Code:

```
# Parse incoming messages and extract sender ID and message text
```

```
# Send the message text to Watson Assistant for processing
```

```
# Example Flask endpoint
```

```
@app.route('/webhook', methods=['POST'])
```

```
def receive_message():
```

```
    data = request.get_json()
```

```
    for entry in data['entry']:
```

```
        for messaging_event in entry['messaging']:
```

```
            sender_id = messaging_event['sender']['id']
```

```
            message_text = messaging_event['message']['text']
```

```
            # Send message_text to Watson Assistant for processing
```

6. Send Responses to Facebook Messenger

- Once you receive a response from Watson Assistant, send it back to Facebook Messenger using the Messenger API.

Code:

```
# Once you receive a response from Watson Assistant, send it back  
to Facebook Messenger
```

```
def send_message(sender_id, message_text):
```

```
data = {  
    'recipient': {'id': sender_id},  
    'message': {'text': message_text}  
}  
response = requests.post(  
    'https://graph.facebook.com/v13.0/me/messages',  
    params={'access_token': 'your_page_access_token'},  
    json=data  
)
```

Step 2: Integrate with Slack:

To integrate your chatbot with Slack, follow these steps:

1. Set Up a Slack App

- Create a Slack App on the [Slack App Directory](<https://api.slack.com/apps>).

2. Obtain App Credentials

- Get your Slack App credentials, including the Client ID, Client Secret, and Verification Token.

3. Set Up OAuth & Permissions

- Configure OAuth & Permissions in your Slack App settings to allow your app to interact with Slack workspaces.

4. Implement OAuth Flow (if necessary)

- If you're using OAuth, implement the OAuth flow to authenticate your app with Slack.

5. Set Up Event Subscriptions

- Configure event subscriptions to receive messages from Slack.

Code:

You'll need to set up an event subscription for message events in your Slack App settings

Define an endpoint to receive Slack events

Example Flask endpoint

```
@app.route('/slack/events', methods=['POST'])
```

```
def slack_events():
```

```
    data = request.get_json()
```

```
    if 'event' in data and 'type' in data['event'] and data['event']['type'] == 'message':
```

```
        user_id = data['event']['user']
```

```
        message_text = data['event']['text']
```

```
        # Send message_text to Watson Assistant for processing
```

6. Process Incoming Messages

- Parse incoming messages from Slack and send them to your Watson Assistant using its API.

Code:

Parse incoming messages and extract user ID and message text

Send the message text to Watson Assistant for processing

Example Flask endpoint (continued)

```
@app.route('/slack/events', methods=['POST'])
```

```
def slack_events():
```

```
data = request.get_json()

if 'event' in data and 'type' in data['event'] and data['event']['type']
== 'message':

    user_id = data['event']['user']

    message_text = data['event']['text']

    # Send message_text to Watson Assistant for processing
```

7. Send Responses to Slack

- Once you receive a response from Watson Assistant, send it back to Slack using Slack's API.

Code:

Once you receive a response from Watson Assistant, send it back to Slack

```
def send_message_to_slack(user_id, message_text):

    payload = {

        'token': 'your_bot_user_access_token',

        'channel': user_id,

        'text': message_text

    }

    response =

requests.post('https://slack.com/api/chat.postMessage',

data=payload)
```

Step 3: Refine Responses

Refining responses involves:

- **Context Management:** Ensure the chatbot maintains context for multi-turn conversations.
- **Natural Language Processing (NLP):** Continuously train and improve your Watson Assistant to understand user queries better.
- **Fallback Mechanism:** Enhance the fallback responses to handle ambiguous or unclear user input.